Ludwig Maximilians Universität München
Institut für Informatik

Diplomarbeit

Aktive Temporale Datenbanken

von
Jan Dünnweber

Aufgabenstellung und Betreuung:
Prof. Dr. Hans Jürgen Ohlbach

München, den 30.4.2003

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

München, den 30.4.2003

Inhaltsverzeichnis

1 Einleitung : Das Konzept der Web-Temporal Notions Database 3

2 Zugriff auf temporale Daten mit WebTNDB 6
2.1 Die Startseite 6
2.1.1 Der Zugang zum gesicherten Bereich von WebTNDB . . . 7
2.1.2 WebTNDB Sicherheitsmechanismen 7
2.2 Die Registrierung eines neuen Benutzers 8
2.2.1 Das Anlegen eines neuen Benuterkontos 9
2.2.2 Die Aktivierung eines neuen Benutzerkontos 10
2.3 Das Anfragesystem 10
2.3.1 Anfragebäume 11
2.3.2 Parameterbäume 11
2.3.3 Der Anfrageeditor 12
2.3.4 Anfrageintervalle 15
2.4 Datenverarbeitung mit WebTNDB 16
2.4.1 Die Datentabelle 17
2.4.2 Das Einfügen neuer Datensätze 17
2.4.3 Das Bearbeiten von Datensätzen 18

3 Implementierung 20
3.1 Die Architektur von WebTNDB 20
3.2 Die Datenzugriffsschicht 22
3.2.1 Der Proxy-Server 22
3.2.2 Das Persistence Framework 23
3.2.3 Zusammenfassung 33
3.3 Die Anwendungsschicht 34
3.3.1 Die interne Darstellung der Bäume 34
3.3.2 Die grafische Darstellung der Bäume 40
3.3.3 Die Selektionsfilter 41
3.3.4 Zusammenfassung 44
3.4 Die Präsentationsschicht 45
3.4.1 Struts und WebTNDB 47
3.4.2 Zusammenfassung 52

4 Ausblick 53

A Die Installation von WebTNDB Servern 55
A.1 Die Datenbank 55
A.2 Der WebCAL Server 62
A.3 Der Proxy-Server 62
A.4 Der Applikations-Server 65

B Jakarta Struts 69
B.1 Struts Komponenten 70
B.2 Struts Features 81

1 Einleitung : Das Konzept der Web-Temporal Notions Database

Unter einer aktiven temporalen Datenbank ist ein System zur Beschreibung, Speicherung und Wiedergewinnung von Zeitdaten, also Terminen und Intervallen, zu verstehen. Die Aktivität des Systems besteht u.a. darin, daß der Zugriff auf bestimmte Zeitinformationen Berechnungen erfordert, während andere aus dem statischen Datenspeicher gelesen werden können. Ein fest gespeicherter Termin läßt sich beispielsweise aus einer Tabelle ablesen, der Zeitpunkt des nächsten Sonnenuntergangs zu einer gegebenen Position oder der Termin des Osterfests in einem gegebenen Jahr muß durch einen astronomischen Algorithmus bestimmt werden.

Für den Anwender ist die Aktivität des Systems beim Zugriff auf alle Daten transparent, d.h. seine Anfragen liefern tabellarische Ausgaben, denen man nicht mehr ansieht welche Zeilen Datensätze darstellen, die aus dem statischen Speicher gelesen wurden und welche durch Berechnungen bestimmt wurden. Ein Administrator kann dagegen beim Speichern eines Datensatzes entscheiden, ob dieser mit einem fixen Termin in der Datenbank abgelegt werden soll, oder, für die Bestimmung des zugehörigen Zeitpunkts ein Algorithmus nötig ist.

Thema dieser Diplomarbeit ist das System WebTNDB : Ein Prototyp für ein aktives Web-Datenbankmanagementsystem.

WebTNDB steht für Web-**t**emporal **n**otions **d**ata**b**ase und bezeichnet ein im Rahmen dieser Arbeit entwickeltes System, das auf relationalen Datenbanken aufsetzt, als zwischengeschaltete Ebene aber aktiv Anfragen interpretiert und angeforderte Daten aufbereitet.

Neben der Unterscheidung gewöhnlicher Anfragen von solchen, die Berechnungen erfordern, profitiert der Anwender durch zwei weitere Vorteile beim Datenzugriff von der Verwendung des aktiven Systems :

Der erste Vorteil ist, daß die Gültigkeit der Information in der Datenbank nicht von lokal verwendeten Zeit- und Kalendersystemen abhängig ist, wie es bei temporalen Daten sonst der Fall ist. WebTNDB bildet Zeitinformation in den gepeicherten Datensätzen intern auf eine Referenzzeitlinie ab, um sie von dort auf Repräsentationen in den vom Benutzer spezifizierten Kalendersystemen und Zeitzonen zu projezieren. Den gleichen Mechanismus verwendet WebTNDB, um dem Benutzer die Spezifikation der Anfrageintervalle in verschiedenen Kalender- und Zeitsystemen zu ermöglichen. Die hierzu benötigte Datumsarithmetik muß eine Vielzahl von Irregularitäten berücksichtigen. Aufgrund von Schaltsekunden, lokal variierenden Schemata für Zeitumstellungen und weiteren kulturell oder historisch bedingten Unterschieden der unterstützten Zeitsysteme stellt der beschriebene Abbildungsmechanismus ein komplexes Problem dar.

WebTNDB verwendet außer den von der Java-Technologie bereitgestellten Werkzeugen den von Hans Jürgen Ohlbach entwickelten Open-Source Web-Server WebCal [Ohl02]. Die für bestimmte Anfragen erforderlichen astronomischen Berechnungen lassen sich auf Basis der dort verwendeten Zeitpartitionierung in den WebCal Server implementierten. WebCal ist in C++ programmiert

und stellt die grundlegenden Datentypen für die Speicherung von Zeiteinheiten sowie Abbildungsmechanismen für Zeitkoordinaten in den Klassen `TimeUnit` und `abstract calendar` (bzw. angepassten Subklassen[1]) bereit. Für die Datenübertragung zwischen WebTNDB und WebCal verwenden diese Systeme die in C++ und Java verfügbaren Bibliotheken für Socket-Kommunikation, die das im Internet übliche Transportprotokoll TCP/IP einsetzen. Für den Zugriff auf die in WebCal verfügbaren Dienste interpretiert WebCal ein darüber liegendes propriäteres Anwendungsprotokoll ([Ohl02, S. 11]).

Der zweite Vorteil, den WebTNDB bietet, ist die Unabhängigkeit von den Schemata, der in den darunterliegenden relationalen Datenbanken verwendeten Tabellen. Anfragen an WebTNDB werden nicht in der für relationale Datenbanken üblichen Anfragesprache SQL (**S**tructured **Q**uery **L**anguage), sondern mittels eines grafischen Anfragemechanismus gestellt. WebTNDB wertet Anfragefunktionen aus, die dem Anwender eine baumartige Struktur der Daten suggerieren. Die Baumsicht auf die Daten ist von Änderungen der Tabellenschemata, falls z.B. zusätzliche Indextabellen für effizientere Zugriffe auf temporale Daten ([KPS00, KPS01]) eingeführt würden, nicht betroffen. Außerdem werden die zwischen den Datensätzen gegebenen Beziehungen in der Baumdarstellung natürlicher wiedergegeben, als durch Referenzen in Tabellen. Auch das objektorientierte Substitutionsprinzip (Polymorphismus), das sonst nur objektrelationale Datenbanken unterstützen, besteht für Beziehungen, die in den Baumdiagrammen durch Verbindungen zwischen Vater- und Kindknoten dargestellt werden, um Generalisierungen oder Spezialisierungen zu beschreiben.

Ein Editor ermöglicht es, Funktions- und Parameterbäume grafisch zu erstellen und vom System in SQL-Anfragen übersetzen zu lassen. Für diese Übersetzung verwendet WebTNDB verschiedene Meta-Datentabellen, die sich vom Anwender erweitern lassen, um neue Typen von Anfrage und Parameterbäumen bereitzustellen. Neben der TNDB-API (**A**pplication **P**rogramming **I**nterface), einer Bibliothek für den Zugriff auf Daten und Meta-Datentabellen, ist ein Proxy-Server Bestandteil von WebTNDB. Dieser ermöglicht den Zugriff auf WebCal-Server über Terminalprogramme wie Telnet und zusätzlich die Auswertung in WebTNDB gepeicherter Anfragefunktionen. Der grafische Baumeditor ist in ein weiteres Programm integriert, das auch für die Speicherung und Verwaltung der Anfrage- und Parameterbäume zuständig ist. Dabei handelt es sich um das Web-Interface, das zugleich die namensgebende und aufwendigste Komponte von WebTNDB ist : Eine J2EE-Applikation[2], die mittels HTTP

[1] Einfache Zeitumstellungen innerhalb eines Kalendersystems lassen sich in speziellen Konfigurationsdateien (sog. DLST-Files) konfigurieren, so daß hierzu keine neue Subklassenbildung und dadurch bedingte Rekompilierung erforderlich ist.

[2] J2EE steht für **J**ava **2** **E**nterprise **E**dition und ist die Sammelbezeichnung für eine Vielzahl von Technologien für das Erstellen mittlerer bis großer Softwaresysteme auf der Basis von Java-Technologie. Die aktuelle J2EE Version 1.3 umfaßt als Ergänzungen zur **J**ava **2** **S**tandard **E**dition (J2SE 1.4.1) u.a. die Servlet API 2.3 und Java-Server-Pages 1.2, mit deren Hilfe sich dynamische Web-Komponenten entwickeln lassen, die Schnittstellenspezifikation JNDI 1.2 für den Zugriff auf Verzeichnisdienste, die Email-Bibliotheken JavaMail 1.2, die JDBC 3.0 Pakete für den Zugriff auf relationale Datenbanken, das server-seitige Komponentenmodell EJB 2.0 u.v.a.m.

(**H**ypertext **T**ransfert **P**rotocol) von einem beliebigen Internet-Client genutzt werden kann, der das im WWW (**W**orld **W**ide **W**eb) übliche Anwendungsprotokoll sowie die Darstellung und Eingabeverarbeitung von HTML-Formularen (**H**ypertext **M**arkup **L**anguage) unterstützt, also von jedem Endgerät, das einen gewöhnlichen Web-Browser bereitstellt, wie er inzwischen auf jedem Standard PC oder verschiedenen Geräten zur mobilen Kommunikation verfügbar ist. Weitere Anforderungen für den Zugriff auf diese Schnittstelle bestehen keine. So wird etwa kein spezielles client-seitig verwendetes Betriebssystem vorausgesetzt oder gar zusätzliche Software benötigt. Die gesamte Aktivität findet serverseitig statt. Client-seitig wird nur die Darstellungsfunktionalität des dort verwendeten Geräts genutzt.

Die Web-Applikation kann Anfragen auswerten und für administrative Zugriffe, wie das Bearbeiten, Einfügen oder Löschen von Daten und, sofern die entsprechenden Berechtigungen bestehen, auch von Meta-Daten oder Benutzerkonten, genutzt werden. Über den WebCal-Proxy-Server können außer dem direkten Zugriff auf alle Funktionen von WebCal, Anfragen an mehrere verteilte Datenbankserver gestellt werden, auf denen temporale Daten über das WebTNDB-Online-Inferface verwaltet werden.

Die Diplomarbeit ist in zwei Teile gegliedert. Der erste Teil „Zugriff auf temporale Daten mit WebTNDB“ erklärt die Funktionsweise von WebTNDB aus der Sicht des Benutzers und kann somit als Bedienungsanleitung für die Web-Applikation und den Proxy-Server dienen. Hier werden der Anfrageeditor und die Erweiterungen zu WebCal bei der Verwendung des WebTNDB-Proxys für den Anwender beschrieben. In Anhang A folgt eine Installationsanleitung für Administratoren, die selbst WebTNDB-Server betreiben wollen.

Der Folgeteil, „Implementierung“, erklärt die dreischichtige Architektur von WebTNDB. Die TNDB-API, bestehend aus der Datenzugriffsschicht und der Anwendungsschicht auf der die Komponenten für die Anfrageauswertung bereitgestellt werden, umfasst die beiden unteren Schichten. Das Web-Interface ist die Präsentationsschicht. Dieses Kapitel stellt die verwendeten Frameworks und Technologien sowie die internen Komponenten und eingesetzen Verfahren von WebTNDB vor und soll Programmierern die nötigen Informationen für Weiterentwicklungen des Systems liefern.

Der erste Teil setzt dabei außer Grundkenntnissen über Datenbankbanksysteme an Vorwissen nicht mehr voraus, als daß der Leser damit vertraut ist, wie sich mithilfe der jeweils verwendeten Hard- und Software Internetinhalte abrufen lassen. Auf Implementierungsdetails wird hier nur in dann eingegangen, wenn das für das Verständnis der Funktionsweise der dargestellten Systemkomponenten für den Benutzer hilfreich ist.

Im zweiten Teil werden Kenntnisse der Programmiersprache Java sowie der Java-Standard-APIs vorausgesetzt. Eine Einführung in die Prinzipien des Front-End Frameworks Jakarta Struts, auf dem die Präsentationsschicht von WebTNDB basiert, ist in Anhang B zu finden..

Konzeptuelle Ideen für Weiterentwicklungen schließen diese Arbeit ab.

2 Zugriff auf temporale Daten mit WebTNDB

Um WebTNDB zu benutzen, ist weder eine Software-Installation noch eine spezielle Konfiguration am Client-Gerät notwendig. Es muß nur die Internetadresse des Applikations-Servers in Form einer URL (**U**niform **R**esource **L**ocator) ins Adressfeld des verwendeten Browserprogramms eingegeben werden. Das kann, sofern diese bekannt ist, die numerische IP-Adresse des Server-Rechners sein oder, wenn ein DNS (**D**omain **N**ame **S**ervice) verfügbar ist, der dem Server-Domain-Namen seine IP-Adresse zuordnen kann, ein Domain-Name. Zum Zeitpunkt der Erstellung dieser Arbeit ist ein Test-System unter dem Domain-Namen *.webtndb.mine.nu verfügbar[3], das beim Lesen dieses Kapitels aufgerufen werden sollte, um die dargestellten Schritte selbst ausprobieren und einfach nachvollziehen zu können. Das folgende Tutorial geht von der Verwendung eines Standard PCs, der Web-Inhalte in einem Browser-Fenster auf dem Bildschirm anzeigt und Benutzereingaben über Tastatur und Maus entgegennimmt, aus. Wird ein anderes Client-Gerät verwendet, so gelten die Beschreibungen natürlich für die dort vorhandenen Bedienelemente analog.

Die Bedienung von WebTNDB gestaltet sich wie bei allen Web-Applikationen (Online-Banking, E-Commerce, etc.) durch Navigation von einer HTML-Seite zur nächsten, wobei die Seiten jeweils Formulare darstellen, auf denen der Benutzer Eingaben für das System vornehmen kann. Die einzelnen Seiten sind, insbesondere wenn sie direkt aufeinander folgen, oft bis auf minimale Änderungen, wie einzelne neue Ausgaben, sehr ähnlich. Diese Ähnlichkeit ist beabsichtigt, um ein konsistentes Erscheinungsbild zu erreichen und den Eindruck zu erzeugen, im Browserfenster würde die Benutzeroberfläche einer Anwendung dargestellt, die je nach Systemzustand andere Bestandteile beinhalten kann. Daß technisch bedingt eine Vielzahl von prinzipiell unabhängigen, dynamisch generierten Formularen angezeigt werden muß, und jede Interaktion, wie ein Mausklick auf eine Schaltfläche oder einen Knoten im grafischen Baumeditor, auf eine neue Seite führt, sollte nicht mehr zu erkennen sein.

2.1 Die Startseite

Nach erfolgreichem Aufruf präsentiert WebTNDB zunächst die Startseite, die neben kurzen Informationen über Versionsnummer, Nutzungsbedingungen und Autor einen Hyper-Link zum Registrierungsformular für neue Benutzer sowie eine Schaltfläche enthält, die einen zum Anmeldeformular für registrierte Anwen-

[3]Dazu werden die DNS-Server der Firma DynDNS.org verwendet. Bei DynDNS.org lassen sich u.a. Domainkennungen, die auf .dyndns.org, .homeip.net oder eben .mine.nu enden, kostenfrei anmelden. Wie lange dieser Dienst in dieser Form noch fortbesteht, läßt sich natürlich nicht vorhersagen, nach den auf der Web-Seite www.dyndns.org verfügbaren Informationen ist aber vorerst nicht mit einer Einstellung dieses Dienstes zu rechnen. DynDNS löst Domainnamen mit sogenannten WildCards auf. Für die Domain *.webtndb.mine.nu bedeuted das, daß der Name webtndb.mine.nu mit einem beliebigen Präfix wie z.B. www, www2 oder pms immer auf die gleiche IP-Adresse abgebildet wird. Eine dem durchschnittlichen Internetanwender vertrautere Domainnamen-Endung als .mine.nu, wie z.B. .de, .com., .org oder .net ist dort auch erhältlich, diese gibt es aber nicht kostenfrei...

der führt. Die Startseite ist die einzige Seite von WebTNDB, die unverschlüsselt zum Anwender übertragen wird und von der Eingaben unverschlüsselt zum Server zurückgeschickt werden.
Das ist auch der eigentliche Zweck der Startseite : Der Aufruf des Programms ist ungesichert möglich, was erwünscht ist, da Standardbrowser beim Aufbau einer gesicherten Verbindung eine Warnmeldung ausgeben. Diese Meldung sollte möglichst nicht sofort beim Anwählen der ersten Seite, noch bevor diese überhaupt dargestellt wird, ausgegeben werden, um den Anwender nicht zu verunsichern.

Lokal ist bei den verwendeten relationalen Datenbanksystemen die Datensicherheit durch die Verwaltung von Benutzerrechten und Passwortabfragen gewährleistet. Ein Web-Interface muß darüberhinaus eigene Sicherheitsvorkehrungen treffen. Vor allem Zugangsdaten aber auch Informationen, die von dazu berechtigten Nutzern aus der Datenbank gelesen oder dort neu eingetragen werden, sollten nicht unverschlüsselt übertragen werden.

2.1.1 Der Zugang zum gesicherten Bereich von WebTNDB

Die einzige Eingabe, die der Anwender auf der Startseite vornehmen kann, ist die Auswahl des Registrierungsformulars oder des Anmeldeformulars mit der Maus. Diese Information ist nicht sicherheitsrelevant und kann folglich noch unverschlüsselt geschickt werden.

Der im folgenden beschriebene Registrierungsvorgang erfordert die Auswahl eines Passworts, mit dem sich der Benutzer vor jeder Sitzung vom System authentifizieren lassen muß, sowie die Eingabe einer gültigen Email-Adresse. Diese sensiblen Daten sollten bei der Übertragung zum Server gegen unberechtigtes Mitlesen oder gar Manipulation durch Dritte geschützt werden. Meldet sich ein registrierter Benutzer beim System an, so ist bei einigen auf die Anmeldung folgenden Vorgängen wie z.B. dem Editieren des Anfragebaums die Sicherheitsrelevanz der übertragenen Daten nicht mehr so offensichtlich. Da aber während einer laufenden Sitzung mit jeder Anfrage auch die sogenannte Session-ID zwischen Client- und Server-Rechner übermittelt wird, anhand der der Server mehrere, möglicherweise zeitgleiche Client-Sitzungen voneinander unterscheidet, sind nach dem Start einer Datenbanksitzung alle Interaktionen sicherheitsrelevant. Könnte ein unberechtigter Eindringling die Session-ID eines zugangsberechtigten Nutzers lesen, wäre es ihm damit möglich, Anfragen zu stellen, die für den Server nicht mehr von denen des registrierten Nutzers unterscheidbar wären. Folglich könnte er sich somit auch dessen Zugriffsberechtigungen auf die Datenbank selbst aneignen (session hijacking). WebTNDB überträgt deshalb alle Anfragen innerhalb einer Datenbanksitzung verschlüsselt.

2.1.2 WebTNDB Sicherheitsmechanismen

Nach der Auswahl von Anmelde- oder Registrierungsformular wird zunächst eine neue Sitzung gestartet und auf das Protokoll HTTPS umgeschaltet. HTTPS steht für **HTTP** over **S**SL und bezeichnet HTTP-Datenübertragung mit dem Zusatzprotokoll SSL. SSL steht für **S**ecure **S**ockets **L**ayer und ist ein Pro-

tokoll, das man sich im Netzwerk-Schichtenmodel ([Tan02, S. 37]) zwischen dem Anwendungsprokoll HTTP und dem Übertragungsprotokoll TCP/IP, das die unteren 4 Schichten beschreibt, vorzustellen hat. SSL verwendet bei der Netzwerkommunikation ein asymmetrisches Schlüsselpaar, bestehend aus einem privaten und einem öffentlichen Schlüssel. Mit dem privaten Schlüssel kann der Server Nachrichten an den Client kodieren, um sich eindeutig zu identifizieren : Gelingt es dem Client, die Nachricht mit dem öffentlichen Schlüssel zu dekodieren, weiß er, daß der Absender in Besitz des privaten Schlüssels ist und kann davon ausgehen, daß der absendende Server der ist, der er zu sein vorgibt. Den öffentlichen Schlüssel verwenden Clients auch, um ihre Anfragen an den Server zu kodieren, zu deren Dekodierung dieser wiederum im Besitz des privaten Schlüssels sein muß.

Für den Anwender ist das erfolgreiche Zustandekommen einer gesicherten Verbindung, durch das vorangefügte „https://“ im Adressfeld des Web-Browsers, sowie, bei den meisten Standard-Browsern, durch ein zusätzlich angezeigtes Symbol, meist ein geschlossenes Bügelschloß o.ä., zu erkennen[4]. Eine Darstellung der nötigen Schritte zur Serverkonfiguration für den Aufbau gesicherter Client-Verbindungen folgt im abschließenden Teil dieses Kapitels.

2.2 Die Registrierung eines neuen Benutzers

WebTNDB speichert für jeden Benutzer bestimmte Statusinformationen, die u.a. dazu dienen, daß ein Benutzer die von ihm in einer Sitzung erstellten Anfrage- und Parameterbäume sowie das spezifizierte Anfrageinterval in Folgesitzungen wiederverwenden kann, um sie weiter zu bearbeiten oder eine Anfrage für die in der Zwischenzeit möglicherweise veränderten aktuellen Datenbestände neu auswerten zu lassen. Weitere Daten, die WebTNDB für jeden Benutzer spezifisch speichern muß, sind z.B. das Paßwort für den Zugang zum gesicherten Bereich oder die Benutzerrolle mit der bestimmte Zugriffsberechtigungen assoziiert sind (s.u.). Intern verwaltet WebTNDB dazu sogenannte Benutzerkonten. Vor der ersten Benutzung des Systems muß ein neuer Nutzer daher zunächst ein neues Benutzerkonto anfordern, indem er das über den Registrierungs-Link auf der Startseite erreichbare Registrierungsformular ausfüllt.

[4]Der öffentliche Schlüssel wird innerhalb eines X.509-v3 Zertifikats (Ein Standard des Comité Consultarif International Téléphonique et Télégraphique CCITT [Go99, S.125]) an den Client übertragen. Beim ersten Aufbau einer gesicherten Verbindung mit WebTNDB vergleicht der Client-Browser dieses Zertifikat mit den ihm bekannten, bei denen es sich um die Zertifikate öffentlich anerkannter CAs (**C**ertificate **A**uthorities) handelt. Die bekanntesten dieser Behörden sind die US-amerikanischen Firmen VeriSign und Thawte, bei denen es, insbesondere für außerhalb der Vereinigten Staaten ansässige Personen, nicht ganz leicht und weiterhin recht kostspielig ist, ein gültiges Zertifikat zu erhalten. Das Testsystem verwendet stattdessen ein mit dem im Installationsteil beschriebenen Werkzeug „keytool“ generiertes Zertifikat, was den Client-Browser dazu veranlassen kann, eine zusätzliche Warnmeldung auszugeben. Nachdem man sich einverstanden erklärt hat, dem unbekannten Zertifikat zukünftig zu vertrauen, sollte diese Meldung bei Folgesitzungen ausbleiben.

2.2.1 Das Anlegen eines neuen Benuterkontos

Die erste Eingabe im Registrierungsfomular ist über ein Auswahlfeld vorzunehmen. Hier wählt ein neuer Benutzer die gewünschte Rolle im System aus. Jede Benutzer-Interaktion mit WebTNDB erfordert eine Autorisierung gemäß einem rollenbasierten Zugriffskontroll-Modell wie es der RBAC-Standard (**R**ole **B**ased **A**ccess **C**ontrol) des NIST (**N**ational **I**nstitute of **S**tandards and **T**echnology) beschreibt ([Nist03]). Jedem Benutzer ist dabei eine Rolle zugeteilt. Prinzipiell wären auch mehrere Rollen pro Benutzer denkbar, das Web-Interface zu WebTNDB erlaubt derzeit jedoch nur die Auswahl einer Rolle. Benutzer, die mehrere Rollen im System übernehmen sollen, müssen daher mehrere Benutzerkonten beantragen. Bei der Bereitstellung der Benutzerinteraktionen kann durch die Vergabe von mit bestimmten Rollen assoziierten Autoritätsstufen festgelegt werden, welche Rollen einen Benutzer zu welchen Interaktionen berechtigen. Die Vorgehensweise hierbei wird im Abschnitt über das Deployment des Web-Interface am Ende Implementierungskapitels erklärt. Auf die Verwaltung der verfügbaren Rollen und die Zuordnung der entsprechenden Autoritätsstufen wird noch im Administrationsteil in Anhang A eingegangen.

Rollen mit denen eine höhere Autoritätsstufe assoziiert ist als die der Rollen „guest“ oder „user“, dürfen nur von Personen angefordert werden, die dazu berechtigt sind. Es ist denkbar, das Online-Interface von WebTNDB zu erweitern, so daß ein Administrator automatisch eine Email mit dem Antrag auf ein neues Benutzerkonto mit den gewünschten Privilegien erhält oder die Aufforderung zur Eingabe eines Administrator-Kennworts sicherstellt, daß bei der Einrichtung eines solchen Kontos ein Adminstrator anwesend ist. Derzeit ist aber noch ein direkter Zugriff auf die Nutzertabellen, wie weiter unten beschrieben notwendig, um solche Konten anzulegen. Das Online-Interface lehnt Anträge auf privilegierte Konten, von unbekannten Nutzern grundsätzlich ab. Der Unterschied zwischen den beiden nicht privilegierten Rollen „user“ und „guest“ ist, daß man sich beim Einrichten eines Kontos für einen Benutzer in der Rolle „guest“ damit einverstanden erklärt, daß dieses wieder gelöscht wird, wenn es über einen längeren Zeitraum nicht genutzt wird.

In die weiteren vier Felder sind ein Benutzername, ein Passwort, eine Bestätigung desselben und eine gültige Email-Adresse einzugeben. Festzulegen, welche Auflagen für gültige Eingaben in diesen Feldern bestehen, ist auch Bestandteil der Deployment-Konfiguration. Die Vorgabe eines bestimmten Musters kann z.B. die Auswahl von Paßwörtern, die bestimmten Sicherheitskriterien genügen, erzwingen. Das Testsystem akzeptiert Benutzernamen und Passwörter ab einer Länge von 6 Zeichen ohne Leerzeichen und fordert, daß Paßwort und Paßwort-Bestätigung übereinstimmen, sowie, daß die Email-Adresse dem Muster einer gültigen Email-Adresse inklusive Namen, Domainnamen und dem @-Zeichen entspricht.

2.2.2 Die Aktivierung eines neuen Benutzerkontos

Wenn das System die Eingaben im Registrierungsformular angenommen hat, wird eine Erfolgsmeldung ausgegeben. Außerdem weist WebTNDB den neuen Benutzer darauf hin, daß eine Email mit Instruktionen, wie der Registrierungsvorgang abzuschließen ist, an die angegebene Email-Adresse geschickt wurde.

Abhängig vom Service-Provider des angegebenen Email-Accounts, i.a. jedoch bereits nach wenigen Sekunden, sollte sich diese Email dort abrufen lassen. In der Email ist eine URL angegeben, die mit einer Kennung versehen ist, so daß der Empfänger, wenn er die Adresse mit seinem Browser-Programm abruft, vom Server wieder identifiziert werden kann. Erst nach dem Abruf dieser Adresse wird ein neues Benutzerkonto aktiviert. Der Abruf lässt sich aus den meisten heute gängigen Email-Clients und Web-Schnittstellen zu Email-Accounts durch einmaliges Anklicken der als Hyperlink markierten URL in der Email tätigen, so daß diese nicht in das Adressfeld des Browsers abgetippt werden muß.

Um sich das erste Mal beim System anzumelden, ist die Aktivierung des neuen Benutzerkontos notwendig. Zusammen mit dem geheimen Passwort ist dadurch sichergestellt, daß neue Benutzer bei der Anmeldung tatsächlich ihre eigene Email-Adresse angeben.

Die abgeschlossene Aktivierung wird vom System wieder mit einer Erfolgsmeldung quittiert. Der Rest des Tutorials beschreibt die Bedienung des Anfrage- und Datenverwaltungssystems. Dabei wird davon ausgegangen, daß der Benutzer beim System angemeldet ist und dort über ein aktives Konto mit Administratorberechtigung verfügt. D.h. einige der im folgenden dargestellten Schritte führen zu Fehlermeldungen, wenn ein neu registrierten Benutzer sie am Testsystem ausführt, da dieser nicht zu Veränderungen an den gespeicherten Datensätzen autorisiert ist. Der Abschnitt über die Server-Installation, der dieses Kapitel abschließt, erklärt, wie Administratorkonten für einen neu installierten WebTNDB-Server angelegt werden können.

Sind die Zugangsdaten für ein solches Konto am Testsystem bekannt, kann man sich dort damit anmelden. Danach gelangt man durch einen Klick auf die Start-Schaltfläche links im Browser-Fenster ins Anfragesystem.

2.3 Das Anfragesystem

Die erste Seite des Anfragesystems zeigt links eine farblich abgesetzte vertikale Kontrollleiste mit vier Aktions-Schaltflächen und in der Mitte des Fensters den Baumeditor bestehend aus dem Pfadeditor mit einem Auswahlfeld und drei Schaltflächen, sowie der grafischen Baumdarstellung darunter.

Die Überschrift im Baumeditor weist darauf hin, daß es sich bei dem zuerst dargestellten Baum um den Anfragefunktionsbaum handelt.

Um das Anfragesystem und den Baumeditor zu bedienen, ist es zunächst nötig, den grafischen Anfragemechanismus zu verstehen.

2.3.1 Anfragebäume

WebTNDB wertet Funktionen der Form $f(t, p_1, ..., p_n)$ als Anfragen aus. Der erste Parameter t einer solchen Funktion ist dabei das Anfrageintervall. Die Funktion f wird durch einen Anfragebaum charakterisiert. *entry* ist die Wurzel aller dieser Bäume, da die Funktionen Einträge aus der Datenbank als Ergebnis haben. Der Anfragebaum, der nur aus der Wurzel besteht wird auch als *entry* geschrieben. Die nur durch diesen Baum charakterisierte Funktion $entry(t, p_1, ..., p_n)$ hätte alle Einträge, deren Termine innerhalb des Anfrageintervalls t liegen und die weiterhin den durch die Parameter $p_1, ..., p_n$ spezifizierten Bedingungen genügen, zum Ergebnis.

Will man die Gültigkeit der Parameter nicht für alle Einträge in der Datenbank auswerten, so läßt sich durch die Angabe eines Ausdrucks, der aus verschiedenen, durch Doppelpunkte voneinander getrennten Bezeichnern besteht, der Pfad eines Baumes als Anfragefunktion spezifizieren. $entry : event(t, p_1, ..., p_n)$ bezeichnet beispielsweise die Anfragefunktion, die nur Einträge vom Typ *event* als Ergebnis hat. Verlängert man den Pfad um einen weiteren Bezeichner, wird auch dieser zum Anfrageselektor und die Anfrage damit spezifischer. Beispielsweise wird durch $entry : event : concert$ ein Pfad beschrieben, der verwendet werden kann, um eine Funktion zu charakterisieren, die Einträge liefert deren Typ *event* ist, **und** deren Typ weiterhin *concert* ist. Entlang der Pfade werden die Knoten also durch ein logisches „und" miteinander verknüpft.

„Oder"-Verknüpfungen lassen sich durch die Angabe mehrerer Pfade, die alle die gleiche Wurzel haben, beschreiben. Die Gesamtheit der Pfade zur Charakterisierung einer Anfragefunktion bildet den Anfragefunktionsbaum. Die Pfade $entry : appointment$, $entry : event : exposition$ und $entry : event : concert$ können z.B. zur Beschreibung einer Anfragefunktion, die Termine für Verabredungen, Ausstellungen und Konzerte in der Datenbank selektiert verwendet werden.

Entlang der einzelnen Pfade wird also von der Wurzel beginnend von Knoten zu Knoten eine zunehmende Spezialisierung der Anfragen beschrieben. Je genereller eine Anfrage gestellt ist, umso höher ist dabei die Trefferwahrscheinlichkeit. Eine Anfragefunktion mit dem Pfad $entry : event$ im Funktionsbaum selektiert beispielsweise generell Einträge vom Typ *event*, also alle Einträge speziellerer Typen wie z.B. *concert* oder *exposition*.

2.3.2 Parameterbäume

Die Parameter $p_1, ..., p_n$ sind auch durch Bäume charakterisiert. Das können wieder einzelne Pfade wie z.B. $season : spring$ sein, wenn man nur Einträge im Frühling selektieren möchte, oder aus mehreren Pfaden zusammengesetzte Parameterbäume wie z.B. $location : eu : de : hsn : kss$, $location : eu : de : hsn : frm$, $location : eu : de : bay : muc$ und $location : usa$, wenn man sich für Termine in Kassel, Frankfurt, München oder Nordamerika interessiert.

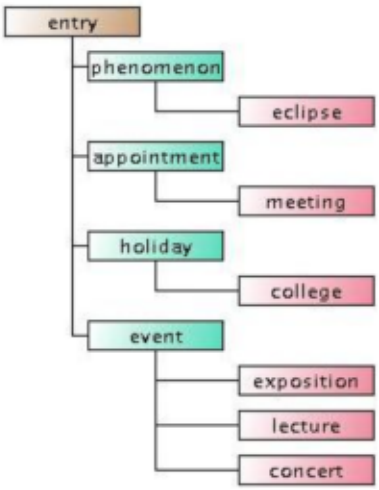
Auch bei der Selektion durch Parameter lassen sich durch kürzere Pfade allgemeinere Anfragen stellen. In den Meta-Datentabellen ist durch die Schemata

der einzelnen Typen genau festgelegt, durch welche Attribute die Einträge jedes Typs spezifiziert sein müssen. Die Präzision der Ortsangabe beim Speichern eines Termins läßt sich dagegen für jeden Eintrag individuell festlegen. Die Anfrageformulierung mit Parameterbäumen, die *location* Pfade enthalten, läßt sich mit Typspezifikationen, wie sie in objektorientierten Programmiersprachen durch Vererbung modelliert werden, vergleichen. Das dort gültige Substitutionsprinzip für polymorphe Referenzen gilt hierbei insofern, als durch einen auf das Land genau spezifizierten *location* Pfad Ereignisse, deren Schauplatz auf das Land, den Staat oder die Stadt genau bekannt ist, angewählt werden können.

Parameterpfade müssen nur für jeweils einen der Parameter $p_1, ..., p_n$ alle die gleiche Wurzel haben. Alle Parameter $p_1, ..., p_n$einer Anfragefunktion f werden also genau genommen durch einen Parameterwald spezifiziert. Parameterbäume mit der Wurzel *season* oder *location* sind dabei für alle Anfragen relevant, während die anderen Parameter wie z.B. *classic* oder *scientific* sinngemäß immer einem Typ wie z.B. *concert* oder *lecture* zugeordnet sind.

2.3.3 Der Anfrageeditor

Oberhalb der grafischen Baumdarstellung befindet sich der Pfadeditor, der für das Hinzufügen und Entfernen von Pfaden im Funktionsbaum verwendet wird. Anfangs wird dort der Wurzelkonten *entry* gefolgt von einem Doppelpunkt und einem Auswahlfeld für den ersten Folgeknoten dargestellt. Nach der Auswahl eines Knotens wie z.B. *event* kann der Pfad *entry* : *event* mittels der „delete"-Schaltfläche aus dem Funktionsbaum entfernt werden, vorrausgesetzt dieser Pfad ist im dargestellten Anfragebaum enthalten. Dieselbe Aktion entfernt auch speziellere Pfade wie *entry* : *event* : *exposition* oder *entry* : *event* : *concert* aus dem Funktionsbaum. Wird ein Pfad ausgewählt, der nicht im Funktionsbaum enthalten ist, so kann dieser durch einen Klick auf die „add"-Schaltfläche neu eingefügt werden.



Um einen spezielleren Pfad zu bearbeiten, muß der entsprechende Kindknoten von *entry* in der grafischen Baumdarstellung angeklickt werden. Der Pfadeditor wandert daraufhin einen Knoten weiter und zeigt einen Pfad wie beispielsweise *entry* : *appointment* oder *entry* : *event*, gefolgt vom zugehörigen Auswahlfeld für den nächsten Kindknoten an.

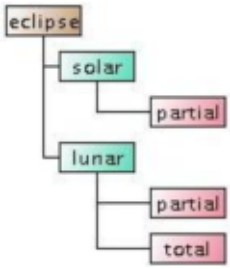
Durch den Klick auf einen Blattknoten der Tiefe Drei, wie z.B. *exposition* oder *lecture*, gelangt man zum Parametereditor für den zugehörigen Parameterbaum. Das Anfragesystem zeigt daraufhin eine neue Überschrift im Baumeditor an, um darauf hinzuweisen, daß ein Parameterbaum dargestellt wird, und ersetzt in der Kontrollleiste eine Schaltfläche durch eine neue, über die man zum Funktionsbaum zurückgelangt. Sofern für einen so ausgewählten Knoten optionale Attribute in den Meta-Datentabellen spezifiziert sind, lassen sich diese in der Parameterbaumdarstellung genau so mit dem Pfadeditor bearbeiten wie die Pfade des Funktionsbaums.

Die im Parametereditor ersetzte Schaltfläche der Kontrollleiste dient im Funktionseditor zum Umschalten auf den Editor für den Parameterwald. Von dort gelangt man über die gleiche Schaltfläche wieder zurück zum Funktionseditor.

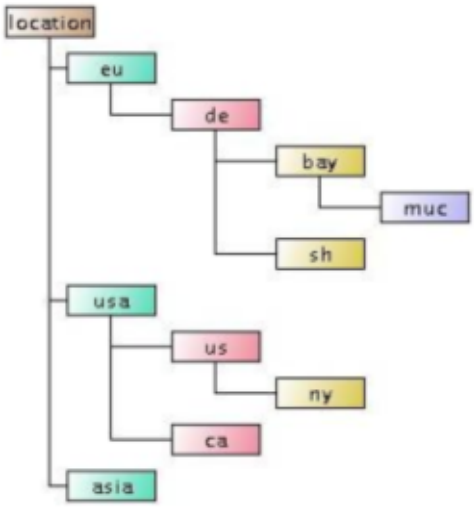
Der Editor für den Parameterwald vereint diesen durch Hinzufügen eines gemeinsamen Wurzelknotens *parameter* zu einem Gesamt-Parameterbaum. Klickt man in dieser Darstellung einen Parameterknoten an, gelangt man wieder in den Parametereditor für den entsprechenden Parameterbaum, kann von dort über die Kontrollleise aber direkt zur Gesamtübersicht zurückspringen.

Die Gesamtübersicht zeigt alle Paramterbäume, für die in den Meta-Datentabellen mögliche Ausprägungen gespeichert sind, als Unterbäume der Wurzel *parameter* an. Die beiden untersten sind dabei die für alle Anfragen relevanten Parameterbäume *season* und *location*. Die darüber aufgelisteten Bäume werden bei der Evaluierung nur berücksichtigt, falls die dadurch parametrisierten Knoten im Funktionsbaum enthalten sind. Der Pfad $lecture{:}scientific$ im Parameterbaum selektiert z.B. nur dann Vorlesungen dieses Typs, wenn der Pfad *entry* : *event* : *lecture* auch im Funktionsbaum enthalten ist. Werden dort nur andere Ereignisse oder überhaupt keine oder durch den Pfad *entry* : *event* alle Ereignisse ausgewählt, wird der *lecture* Parameterbaum ignoriert und folglich werden auch Vorlesungen anderer Typen selektiert.

Die maximal mögliche Tiefe der einzelnen Parameterbäume ist durch die Zahl der in den Meta-Datentabellen spezifizierten Attribute der entsprechenden Typen begrenzt. Das Testsystem erlaubt nur für Einträge des Typs *entry* : *phenomenon* : *eclipse* mehr als ein Attribut, für die mit Parameterpfaden wie z.B. *eclipse* : *solar* : *partial* oder *eclipse* : *lunar* : *total* nach totalen Mond- und partiellen Sonnenfinsternissen gesucht werden kann.



Eine Ausnahme ist der globale Parameter *location* für den, je nach gewünschter Präzision bei der Ortswahl, Bäume mit einer Tiefe von bis zu fünf Knoten zusammengestellt werden können, wenn einzelne Pfade die Suche bis auf die Stadt genau einschränken.



Die Breite der Parameterbäume, also die Anzahl der Knoten gleicher Tiefe, ist prinzipiell beliebig. Durch einen Klick auf die „options“-Schaltfläche oberhalb des Auswahlfelds im Pfadeditor gelangt man zu einer Tabelle der möglichen Attribut-Belegungen und kann dort nach Bedarf selbst optionale Belegungen hinzufügen oder entfernen. Sofern es sich nicht um eine statische Tabelle, wie die der unter *season* aufgelisteten vier Jahreszeiten, handelt, paßt das System danach die Auswahlfelder entsprechend an.

Somit ist es also möglich, über das Web-Interface auch bestimmte Meta-Datentabellen zu bearbeiten. Versucht ein Benutzer dabei Optionen zu löschen, die als Belegungen in der Datenbank vorkommen, was ungültige Informationen in den Datenbeständen zur Folge hätte, wird die entsprechende Anforderung nicht ausgeführt.

Die Schemata und Inhalte aller Meta-Datentabellen des Testsystems sind im Abschnitt über die Installation aufgelistet, der auch beschreibt, wie ein Administrator hier beliebige Manipulationen vornehmen kann.

2.3.4 Anfrageintervalle

Der erste Parameter t einer von WebTNDB auszuwertenden Funktion $f(t, p_1, ..., p_n)$, das Anfrageintervall, wird auch in einem grafischen Editor spezifiziert. In der Kontrollleiste befindet sich die Schaltfläche, über die man zum Intervalleditor gelangt. Um eine Kalenderdarstellung auszuwählen, wird der Benutzer nach Betätigen dieser Schaltfläche zunächst zur Auswahl einer Region aufgefordert. In einem Auswahlfeld läßt sich dazu jede der für das Testsystem verfügbaren Regionalrepräsentationen auswählen. Intern läßt WebTNDB diese Optionen durch die Klassenmethode `getAvailableLocales( )`der Klasse `Locale` aus dem Paket `java.util` ermitteln. Daher ist die Anzahl der verfügaren Repräsentationen von der Java-Installation auf dem Server-Rechner abhängig. Bei der voreingestellen Repräsentation handelt es sich, falls innerhalb der laufenden Sitzung hier bereits Einstellungen vorgenommen wurden, um die zuletzt ausgewählte Repräsentation. Beim ersten Aufruf des Intervalleditors wird die Voreinstellung durch einen Aufruf der `getDefault( )` Methode bestimmt, die ebenfalls statisch von der Klasse `java.util.Locale` bereitgestellt wird. Wenn keine neuen Änderungen an dieser Einstellung nötig sind, kann man mit der entsprechend beschrifteten Schaltfläche in der Kontrollleiste mit der nächsten Einstellung für den Intervalleditor fortfahren. Für das Testsystem ist die Defaulteinstellung eine deutsche Kalenderrepräsentation. Eine Änderung an der ausgewählten Repräsentation betrifft die für die einzelnen Monate und Wochentage verwendeten Symbole, sowie die Darstellung der Wochen in der Monatsübersicht. Während diese in Deutschland mit dem Montag beginnen, wird z.B. nach Auswahl des isländischen Kalenders der Sonntag als erster Tag der Woche dargestellt. Bei der Auswahl einer Regionalrepräsentation mit arabischen oder chinesischen Schriftzeichen ist weiterhin zu beachten, daß bei einigen Browserprogrammen eventuell ein Plugin für die Darstellung dieser Unicode-Symbole zusätzlich installiert werden muß.

Die zweite Einstellung für den Intervalleditor ist die Auswahl einer Zeitzone. Voreingestellt ist auch hier wieder die Default-Zeitzone auf dem Testsystem, bzw. die zuletzt ausgewählte Zeitzone. Default-Zeitzone und verfügbare Zeitzonen werden für diese Einstellung durch die Klassenmethoden `getAvailableIDs( )` und `getDefault( )` der Klasse `java.util.TimeZone` bestimmt und sind damit ebenfalls von der Java-Installation auf dem Server-Rechner abhängig. Die Auswahl einer Zeitzone bestimmt zugleich ein Schema für die Sommer/Winter-Zeitumstellung.

Um das Anfrageinterval selbst auszuwählen, ist als erstes das Jahr und der Monat, ab dem Einträge selektiert werden sollen, einzugeben. Der Monat wird über ein Auswahlfeld gewählt und das Jahr in ein Textfeld eingegeben. WebTNDB besteht dabei auf Eingaben im SQL-Escape-Syntax Format. Das Syntaxmuster für eine Jahreszahl ist dabei „`yyyy`“, eine numerische Eingabe mit genau vier Ziffern für Jahreszahlen zwischen `0` und `8099`. Jahreszahlen von Daten vor dem Jahr `1000` sind mit führenden Nullen zu schreiben. Diese Beschränkungen stammen von den Datentypen `java.sql.Date` und `java.sql.Time`, die WebTNDB intern verwendet, da Objekte dieser Typen von jedem Datenbankt-

reiber auf `Time` oder `Date`-Spalten in der Datenbank abgebildet werden können. Die Darstellung von Literalen für Zeitangeben in SQL-Statements ist nicht standardisiert und variiert daher bei verschiedenen Implementierungen. Um die Portabilität zu unterschiedlichen relationalen Datenbanken zu gewährleisten, müssen die Stringliterale, die zur Angabe der Jahreszahl und auf der Folgeseite zur Angabe der Uhrzeit verwendet werden, in SQL-Escape-Syntax eingegeben werden. `java.sql.Date` und `java.sql.Time` stellen Klassenmethoden bereit, die WebTNDB verwendet, um die eingegebenen Strings zu parsen und daraus Instanzen der jeweiligen Klasse zu erzeugen ([WFCHH99, S. 820, S. 936]).

Auf der Folgeseite wird der Startzeitpunkt für das Anfrageintervall ausgewählt. Zuerst ist dazu die Uhrzeit in ein Textfeld einzugeben, für das Eingaben der Form „`hh:mm:ss`“ akzeptiert werden. Für die Stunde sind dabei Werte von `0` bis `23` und für Minute und Sekunde vom `0` bis `59` zulässig. In der Kalendergrafik wird für jeden Tag des ausgewählten Monats eine Schaltfläche zur Angabe des Datums angezeigt. Mit einem Klick auf das gewünschte Datum wird auch die eingegebene Uhrzeit bestätigt.

Für die Auswahl des Endzeitpunkts sind auf die gleiche Weise auf den beiden Folgeseiten nochmals Jahr, Monat, Uhrzeit und Tag anzugeben. Danach wird der Intervaleditor automatisch beendet und der Benutzer gelangt wieder zum Anfrage-Editor. Die Auswahl eines Endzeitpunkts vor dem Startzeitpunkt ist nicht ausgeschlossen und führt immer zu einer leeren Ergebnismenge. Das im Intervalleditor eingegebene Intervall wird genauso wie der Funktionsbaum und die Parameterbäume für jeden Benutzer individuell in der Datenbank gespeichert, so daß auch die dort eingegebenen Daten in zukünftigen Sitzungen wieder verfügbar sind. Ein neuer Benutzer findet vor der ersten Auswahl eines Anfrageintervalls den Zeitraum, der einen Montat vor dem Tag seiner Anmeldung beginnt und einen Monat nach seiner Anmeldung endet als Anfrageintervall voreingestellt.

2.4 Datenverarbeitung mit WebTNDB

Mit dem Datenbankmanagementsystem WebTNDB kann man Anfragen natürlich nicht nur formulieren und editieren, sondern auch auf den Datenbeständen relationaler Datenbanken auswerten. Weiterhin können direkt über das Web-Interface Datensätze bearbeitet, neu eingefügt oder gelöscht werden.

Die oberste Schaltfläche in der Kontrollleiste, dient im Anfragesystem dazu, die durch den aktuell gespeicherten Funktionsbaum, die Parameterbäume und das Anfrageintervall spezifizierte Anfrage für die bei der Installation konfigurierte Datenbank auszuwerten.

Nach Betätigung dieser Schaltfläche wird anstelle des Baumeditors eine tabellarische Übersicht über die den Bedingungen der Anfragefunktion genügenden Datensätze in der Datenbank angezeigt. Über die zwei oberen Schaltflächen in der Kontrollleiste gelangt man zurück zum Editor für den Funktionsbaum oder den Parameterwald. Darunter befindet sich weiterhin die Schaltfläche mit der sich der Intervalleditor aufrufen läßt, um die gleiche Anfrage für ein neues Intervall auszuwerten, und eine weitere, um die Datenbanksitzung zu beenden.

Hat man die gewünschten Informationen gefunden, sollte man sich nach jeder Sitzung durch Betätigen dieser Schaltfläche explizit abmelden. WebTNDB muß sonst sitzungsbezogene Daten nach der letzten Aktion noch bis zum voreingestellten Session-TimeOut im flüchtigen Serverspeicher halten, was unnötig Ressourcen verschwendet. Bei der nächsten Anmeldung gibt WebTNDB eine Warnmeldung aus, wenn die letzte Sitzung des Benutzers nach einem TimeOut vom Server beendet werden mußte.

2.4.1 Die Datentabelle

Die Zeilen der Datentabelle stellen alle bei der Auswertung der Anfragefunktion gefundenen Datensätze dar (möglicherweise also auch keine). In den einzelnen Spalten werden jeweils der Titel des Eintrags, eine Beschreibung sowie Ort und Termin aufgelistet. In etwas kleinerer Schrift werden in der Titelspalte auch der Typ und weitere Spezifizierer angegeben, durch die der Eintrag mit der gespeicherten Anfragefunktion selektiert wurde. In der Ortsspalte wird zusätzlich die Präzision der gespeicherten Ortsangabe für jeden Eintrag angezeigt. Falls für einen Eintrag eine URL gespeichert wurde, wird in der Spalte mit der Beschreibung zusätzlich ein Link angezeigt, über den sich ein weiteres Browserfenster mit zusätzlichen Informationen zu diesem Eintrag öffnen läßt.

In der letzten Spalte jeder Zeile wird für jeden Eintrag jeweils eine Schaltfläche zum Löschen und eine zum Bearbeiten des Datensatzes in der Datenbank angezeigt. In der letzte Zeile der Tabelle befindet sich die „add"-Schaltfläche, die verwendet wird, um einen neuen Datensatz in die Datenbank einzufügen.

2.4.2 Das Einfügen neuer Datensätze

Für einen neuen Eintrag muß als erstes die Präzision für die Ortangabe eingestellt werden. Abhängig davon, ob der Schauplatz auf die Stadt, das Bundesland bzw. die Provinz, das Land oder nur auf den Kontinent ganau bekannt ist, stellt das Auswahlfeld zur Ortswahl auf der Folgeseite Städte, Staaten, Länder oder Kontinente bereit. Nach dem Ort ist der Typ des Eintrags auszuwählen. Die Optionen hierfür liest WebTNDB aus den Meta-Datentabellen. Das Testsystem stellt für Ereignisse den Typ *event*, für Naturereignisse den Typ *phenomenon*, für Feiertage den Typ *holiday* und für Verabredungen den Typ *appointment* bereit. Auf der Folgeseite ist für jeden Typ ein zusätzlicher Selektor auszuwählen, der den Eintrag weiter kategorisiert. Für den Typ *event* stehen dort beispielsweise *concert*, *lecture* und *exposition* zur Auswahl. Tatsächlich bestimmt die Auswahl dieser Selektoren, welche Einträge durch die namensgleichen Knoten der Pfade in den Anfragefunktionsbäumen selektiert werden. Wieviele weitere Einstellungen für zusätzliche Spezifizierer vorgenommen werden müssen, ist durch das dem ausgewählten Typ zugeordnete Schema in den Meta-Datentabellen festgelegt. Die Spezifizierer entsprechen den bei einer Anfrage durch die Parameterbäume beschriebenen Attributselektoren.

Auf der letzten Seite des Einfügevorgangs wird die „Proceed"-Schaltfläche in der Kontrollleiste durch die „Submit"-Schaltfläche ersetzt, durch deren Anklicken

der neue Eintrag in die Datenbank übernommen wird. Abhängig von Typ und Ausprägung sind auf dieser Seite unterschiedlich viele Informationen zur Beschreibung des neuen Eintrags, wie z.B. ein Name oder die Adresse einer Web-Seite mit Zusatzmaterial, in Textfelder einzutragen. Die dort eingegebenen Stringliterale sollten den durch die zuvor gewählten Selektoren und Spezifizierer typisierten Eintrag sinnvoll beschreiben. Einzelne Textfelder sind optional, bei anderen ist immer eine Angabe zu machen. Für Einträge deren Termine nicht rechnerisch bestimmt werden müssen, ist immer der Start- und der Endzeitpunkt anzugeben. Eingaben in den dafür vorgesehenen Textfeldern müssen aus den schon bei bei der Beschreibung des Intervalleditors angesprochenen Gründen wieder in SQL-Escape-Syntax getippt werden. Für die Uhrzeit ist das wieder das Format „`hh:mm:ss`“ und für das Datum „`yyyy-mm-dd`“. Für die Jahreszahl im Format „`yyyy`“ gelten dabei wieder die schon vom Intervalleditor bekannten Beschränkungen, für „`mm`“ werden Eingaben von `01` bis `12` zur Angabe des Monats und für „`dd`“ Eingaben von `01` bis `31` zur Angabe des Tages akzeptiert. Die Zeitangaben müssen im gregorianischen Kalender gültige Daten sein und der Endzeitpunkt darf hier nicht vor dem Startzeitpunkt liegen.

Akzeptiert das System alle Eingaben, gelangt man wieder zurück zur Datentabelle. Der neue Eintrag ist dort jedoch nur dann sofort sichtbar, wenn er durch die mit der aktuellen Anfragefunktion spezifizierten Selektion ausgewählt wird. Sonst müssen zuvor der Funktionsbaum, die Parameterbäume oder das Anfrageintervall angepasst und die Anfrageergebnisse in der Datentabelle neu ermittelt werden. Gerade neue Benutzer sollten sich daher angewöhnen, neue Einträge in der Selektion zu spezifizieren, bevor sie diese in der Datenbank speichern, um keine Daten zu „verlegen“.

2.4.3 Das Bearbeiten von Datensätzen

Oberhalb der „delete“-Schaltfläche, mit der sich Datensätze vollständig aus der Datenbank entfernen lassen, befindet sich die „edit“-Schaltfläche. Diese kann benutzt werden, um Teilinfomationen an den gespeicherten Datensätzen zu verändern oder zu einer Detailansicht zu gelangen. Nach einem Klick auf diese Schaltfläche wird der Benutzer durch die gleiche Seitenfolge geführt, die beim Einfügen des Datensatzes in die Datenbank durchlaufen wurde. Nur sind bei diesem Durchlauf in allen Text- und Auswahlfeldern, die mit dem Datensatz dafür gespeicherten Informationen bereits eingetragen. Ändert man auf der ersten Seite die Präzision der Ortsangabe, kann man sich so z.B. die Information holen in welchem Bundesland, Land oder Kontinent, die für einen Eintrag gespeicherte Stadt liegt. Anschließend muß der „edit“ Vorgang aber mit einem Klick auf die Schaltfläche „cancel“ beendet werden, da die Ortsangabe mit geringerer Präzision sonst mit dem Eintrag gespeichert wird.

Beabsichtigte Änderungen an den gespeicherten Datensätzen können genau wie bei dem zuvor Beschriebenen „add“-Vorgang vorgenommen werden und mit einem Klick auf die „submit“-Schaltfläche auf der letzten Seite des „edit“-Durchlaufs in die Datenbank übernommen werden. Auch hierbei ist zu beachten, daß Änderungen dazu führen können, daß Einträge durch die gespeicherte

Selektion nicht mehr erfasst werden. Ist dies nicht erwünscht, so müssen das Anfrageintervall oder die entsprechenden Bäume angepasst werden, bevor Änderungen gespeichert werden.

3 Implementierung

Dieses Kapitel setzt vorraus, daß der Leser mit Java und XML vertraut ist. Eine Einführung in die Programmierung mit Java kann z.B. [CHW00] liefern. Die Grundlagen von Technologien wie JDBC, JSPs, Servlets u.a. sind z.B. in [Paw00] nachzulesen.

3.1 Die Architektur von WebTNDB

Ein wichtiges Konzept von J2EE sind Komponenten und Container. J2EE definiert eine Vielzahl von Basisklassen und Interfaces. Durch Erweiterung der Basisklassen bzw. Implementierung der Interfaces ist es möglich, neue Software zu bauen die sich in andere Programme integrieren läßt, da ihre Schnittstellen dort bekannt sind.

Die Methoden solcher Schnittstellen nennt man Callback-Methoden oder, wenn beim Aufruf eine bestimmte Reihenfolge feststeht, manchmal auch Lifecycle-Methoden. Programme, in die Komponenten mit solchen Schnittstellen eingebettet werden, sind sogenannte Container oder Frameworks, die wiederum selbst in Container eingebettet werden.

Für WebTNDB ist Tomcat ein solcher Container. Tomcat stellt als solcher die Basisfunktionalität eines Web-Servers bereit : Er startet Threads, die Anfragen verschiedener Clients parallel bearbeiten, und übersetzt Daten aus Java-Objekten in HTTP-Nachrichten und umgekehrt.

Jakarta Struts ist ein sogenanntes Font-End Framework, durch dessen Konfiguration bestimmt ist, welche Java-Komponente für welche Interaktion mit dem Client zuständig ist. Die für Struts konfigurierten Komponenten, sogenannte Actions, greifen auf bestimmte Dienste des Systems zu und steuern den Fluß von Seite zu Seite. Weiterhin sind für Struts sogennante Tiles konfiguriert, aus denen die einzelnen Seiten zusammengesetzt werden. Die Struts-Actions und Tiles bilden somit die Präsentationsschicht von WebTNDB.

WebTNDB ist eine mehrschichtige, sogenannte 3-tier application. Monolithische Architekturen, Business-Logik in JSPs oder quer über die Anwendung verstreute direkte Zugriffe auf Datenbankverbindungen sind die häufigsten Architektur-Fehler, die bereits kleinere Projekte nach kurzer Zeit zum scheitern bringen können, wie inzwischen allgemein bekannt ist und in der Literatur über Refactoring und Anit-Patterns nachgelesen werden kann ([Tat02]).

Für WebTNDB wurde von Anfang an ein dreischichtiges Modell gewählt. Ein grundlegendes Prinzip von Schichtenmodellen ist, daß immer nur obere Schichten darunterliegende kennen dürfen und niemals umgekehrt. Tiefere Schichten werden somit wiederverwendbar, höhere austauschbar und die Verteilung von Aufgaben ist klar geregelt.

Die drei Schichten bei dem von WebTNDB implementierten 3-tier Model sind die schon angesprochene Presentation-Tier, darunter die Anwendungsschicht (Business-Tier) auf der Dienste, beispielsweise für die Umsetzung von Bäumen in Datenbankanfragen, bereitgestellt werden, und zuunterst die Datenzugriffsschicht (Integration-Tier), die für die Interaktion mit anderen Systemen wie

WebCAL und relationalen Datenbanken zuständig ist.

Beim 3-tier Modell handelt es sich um ein sogenanntes architectural pattern, also ein Muster für die Struktur eines gesamten Systems, das in diesem Fall die Makro-Architektur von WebTNDB beschreibt. Ein weiteres architectural pattern auf einem etwas niedrigerem Level ist das Model-View-Controller Modell, auf das im Abschnitt über die Präsentationsschicht noch eingegangen wird.

Design-Patterns oder Entwurfsmuster beschreiben dagegen die Stuktur und Interaktionen einzelner Komponenten und stehen immer im Kontext einer konkreten Problemstellung. In den letzten Jahren hat die Anerkennung und damit die Bedeutung solcher Entwurfsmuster stark zugenommen. Design-Patterns erlauben die Wiederverwendung von Ideen als Lösungansätze konkreter Problemstellungen und geben diesen erprobten Lösungen eindeutige Namen. Design-Patterns können dazu verwendet werden, Pakete von Klassen und deren Interaktionen durch einen einzigen Namen zureichend präzise zu dokumentieren, um deren Schnittstellen und Funktionalität verständlich zu machen.

Davon soll auch in dieser Arbeit profitiert werden. Die sogennanten GoF-Patterns wurden nach den vier Autoren des Design-Pattern Katalogs [GHJ97], in dem sie erstmals dokumentiert wurden, benannt, die man auch als die **G**ang **o**f **F**our bezeichnet. Dabei handelt es sich um Entwurfsmuster wie Factory, Singleton oder Composite, die inzwischen so bekannt sind, daß sie hier nicht näher erläutert werden müssen. Wenn der Name eines solchen Patterns im folgenden bei der Quelltext Dokumentation verwendet wird, kann auf den Design-Pattern Katalog [GHJ97] oder eine von mittlerweile Hunderten von Internet-Seiten mit Diagrammen und Erklärungen zur Referenz zurückgegriffen werden.

Einen weiteren Design-Pattern Katalog, der speziell für Web-Applikationen Lösungsansätze dokumentiert, stellen die J2EE Patterns des Sun Java Center dar. Da viele Komponenten von WebTNDB Implementierungen einiger dieser Entwurfsmuster darstellen, ist es für das Verständnis der folgenden Dokumentation hilfreich, auch diese Dokumentation zu nutzen.

J2EE Patterns gehen jedoch oftmals von der Verwendung eines weiteren Frameworks, den Enterprise Java Beans, aus, auf das bei WebTNDB verzichtet wurde. Einige Patterns sind daher abweichend von der offiziellen Dokumentiation implementiert. Einzelne Klassen wurden aber auch speziell so strukturiert, das die Portierung zu einem EJB-Container möglichst einfach wäre. Überall wo EJBs bei Design-Entscheidungen eine Rolle gespielt haben, wird das angesprochen. Kenntnisse über diese interessante Technologie sind für das Verständnis von WebTNDB jedoch nicht notwendig, können aber z.B. beim Lesen der empfehlenswerten Einführung von Richard Monson-Haefel erworben werden [MH01].

Da hier aus Platzgründen nur Auszüge der Quelltexte wiedergegeben werden können, wird für das Verständnis der Zusammenhänge vorrausgesetzt, daß der Leser dieser Dokumentation auch Zugriff auf alle Quellen von WebTNDB hat. Die folgende Dokumentation geht nur bei besonders fundamentalen Verfahren im Detail auf deren Implementierung ein. Zahlreiche Komponenten werden hier nur mit deren Namen und Funktion aufgelistet, um die nötigen Informationen für Weiterentwicklungen auf Basis der TNDB-API wiederzugeben und das Auffinden von Komponenten in der Pakethierarchie zu erleichten. Sollen Verän-

derungen an der TNDB-API selbst vorgenommen werden, ist es unumgänglich auch die Quelltexte zu lesen.

Jeder Abschnitt schließt mit einer Zusammenfassung, um die Funktion der wichtigsten Komponenten einer Schicht und ihre Bedeutung in der nächsthöheren Schicht von den zuvor besprochenen Implementierungsdetails zu abstrahieren.

Höhere Schichten bauen auf den darunterliegenden auf. Um sie zu verstehen, ist es folglich nötig, die darunterliegenden zu kennen. Daher beginnt diese Dokumentation mit der untersten Schicht von WebTNDB.

3.2 Die Datenzugriffsschicht

Die Datenzugriffsschicht oder Integration-Tier umfasst das Paket `rdbacces`[5] das mit seinen Klassen für den Zugriff auf relationale Datenbanken ein sogenanntes Persistence-Framework darstellt. Außerdem die Klassen `WebTNDBServer`, `WebTNDBService` und `Interpreter` des Pakets `service`, in denen der Proxy-Server implementiert ist.

3.2.1 Der Proxy-Server

Da der Proxy-Server praktisch ein eigenständiges Programm darstellt, läßt er sich nicht eindeutig, in das durch die 3-tier Architektur vorgegebene Schichtenmodell einordnen. Hier wird z.B. auch auf verschiedene Klassen der Business-Tier zugegriffen. Um die Implementierung des Proxies im Detail zu besprechen, müsste vorweg auch auf die Netzwerkprogrammierung mit Sockets eingegangen werden. Da das aber kein tyisches Thema für Online-Datenbanken darstellt, wird hier nur auf die Literatur [Paw00, S. 107] verwiesen und im folgenden gleich die Funktion der einzelnen Klassen dargestellt :

Die Klasse `WebTNDBServer` enhält die Klassenmethode `main( )`, also das Hauptprogramm und kann von der Kommandzeile gestartet werden. Hier wird ein neuer `WebTNDBServer` instanziiert und dessen `initService( )` und `provideService( )` Methoden aufgerufen. `initService( )` baut die Netzwerkverbindung auf und `provideService( )` stellt den eigentlichen Dienst bereit. In `provideService( )` wartet der Proxy in einer Endlosschleife auf neue Clients und instanziiert, sobald sich ein solcher dort anmeldet, einen `WebTNDBService` sowie einen neuen Thread, der diesen ausführt, und startet ihn. Die Instanzmethode `finalize( )` wird vom Garbage-Collector aufgerufen, wenn der Server beendet wird, und trennt die Verbindung wieder, funktioniert also wie ein Destruktor.

Ein `WebTNDBService` implementiert den Proxy-Dienst in seiner `run( )`-Methode. Da weiterhin dort client-spezifische Verbindungsdaten gespei-

[5] Entsprechend der üblichen Konvention für weltweit eindeutige Paketnamen, Pakethierarchien so zu schachteln, daß die äußersten Pakete den umgekehrten Domain-Namen des Rechners ergeben, sind alle Pakete des Testsystems Subpackages des Pakets nu.mine.webtndb. Da ein Umzug zu einer anderen Domain aber jederzeit möglich ist, sind in der Quelltext-Dokumentation jeweils nur die Namen der Subpackages angegeben.

chert werden, muß der Server für jeden Client eine eigene Instanz dieser Klasse anlegen. Die Methode `run( )` startet in einer Endlosschleife abwechselnd einen Thread, der eine Eingabe vom Client liest und diese zum WebCAL-Server schickt, sowie einen, der auf dessen Antwort wartet und sie an den Client zurückschickt, und beendet diese in der gleichen Reihenfolge. Damit ist die durch den Proxy bereitgestellte Vermittlung zum WebCAL Server prinzipiell erklärt.

Um Datenbank-Kommandos, die mit einem # beginnen, zu verarbeiten, verwendet ein `WebTNDBService` eine Instanz der Klasse `Interpreter`. Die Methode `process( )` des `Interpreters` interpretiert alle diese Kommandos und greift dafür auf verschiedene `private` Helper-Methoden, wie die im Abschnitt über die Installation gezeigte `tndb( )`-Methode zur Datenbankwahl (Anhang A), zurück. Die Namen dieser Helper beschreiben auch deren Funktion. Die Implementierung greift auf weitere Komponenten der Integration-Tier und der Business-Tier zu, deren Beschreibung noch folgt. Die Klasse `ServerToolkit` stellt Klassenmethoden bereit, die es dem Proxy ermöglichen, auf solche Komponenten zuzugreifen, die sonst nur in anderen Schichten verfügbar sind, und wird ausschließlich vom Proxy-Server verwendet.

3.2.2 Das Persistence Framework

WebTNDB arbeitet auf der Business-Tier mit Objekten wie Anfragefunktionen, Parameterbäumen, Intervallen und Benutzerkonten. Für die persistente Speicherung dieser Objekte stellen relationale Datenbanken nur Tabellen zur Verfügung. Ein Persistence Framework hat die Aufgabe, Objektzustände auf Tabellen abzubilden. Weitere Aufgaben eines Persistence Frameworks sind die Aufrechterhaltung der referentiellen Integrität und das Managemant durch Objekt-Interaktionen implizierter Tabellen-Transaktionen.

Um die Attribute jedes denkbaren Objekts auf eine Tabelle abzubilden, gibt es sogenannte universelle OR-Mapper (http://db.apache.org/ojb). Leider haben diese z.T. noch Performanzprobleme. Weiterhin werden die Schemata der intern verwendeten Tabellen bei solchen Tools i.a. außerhalb des Codes durch Deskriptor-Dateien festgelegt und die die Art und Weise ihrer Verarbeitung ist durch die Klassen dieser Tools bestimmt. Die Datenzugriffsschicht von WebTNDB sollte dem Entwickler eine höhere Flexibilität ermöglichen, da hier in Zukunft auch Veränderungen, wie z.B. die Einführung zusätzlicher Indextabellen für effizientere Zugriffe auf temporale Daten denkbar sind ([KPS00, KPS01]).

WebTNDB verwendet daher ein eigenes Persistence Framework. Einige hierbei umgesetzte Ideen für besonders effiziente Implementierungen, sind dem Kapitel „Patterns applied to a Persistence Framework“[6] aus dem sehr praktischen Buch „J2EE Design Patterns Applied“ entnommen [BCJKNR02, S. 89]. Das Framework verbirgt die physikalischen Struktur der Tabellen vor den höheren Schichten und ermöglicht somit Veränderungen an den Tabellenschemata vorzunehmen, ohne daß die übrigen Schichten der Anwendung davon betroffen sind.

[6]Dieses Kapitel kann kostenlos von www.wrox.com heruntergeladen werden.

Data Access Objects Data Access Objects oder DAOs abstrahieren die sogenannte CRUD-Logik von einer Datenquelle. Dabei handelt es sich um die typischen **C**reate, **R**eplace, **U**pdate und **D**elete Kommandos, mit denen üblicherweise auf persistent gespeicherte Daten zugegriffen wird.

In der abtrakten Klasse *DAO* des Pakets `rdbaccess.daos` stellt WebTNDB mit den abstrakten Methoden *findByPrimaryKey( )*, *findByAttributeValue( )*, *findAll( )*, *insert( )*, *update( )*, *delete( )* und *createValueObject( )* eine Schnittstelle für solche Datenbank-Zugriffe bereit.

Darüberhinaus sind in der Klasse *DAO* verschiedene Helper-Methoden, die für jede DAO-Implementierung u.a. für das Management von Transaktionen benötigt werden, bereits konkret implementiert. Eine Realisierung nur durch Schnittstellen, wie man sie bei EJB-Implementierungen oft sieht ([BCJKNR02, S. 103]), war hier nicht möglich.

Transaktionen beschreiben Einheiten von Arbeitsvorgängen aus Sicht der Datenbanken und sollten daher außerhalb der Intergration-Tier im Code nicht sichtbar sein, um die übrigen Schichten möglichst lose an die Datenbank zu koppeln. Ein großer Vorteil von EJBs ist, daß sich das transaktionale Verhalten durch sogenannte Transaktionsattribute beim Deployment auf Methodenebene vereinbaren läßt ([MH01, S. 404]). Die Aufgabe, transaktionalen Code zu schreiben, der sehr schwer zu debuggen ist, insbesondere wenn Transaktionen mehrere Komponenten umfassen, kann dem Entwickler dann von der EJB-Container-Implementierung abgenommen werden. Ein EJB-Container, der Transaktionen sogar für Komponenten steuert, die über mehrere Rechner verteilt sind, wird manchmal auch als Component Transaction Monitor (CTM) bezeichnet ([MH01, S. 13]). Die im folgenden besprochenen Fassadenklassen implementieren die grundlegenden Konzepte eines solchen CTMs für die von WebTNDB verwendeten DAOs. Da das für verteilte Transaktionen benötigte Mehr-Phasen-Commit-Protokoll von den nicht kommerziellen JDBC-Treiber Implementierungen i.a. jedoch nicht untersützt wird, wird hier aber davon ausgegangen, daß alle in eine Transaktion involvierten Tabellen in der gleichen Datenbank gespeichert sind.

Für jede der in Abschnitt 2.5.2 und 2.5.3 besprochenen Tabellen ist im Paket `rdbaccess.daos` eine Erbenklasse von *DAO* mit einer konkreten Implementierung der CRUD-Methoden für die jeweilige Tabelle bereitgestellt. `UserDAO` dient z.B. für den Zugriff auf Benutzerkonten.

Als Parameter bzw. Rückgabetyp der CRUD-Methoden sind `rdbaccess.vos.`*ValueObject*s vereinbart. Diese Objekte dienen zum Transport von Daten zwischen dem Persistence Framework und den darüberliegenden Schichten. Die Implementierung der *ValueObject*s wird später noch detailiert besprochen.

In der Implementierung der Methode `findByPrimaryKey( )` des `UserDAO`s kann man sehen, wie das Persistence Framework mittels JDBC auf die Tabellen der Datenbank zugreift, um die Attribute eines solches *ValueObject*s mit den aus einer Zeile der `users` Tabelle gelesenen Daten zu initialisieren :

Die mit der Schnittstelle *Statement* bereitgestellte Methode *executeQuery( )* setzt ein SQL-SELECT-Statement auf die Datenbank ab und

liefert als Ergebnis die Referenz auf ein Objekt mit der Schnittstelle *ResultSet*. Diese Schnittstelle stellt die Sicht des Java-Programms auf die Tabellen der Datenbank dar und beschreibt die Methoden `next( )` um eine Zeile zu lesen und `getXxx( )` für das Lesen eines Datums mit dem Typ *Xxx* aus einer Spalte der Tabelle. Konkrete Implentierungen dieser Methoden für eine spezielle Datenbank sind Bestandteil des zugehörigen JDBC-Treibers ([WFCHH99, S. 57]).

```
protected ValueObject findByPrimaryKey(Object primaryKey)
 throws SQLException, DataAccessException,
        ServiceLocatorException, NoDataFoundException {
  String userName = (String)primaryKey;
  UserVO userVO = new UserVO( );
  preparedStatement = connection.prepareStatement(
                                      sql.getScaffold( ) );
  preparedStatement.setString(1, userName);
  manageTransaction(supportsCMT( ), userName);
  resultSet = preparedStatement.executeQuery( );
  if (resultSet.next( )) {
    ...
    userVO.setEmail(resultSet.getString("email"));
    userVO.setPassword(resultSet.getString("password"));
    ...
    userVO.setRowVersion(resultSet.getLong("rowversion"));
  } else {
    throw new NoDataFoundException(
      "record id : " + userName + " could "
    + "not be found in the users table. " );
  }
  return userVO;
}
```

SQL-Statements Der SQL-Code ist in den DAOs nicht sichtbar. Im Paket `rdbaccess.util` sind die Klassen `SQLCode` und `SQLStatement` dafür deklariert. `SQLCode` ist ein Singleton ([GHJ97, S. 127]), das wenn die Anwendung startet, `Scaffolds` aus der Textdatei `config/sql.properties`, die sich dafür im WebSpace der Applikation befinden muß, in einen Cache läd. Dabei handelt es sich um SQL-Kommandos in denen Parameter durch `?` repräsentiert sind. Um daraus gültige SQL-Statements zu erzeugen, werden JDBC-*PreparedStatements* verwendet, in denen die `?` durch Parameterbelegungen ersetzt werden können ([WFCHH99, S. 63]). Da String-Instanzen in Java teuer sind, puffert der Cache die Kommandos, so daß die immer gleichen Anfragestrings nicht für jeden Zugriff neu angelegt werden müssen.

ServiceLocator Um JDBC-*Connection*s herzustellen, verwenden die DAOs den `rdbaccess.ServiceLocator` ([SJC02, S. 367]). Dieser wird im Persistence Framework auch für die Erzeugung von DAO-Instanzen selbst verwendet. Dazu ermöglicht der `ServiceLocator` den Zugriff auf Factories für *Connection* und *DAO* Objekte. Anstatt *DAO* und *Connection* Instanzen selbst anzulegen, werden die `create` Methoden dieser Factories aufgerufen. Die konkreten Klassen der erzeugten Objekte zu kennen, ist dazu nicht nötig. Die DAOs könnten also zu einem EJB-Server portiert werden, ohne daß der Client-Code davon betroffen wäre. Bei den `DataSources` sieht der Client außerdem nicht mehr, ob diese via JNDI geortet werden oder ob es sich um die für die Struts-Applikation konfigurierte Standard-Datenbank handelt. Intern verwendet der `ServiceLocator` eine `java.util.HashTable` als Cache für Referenzen auf die *Connection*- und *DAO*-Factories, damit diese nicht für jede Anfrage neu lokalisiert werden müssen ([BCJKNR02, S. 126]).

Data Access Façades Die CRUD-Methoden der DAOs können erst aufgerufen werden nachdem die aus der Basisklasse geerbten *Connection* und *Statement* Attribute initialisiert wurden. Die abstrakte Basisklasse *DAO* stellt dafür die Methoden `initAccess( )` und `prepareStatement( )` bereit.
`initAccess( )` bekommt als Parameter die *Connection* oder, in einer dazu überladenen Variante, einen Schlüssel, mit dem der `ServiceLocator` eine *Connection*-Factory auswählen kann, übergeben. Prinzipiell ist es somit möglich, für jede Transaktion eine eigene Datenbank anzugeben. Sinnvoll ist das, wenn für die einzelnen Knoten der Anfrage- und Parameterbäume unterschiedliche Datenbanken verwendet werden, so daß beispielsweise Konzerte in einer anderen Datenbank gespeichert sind als Vorlesungen oder beispielswiese Einträge mit dem *location*-Attribut Frankreich in einer anderen Datenbank als solche mit dem Attribut Deutschland.

Die DAO-Methode `cleanUp( )` schließt die Verbindung wieder oder, etwas genauer, läßt sie durch die entrspechend implementierte Schnittstellenmethode `close( )` für weitere Zugriffe in den *Connection*-Pool zurücklegen.

Die abstrakte Basisklasse `rdbaccess.daos.`*DataAccessFacade* stellt zu jeder CRUD-Methode eine gleichnamige Implementierung bereit, die dem in die Façade gepacktem DAO nacheinander die Nachrichten `initAccess( )`, `prepareStatement( )`, einen Aufruf der jeweils implementierten CRUD-Methode und abschließend `cleanUp( )` schickt. Außerhalb des Persistence Frameworks können die CRUD-Methoden in den DAOs nur durch Data Access Façades (im folgenden DAF) angesprochen werden.

Zu jeder konkreten DAO-Implementierung ist im Paket `rdbaccess.daos` auch eine konkrete Erbenklasse der DAF implemeniert. Zum `UserDAO` gehört also z.B. `UserDAF`. Diese konkreten DAFs können mit einem DAO, das sie umschließen sollen, instanziiert werden. Welches DAO also die gerade aufgezählten Nachrichten, die für einen Datenbankzugriff notwendig sind, erhält, wird erst mit der Instanziierung einer Erbenklasse konkretisiert. DAF stellt somit eine Template Implementierung dar ([GHJ, S. 325]).

In den konkreten Erbenklassen der DAF sind die CRUD-Methoden durch Varianten überladen, die nur ein *ValueObject* (oder gar nichts) als Parameter erwarteten. Diese greifen dann auf die mit der Initialisierung festgelegte Datenbank zu. Das Web-Interface verwendet ausschließlich diese Variante und initialisiert alle DAFs mit der gleichen Datenbank. Sollen die Daten auf mehrere Datenbankserver verteilt werden, muß das Web-Interface für jeden dieser Server einzeln installiert werden.

Die überladenen Varianten der CRUD-Methoden bedeuten ein Komfort-Plus, stellen aber nicht die eigentliche Stärke der DAFs dar. Solche Methoden könnten auch in den Erbenklassen der DAOs implementiert sein und man könnte sogar ganz auf die DAFs verzichten, wenn dort auch die benötigten *Connection*s und *Statement*s initialisiert und wieder freigeben würden.

Die primitiven CRUD-Methoden des DAO so zu überschreiben, wäre prinzipiell denkbar, wobei sich die *Connection* dann immer im sogenannten auto-commit Modus befände ([WFCHH99, S. 69]). D.h. durch die Ausführung jedes einzelnen SQL-Statements wird eine Transaktion beschrieben. Wenn sich die Datenbankinteraktionen eines Systems auf CRUD reduzieren lassen, stellt das auch kein Problem dar. Es ist aber gerade die Aufgabe eines Persistence Frameworks, Interaktionen mit den Datenbanken abstrahierend von den einzelnen Tabellen zu ermöglichen.

Transaktionen Façades sind dadurch charakterisiert, daß sie Gruppen von Aktivitäten zu Bestandteilen einzelner Methoden erklären. Diese Eigenschaft macht sie zum idealen Ort für die Handhabung von Transaktionen ([Cra02]).

Die DAF-Implementierungen enthalten daher die für den Tabellen-Typ des umschlossenen DAOs typischen transaktionalen Interaktionen. Außerhalb der DAFs sind das ganz normale `public` Methoden.

Wenn zusätzlich weitere Tabellen in eine Transaktion involviert sind, schickt die Façade den zugehörigen DAOs die Nachricht `makeDependent( )`, deren Verarbeitung in der abstrakten Superklasse so implementiert ist, daß sie danach die gleiche *Connection* verwenden. Das ist notwendig, da die Methode zum Ausschalten des auto-commit Modus eine Instanzmethode der *Connection* ist. DAOs verwenden intern die Methoden `enableCMT( )` und `disableCMT( )`, um das Transaktions-Managemant an die Façade zu übertragen.

`CMT` steht dabei für **C**lient **M**anaged **T**ransactions, wobei der DAO-Client die Façade ist[7]. Diese Methoden speichern den Status im autoCommit-Flag des DAOs, dessen Inhalt beim Aufruf der DAO-Methode
`manageTransaction( )` abgefragt wird, bevor diese den Zustand für das *Connection* Objekt setzt. `manageTransaction( )` wird von den konketen DAOs vor jeder CRUD-Methode aufgerufen. Aufgrund dieser Struktur darf das Connection-Handling auch für die primitiven CRUD-Methoden nicht innerhalb der DAOs vorgenommen werden, sondern ist in der DAF implementiert.

Alle CRUD-Aufrufe innerhalb einer transaktionalen DAF-Methode können

[7]das Akronym weist auch auf eine Analogie zur EJB-Terminologie hin, die es verwendet um **C**ontainer **M**anaged **T**ransactions zu bezeichnen.

bei diesem Design die gleiche *Connection* verwenden. Mißlingt einer dieser Aufrufe, wird die DAO Methode `setRollbackOnly( )` aufgerufen[8]. Diese markiert die gesamte Transaktion als gescheitert, so daß bei einem Commit-Versuch auch alle bis dahin gelungen Teilaktionen der Transaktion zurückgesetzt werden. Das Persistence Framework von WebTNDB gewährleistet also die von Transaktionen gefordete Eigenschaft Atomacy innerhalb der dort festgelegten Transaktionsgrenzen. D.h. von den Teilaktionen innerhalb dieser Grenzen finden entweder alle oder gar keine statt. Die übrigen von Transaktionen geforderten Eigenschaften, Consistency, Isolation und Durability (vgl. z.B. [MH01, S. 400]), sichert die relationale Datenbank selbst.

Grundsätzlich könnten bei dieser Implementierung nicht nur mehrere Tabellen, sondern auch mehrere Komponenten in Transaktionen involviert sein. Die Schnittstellen der Façade müßten dann aber feingranularer gehalten sein als die der konkreten DAFs von WebTNDB. Es müßte dadurch auch außerhalb der DAFs möglich sein, eine neue Transaktion zu starten, zu beenden und auf einzele Teilaktionen zuzugegreifen. Um die Transaktionen vor höheren Schichten zu verbergen, würde das eventuell die Implementierung weiterer Fassadenklassen auf der Business-Tier erfordern.

Die durch WebTNDB ausgelösten Transaktionen konnten alle in einzelne Methoden der DAFs gekapselt werden.

Für alle DAOs kann durch die in der Basisklasse definierte Methode `enableLogging( )` ein Logging eingeschaltet werden, das alle Datenbank-Interaktionen protokolliert. Um die Fehlersuche bei komplexen Transaktionen zu erleichtern, wird dieses Logging bei allen transaktionalen Methoden eingeschaltet. Das Web-Interface läßt dieses Protokoll ins Log-File des Web-Containers schreiben. Das ist die Datei `catalina.out` im Unterverzeichnis `logs` der Tomcat-Distribution. Alle Transaktionen können durch Einsicht dieser Datei mitverfolgt und nachvollzogen werden.

Ein Beispiel für eine transaktionale Methode aus der schon bekannten Klasse UserDAF ist die Methode `copyFunctionTree( )`, die einen Anfragebaum und einen Parameterwald für einen Benutzer kopiert und in den Datentabellen speichert. Der erste Parameter für diese Methode ist ein Datenbank-Schlüssel. Der letzte gibt das Ziel als Objekt vom Typ `UserVO` an. Ein `UserVO` repräsentiert einen Benutzer, bestimmt in diesem Fall also den Benutzer, dem die Anfragedaten zugeordnet werden sollen.

[8]Die Schnittstelle *EJBContext* über die in EJB-Implementierungen auf den Container zugegriffen wird, hat eine Methode mit der gleichen Funktion und dem gleichen Namen.

```
public void copyFunctionTree( int dataBaseKey, Map function,
                              Map queryMap, UserVO destination )
 throws DataAccessException, ServiceLocatorException {
  DAOFactory queryFactory = getDAOHome(ServiceLocator.QUERY_DAO),
             functionFactory = getDAOHome(ServiceLocator.FUNCTION_DAO);
  FunctionDAO functionDAO = (FunctionDAO)functionFactory.createDAO(true);
  QueryDAO queryDAO = (QueryDAO)queryFactory.createDAO(true);
  String ownerName = destination.getUserName( );
  functionDAO.initAccess(dataBaseKey); queryDAO.makeDependent(functionDAO);
  functionDAO.enableLogging( );  queryDAO.enableLogging( );
  Iterator functionValues = function.values( ).iterator( ),
           parameters = queryMap.keySet( ).iterator( );
  try {
   while (functionValues.hasNext( )) {
    FunctionVO branch = (FunctionVO)functionDAO.createValueObject( );
    branch.setQuery((String)functionValues.next( ));
    branch.setOwnerName(ownerName);
    functionDAO.prepareStatement("functiondao.insert", SQLStatement.INSERT);
    functionDAO.insert(branch);
   }
   while (parameters.hasNext( )) {
    String parameter = (String)parameters.next( );
    Collection branches = ((Map)queryMap.get(parameter)).values( );
    Iterator queryValues = branches.iterator( );
    while (queryValues.hasNext( )) {
     QueryVO branch = (QueryVO)queryDAO.createValueObject( );
     branch.setParameter(parameter);
     branch.setQuery((String)queryValues.next( ));
     branch.setOwnerName(ownerName);
     queryDAO.prepareStatement("querydao.insert", SQLStatement.INSERT);
     queryDAO.insert(branch);
    }
   }
  } catch (SQLException sqle) {
   functionDAO.setRollbackOnly(sqle);
   throw new DataAccessException(
               "Error in user-DAO.copyFunctionTree( ) : ", sqle );
  } finally {
   functionDAO.commit(queryDAO);
   functionDAO.cleanUp( );
  }
  destination.setFunction(function);
  destination.setParameters(queryMap);
}
```

Die Bäume sind durch die *Map*-Parameter spezifiziert. Die `function-`*Map* enthält alle Pfade des Anfragebaums mit ihrem Zeilen-Index in der `functions`-Tabelle als Schlüssel. Die `queryMap` ist eine *Map* von *Map*s. Die Parameternamen sind dort Schlüssel für den entsprechenden Baum mit der gleichnamigen Wurzel. Die einzelnen Bäume sind wieder durch *Map*s von Pfaden mit Zeilen-Indizes, hier natürlich in der `queries`-Tabelle, als Schlüssel gespeichert.

Instanzen von `UserVO`, die intern zur Repräsentation von Benutzerkonten verwendet werden, haben *Map*-Attribute mit genau dieser Struktur, die zur Speicherung der mit den Konten assoziierten Anfrage- und Parameterbäumen dienen. Bei der Klasse `UserVO` handelt es sich um die Implementierung eines Value Objects.

Nach der ersten Anmeldung eines neu aktivierten Benutzers ruft WebTNDB eine Variante von `copyFunctionTree( )` (`copyDefaultFunction( )`) auf, die ihm für seine erste Sitzung den Funktionsbaum und die Parameterbäume des „default"-Benutzers zuweist.

Value Objects Value Objects werden als Puffer für Datensätze verwendet und zwischen den Schichten ausgetauscht, um Datensätze zu übertragen[9]. Tatsächlich handelt es sich dabei um einfache Java-Beans[10], die nur aus Methoden zum Laden und Speichern von Attributbelegungen bestehen.

Auf den höheren Schichten repräsentieren Value Objects anwendungsspezifische Entitäten. Für WebTNDB also z.B. Benutzerkonten oder Anfragebäume. Die Belegung von Attributwerten mit Daten aus den Spalten einer oder mehrerer Tabellen geschieht ausschließlich innerhalb des Persistence Frameworks. Ebenso umgekehrt, die Speicherung von Value Objects durch Abbildung ihrer Attribute auf die Spalten der entsprechenden Tabellen.

Durch Value Objects wird also von den Schemata der Tabellen abstrahiert. Das `UserVO` hat z.B. die Methoden `getFunction( )` und `getParameters( )` mit dem Ergebnistyp *Map*. Dadurch kann eine Business-Methode, durch Zugriff auf ein solches `UserVO` auf den einem Benutzer zugeordneten Anfragebaum oder den Parameterwald in der zuvor beschriebenen Darstellung zugreifen - unabhängig davon wie diese Daten in den Tabellen gespeichert sind.
Gelegentlich liest man, daß Value Objects dazu dienen, die Performanz eines Persistence Framworks zu erhöhen. Das ist richtig, wenn die DAOs als EJBs implementiert sind : Hier ist für den Aufruf einer CRUD-Methode i.a. jedesmal ein teurer Remote-Zugriff notwendig, da sich das DAO prinzipiell auf einem entfernten Rechner befinden könnte[11].

[9] Value Objects werden in der Literatur machmal auch als Transfer Objects (z.B. in [SJC02]) oder Data Transfer Objects, kurz DTOs, bezeichnet (z.B. in [Cra02]).

[10] Mit Java-Beans sind hier natürlich einfache Java Klassen gemeint, die den in [CWH00] beschriebenen Konventionen für Java-Beans genügen. Da EJBs in der J2EE-Literatur gelegentlich auch einfach Beans genannt werden, soll hier ausdrücklich darauf hingewiesen werden, daß das dadurch beschriebene, komplexe Komponenten-Framework ein von den Java-Beans unabhängiges und völlig unterschiedliches Konzept darstellt !

[11] Das gilt nicht für die mit der EJB-Spezifikation 2.0 eingeführten Local-Interfaces ([MH01, S. 25]), auf die hier jedoch nicht näher eingegangen wird.

Sind die Schnittstellen der DAOs so feingranular, daß dadurch auf einzelne Spalten einer Tabelle zugegriffen wird, erfordert die Verarbeitung einzelner Tabellenzeilen u.U. eine Menge Netzwerkverkehr. Mit Value Objekts läßt sich dieser erheblich reduzieren, da ganze Datensätze nach einem einzigen entfernten Methodenaufruf übertragen werden.

WebTNDB profitiert von den Value Objects nur durch deren Abstraktion von den Schemata der Tabellen. Bei einer Portierung zu einem anderen Applikations-Server, kann dieses Design aber von Anfang an Performanzvorteile bei der Nutzung des Frameworks einbringen.

Beziehungen und referenzielle Integrität Beziehungen zwischen Datensätzen werden in Tabellen durch Fremdschlüssel gespeichert. Objektorientierte Systeme können Assoziationen durch Objektreferenzen darstellen. Das Persistence Framework baut mithilfe von Objektreferenzen Aggregationen von Value Objects zusammen, um Beziehungen zu repräsentieren. Anfragen können also die Instanziierung von mehr als einem Value Object zur Folge haben. Entsprechend muß ein DAO bei der Speicherung eines Value Objects eventuell auch die Attributwerte mit dem Parameter assozierter Value Objects berücksichtigen und zusätzliche Tabellen aktualisieren.

Die Methode `findByPrimaryKey( )` des UserDAOs impliziert z.B. die Instanziierung eines `RoleVO` zur Repräsentation der Benutzerrolle. Wird das `UserVO` wieder in die Datenbank zurückgeschrieben, sind eventuell auch Änderungen an dem damit assozierten `RoleVO` zu berücksichtigen.

Die abstrakte Basisklasse `rdbaccess.vos.`*`ValueObject`* definiert daher für alle konkreten Value Object Implementierungen die boolschen Attribute `insertFlag`, `updateFlag` und `deleteFlag`. Diese Markierungen werden dazu verwendet, geänderte Value Objects zu kennzeichnen. Alle `setter`-Methoden eines Value Objects setzten das `updateFlag` auf `true`. Die DAO-Methode `update( )` kann durch einen Aufruf von `getUpdateFlag( )` folglich auch für jedes Value Object, das mit dem ihm übergebenen assoziiert ist, feststellen, ob daran Veränderungen vorgenommen wurden und die entsprechenden Tabellen aktualisieren.

Mit dem `insertFlag` und dem `deleteFlag` können neu angelegte Value Objects für das Einfügen, bzw. nicht mehr benötigte für das Löschen in den zugehörigen Tabellen markiert werden. Ein einziger `delete( )`-Aufruf kann also einen kaskadierenden Delete-Vorgang zur Folge haben. Wird z.B. ein Selektor gelöscht, können auch alle zugehörigen Spezifizierer aus den Meta-Datentabellen entfernt werden.

Das ist auch der Grund, aus dem alle CRUD-Methoden mit einem Value Object parametrisiert sind : Ein DAO kann daraus Beziehungen zu anderen Value Objects ablesen und Folgeaktionen zur Wahrung der referenziellen Integrität automatisch auslösen. Für die Methode `delete( )` wäre sonst natürlich der Primärschlüssel als Parameter ausreichend.

Sperrverfahren Bei Datenbanktransaktionen verwendet die Datenbank ein pessimistisches Sperrverfahren, das die Korruption von Datensätzen durch Interferenzen ausschließt. Das Persistence Framwork von WebTNDB implementiert für nicht transaktionale, mehrstufige Vorgänge ein zusätzliches Sperrverfahren :

In `rdbaccess.vos.`*`ValueObject`* ist für jedes Value Object das Attribut `rowversion` vereinbart. Diese Zeilenversion wird bei jedem `update( )`-Aufruf inkrementiert und überprüft, um sicherzustellen, daß der Datensatz in der Tabelle nicht bereits aktueller ist als der Parameter. Ist dies der Fall, wird nichts gepeichert und eine `OptimisticLockException` ausgelöst.

Wie der Name sagt, handelt es sich hierbei also um ein optimistisches Sperrverfahren : Es wird davon ausgegangen, daß Konflikte durch konkurierende `update( )`-Versuche kaum auftreten. Daher wird nicht jeder Datensatz in der Bearbeitung durch einen Benutzer von vornherein für alle Zugriffe durch andere Benutzer gesperrt.

Das Web-Interface profitiert von dieser Implementierung z.B. beim „edit"-Durchlauf, also dem mehrstufigen Vorgang zur Bearbeitung eines Datensatzes. Grundsätzlich können mehrere Benutzer gleichzeitig auf verschiedene Seiten des Editors für ein und denselben Datensatz zugreifen, diese zur Detailansicht nutzen und über die „Cancel"-Schaltfläche wieder verlassen. Zugriffe auf die Datenbank bei der Navigtion durch die einzelnen Editorseiten finden alle im autocommit Modus statt, da es sich bis zur letzten Seite ohnehin ausschließlich um Nur-Lese-Zugriffe handelt. Erst wenn ein Benutzer auf der letzten Editorseite auf „Submit" klickt, werden „Submit"-Versuche anderer Benutzer, deren Detailansicht die letzten Änderungen noch nicht wiedergibt, zurückgewiesen. WebTNDB fängt die ausgelöste `OptimisticLockException` ab und führt die betroffenen Benutzer daraufhin zur ersten Editorseite zurück, die dann bereits die aktualisierten Daten anzeigt. Dort wird der Benutzer aufgefordert, seine Änderungen erneut durchzuführen.

Für die Inkrementierung der Zeilenversion wird intern eine zusätzliche Tabelle namens `counter` verwendet. Im Paket `rdbaccess.util` befindet sich die Klasse `Counter`, in der Methoden für den Zugriff auf diese Tabelle implementiert sind. Diese verwenden die MySQL-Funktion `LAST_INSERT_ID( )`, die den im Parameter übergebenen Ausruck auswertet und das Ergebnis zurückliefert. `Counter` selektiert mittels `LAST_INSERT_ID(seqnum + 1)`, also die Spalte `seqnum`, inkrementiert den Wert und speichert das Ergebnis. MySQL garantiert dabei wechselseitigen Ausschluß, so daß diese Methode für die Generierung eindeutiger Sequenznummern verwendet werden kann ([BCJKNR02, S. 142]).

Das Persistence Framework verwendet `Counter` daher auch für die Generierung eindeutiger Zeilenindizes, in `INSERT`-Statements auf Tabellen, die Zeilenindizes als Primärschlüssel benutzen. Die Funktion `LAST_INSERT_ID( )` ist MySQL spezifisch und muß bei der Migration zu einer anderen Datenbank in der Datei `config/sql.properties` eventuell durch die dort gegebene Funktion zur Generierung eindeutiger Sequenznummern ersetzt werden ([BCJKNR02, S. 151]).

3.2.3 Zusammenfassung

Auf der Integration-Tier einer Anwendung stellt insbesondere die Integration von Datenbanken ein komplexes Problem dar. Eine Implementierung muß Lösungen für zahlreiche datenbanktypische Teilprobleme, wie die Generierung von Primärschlüsseln und die Umsetzung von Sperrverfahren, beinhalten. Besonders aufwendig sind effiziente Lösungen für Datenmodelle, in denen Tabellen in unterschiedlichen Beziehungen zueinander stehen und weiterhin Transaktionen berücksichtigt werden müssen.

Ein einheitliches Persistence Framework erleichtert durch Zentralisierung aller Datenbankzugriffe die Migration zu unterschiedlichen Datenbanken und kann vor allem auch Abhängigkeiten von den Schemata der Tabellen in der Datenbank minimieren.

Erprobte Integration-Tier Design-Patterns sind zahlreich in der Fachliteratur dokumentiert und deren Studium kann enorm dazu beitragen das Verständnis und die Implementierung eines Persistence Frameworks zu erleichtern.

Das Persistence Framework von WebTNDB implementiert für jede Tabelle eine `*DAO`-Klasse, eine `*DAF`-Klasse und eine `*VO`-Klasse. Am Beispiel der `users`-Tabelle wurden in diesem Abschnitt typische Teile dieser Komponenten, die in den entsprechenden anderen Klassen des Persistence Frameworks analog umgesetzt sind, vorgeführt. Die wichtigsten der dabei implementierten Design-Patterns sind die J2EE-Patterns Data Access Object ([SJC02, S. 390]) und Data Transfer Object ([SJC02, S. 261]) und das GoF-Pattern Façade ([GHJ97, S. 185]).

Will man Erweiterungen am Persistence Framework vornehmen, so sollten einem diese Patterns vertraut sein, um die Schnittstellen aller Komponenten zu verstehen.

Für Anwender des Persistence Framework sind auf darüberliegenden Schichten nur die DAF und VO Klassen relevant. Wenn man nur diese verwendet, kann einem das Persistence Framework viele Schritte zur Lösung datenbanktypischer Probleme abnehmen.

Die DAFs stellen öffentliche Methoden für alle benötigten Zugriffe auf die Datenbank bereit. Als Rückgabewerte und Parameter werden Instanzen der VO Klassen verwendet, die höheren Schichten zur Pufferung von Datensätzen zur Verfügung stehen.

Von den komplexen Vorgängen zur Aufrechterhaltung der referenziellen Integrität und dem Management von Transaktionen auf Code-Ebene wird dadurch vollständig abstrahiert. Ein Transaktionsprotokoll, anhand dessen Transaktionen innerhalb des Persistence Frameworks mitverfolgt werden können, wird automatisch erstellt.

Wie sich DAF-Instanzen mithilfe des ServiceLocators ([SJC02, S. 367]) beschaffen lassen, wird im Abschnitt über die Presentation-Tier erklärt.

3.3 Die Anwendungsschicht

Die Pakete `data`, `parser` und `graphics` machen die Anwendungsschicht oder Business-Tier von WebTNDB aus. Die Klassen in diesen Paketen und ihren Subpackages stellen die Funktionalität für die Verarbeitung der Bäume bereit.

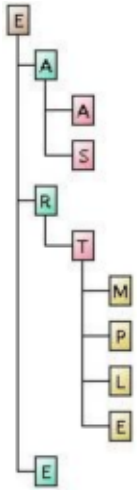
Das Paket `data` enthält Klassen, die für die interne Darstellung von Bäumen verwendet werden, und Filter, die aus Bäumen SQL-Anfrageselektoren, also „`WHERE`"-Klauseln erzeugen.

`parser` enthält die Klasse `Parser`, die ':'-separierte Pfade in Baumstrukturen übersetzt. Im Paket `graphics` sind die Klassen, mit denen die grafische Darstellung der Bäume generiert wird, zusammengefasst.

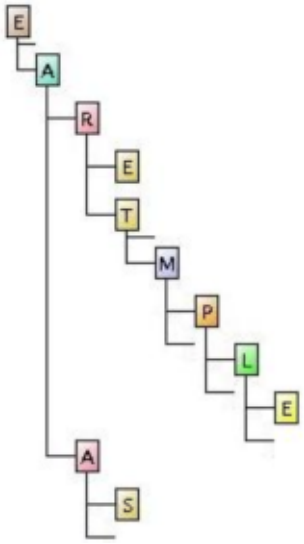
3.3.1 Die interne Darstellung der Bäume

Intern speichert WebTNDB ausschließlich Binärbäume, d.h. Bäume, bei denen jeder Knoten maximal zwei Kindknoten hat. Eine überraschende Tatsache, die auch WebTNDB ausnutzt, ist, daß sich bei einer entsprechenden Interpretation jeder beliebige Baum als Binärbaum darstellen läßt ([Sed92, S. 67]).

Um einen Baum in diese Darstellung zu überführen, wird von der Wurzel beginnend ein Kindknoten des entstehenden Binärbaums mit dem direkten Nachfolger des entsprechenden Knotens im ursprüglichen Baum belegt und der zweite Kindknoten mit dem ersten Schwesterknoten aus dem ursprünglichen Baum. Ein Beispielbaum könnte also z.B. so aussehen :



Eine Transformation in einen Binärbaum, der die gleiche Information enthält, läßt sich nach dem soeben beschriebenen Verfahren durchführen und ergibt für obiges Beispiel einen Baum, der sich wie folgt darstellen lässt :



Liest man umgekehrt in dieser Darstellung Pfade entlang der zweiten Kindknotenreferenz als Schwesterknoten des ersten Knotens eines solchen Pfades, ergibt sich daraus wieder der ursprüngliche Baum. Natürlich müssen die beiden Kindknoten dafür voneinander unterscheidbar sein. Im Code ist das anhand der zwei unterschiedlichen Referenzen möglich. Im obigen Bild ist der erste Kindknoten eines Knotens immer unter dem zweiten dargestellt.

`data.tree.TreeNode` und `data.tree.Tree`. Bei `TreeNode` handelt es sich um eine rekursive Datenstruktur zur Darstellung einer dreifach verketteten Liste. `TreeNode` hat also drei Attribute, die selber den Typ `TreeNode` haben. Dadurch werden der Vaterknoten, der erste Kindknoten und der zweite Kindknoten referenziert. `TreeNode` dient zur Darstellung eines einzelnen Knotens in einem Binärbaum.

Daß es sich dabei um einen Binärbaum handelt, ist nur für die Algorithmen im Paket `data.tree` relevant. Die außerhalb des Pakets benutzten Methoden, wie `addChild( )` zum Hinzufügen von Kindknoten an einen Knoten, beziehen sich alle auf Bäume, bei denen jeder Knoten beliebig viele Kindknoten haben kann. Also auf einen Baum, wie er sich ergibt, wenn man einen gespeicherten Binärbaum, wie in obigem Beispiel erklärt interpretiert.

Ein Attribut namens `data` dient zum Speichern eines Datensatzes für jeden Knoten und ist als `java.lang.Object` deklariert, so daß es sich dabei prinzipiell um jeden Datentyp handeln kann. Die Klasse `data.tree.Tree` implementiert alle Schnittstellenmethoden einer `java.util.`*`AbstractCollection`* für beliebige Bäume und verwendet dabei intern `data.tree.TreeNode` instanzen. Die Methode `add( )` fügt Kindknoten aber immer an den Wurzelknoten an, so daß man zur Verarbeitung von Bäumen mit einer Tiefe von mehr als zwei Knoten immer auch Referenzen vom Typ `data.tree.TreeNode` für die Addressierung einzelner Knoten speichern muß.

Traversierung Zur iterativen Verarbeitung aller Knoten eines Baums stellt WebTNDB zwei Möglichkeiten bereit, mit deren Hilfe ein Durchlauf in unterschiedlichen Reihenfolgen möglich ist.

Die erste Variante ist ein Breitendurchlauf (Level-Order-Traversierung, [Sed92, S. 71]), bei dem alle Knoten, Ebene für Ebene, der Reihe nach durchlaufen werden.

Die *Collection*-Methode `iterator( )` in `data.tree.Tree` liefert eine Referenz auf einen
`data.tree.TreeIterator`, der die Schnittstelle `java.util.`*Iterator* für einen solchen Breitendurchlauf implementiert.
Breitendurchlauf bezieht sich dabei natürlich auf den Baum aus Anwendersicht und nicht auf seine interne Darstellung als Binärbaum.

Ein Breitendurchlauf ist nötig, um Bäume in „`WHERE`"-Klauseln für SQL-Statements zu übersetzen, bei denen ein Abstieg von Ebene zu Ebene eine „`AND`"-Erweiterung und das Wandern von Knoten zu Knoten auf einer Ebene eine „`OR`"-Erweiterung des Ausdrucks zur Folge hat. So entstehen aus Anfragebäumen SQL-Ausdrücke wie z.B. „`WHERE (type='event' AND selector='concert') OR (type='event' AND selector='lecture')`" mit dem *event*-Einträge vom Typ *concert* und *lecture* selektiert würden[12].
Der in `data.tree.TreeIterator` implementierte Algorithmus, muß natürlich die interne Darstellung des Baums berücksichtigen. Dabei werden zwei Instanzen der Hilfsklasse `data.Queue`, die eine einfache Warteschlange implementiert, verwendet. Die Schleife über alle Knoten versucht immer zunächst dem Pfad entlang der ersten Kindknotenreferenz zu folgen. Ist diese mit einem Knoten belegt, wird dieser in die erste Warteschlange eingereiht, um ihn für die Verarbeitung beim Abstieg auf den nächten Level vorzumerken. Anschließend folgt der Algorithmus der zweiten Kindknotenreferenz und prüft, ob diese mit einem Knoten belegt ist. Ein solcher wird gegebenenfalls in die zweite Warteschlange eingereiht. Dort reiht er sich in der für den Breitendurchlauf gewünschten Reihenfolge ein, da es sich um dabei, in der Interpretation des Anwenders, um den nächsten Schwesterknoten seines Vorgängers im Breitendurchlauf handelt. Wird kein weiterer Kindknoten auf diesem Pfad gefunden, steigt der Algorithmus zum nächsten Level ab, indem er mit dem vordersten Knoten der ersten Warteschlange fortfährt. Der Algorithmus bricht ab, nachdem ein Knoten ohne Belegung für die zweite Kindknotenreferenz gefunden wird und auf der ersten Warteschlange keine Knoten für einen weiteren Abstieg mehr vorgemerkt sind. Also nach der Verarbeitung des letzten Blattknotens auf der untersten Ebene. In der zweiten Warteschlange befinden sind anschließend alle Knoten, in der für den Breitendurchlauf gewünschten Reihenfolge gespeichert. Die Zeitkomplexität des Verfahrens ist linear in der Zahl der Knoten, wie man der Implementierung

[12] Das Beispiel verwendet eine vereinfachte Darstellung, um das Verständnis des Ausdrucks zu erleichtern. Ein von WebTNDB für diese Selektion tatsächlich generierter Ausdruck würde natürlich statt '`event`' die typeID `1` enthalten, statt `selector` den Bezeichner `firstinfocolumn`, der die erste Spalte referenziert, die im Schema '`event`' zur Angabe der Kategorie dient, und statt '`concert`' und '`lecture`' die numerischen IDs `1` und `3` aus der Meta-Datentabelle `columnvalues`.

unschwer ansieht, da mit jeder Iteration ein Knoten in die zweite Warteschlange eingereiht wird und damit abgearbeitet ist :

```
public TreeIterator(TreeNode root) {
 levelOrder = new Queue( );
 nodeQueue = new Queue( );
 if ((currentNode = root) != null)
  levelOrder.enqueue(currentNode);
 TreeNode next, node = currentNode;
 while (node != null) {
  if ((next = node.getFirstDescendant( )) != null)
   nodeQueue.enqueue(next);
  if ((next = node.getFollowingSibling( )) != null)
   levelOrder.enqueue(node = next);
  else if (!nodeQueue.isEmpty( ))
   levelOrder.enqueue(node = (TreeNode)nodeQueue.dequeue( ));
  else
   node = null;
 }
 treeIterator = levelOrder.iterator( );
}
```

Die zweite Möglichkeit zur Traversierung eines Baums ist ein Tiefendurchlauf (Preorder-Traversierung, [Sed92, S. 68]). Dabei wird solange zum ersten Knoten auf der nächsttieferen Ebene abgestiegen, bis die tiefste erreicht ist. Danach läuft man jedesmal zum ersten Knoten auf einer höheren Ebene zurück, der einen Schwesterknoten hat, und startet von diesem Schwesterknoten wieder in die Tiefe bis alle Knoten besucht wurden.

Bezogen ist der Tiefendurchlauf natürlich auch wieder auf die Anwendersicht des Baums. Bei der internen Speicherung als Binärbaum ließ sich für diese Variante aber ein deutlich einfacheres, rekursives Verfahren implementieren.

Den Tiefendurchlauf verwenden der Parser für Pfade und der Renderer, der die grafischen Darstellungen der Bäume erzeugt.

Der Algorithmus ist in der Klasse `data.util.TreeTraverser` in die `private` Methode `depthFirst` gekapselt. Diese ruft sich in einer Iteration über alle Knoten einer Ebene (im Binärbaum also auf dem Pfad entlang der zweiten Kindknotenreferenz) solange selber für den ersten Kindknoten auf (folgt im rekursiven Abstieg also der ersten Kindknotenreferenz im Binärbaum), bis sie auf der untersten Ebene angekommen ist (im Binärbaum also bis die erste Kindknotenreferenz eine Nullreferenz ist). Der dadurch bedingte rekursive Aufstieg führt die Methode soweit zurück, bis die Iteration auf einer Ebene fortgesetzt werden kann. Also bis zum letzten Aufruf mit einem Knoten dessen zweite Kindknotenrefenrenz in der Binärbaumdarstellung keine Nullreferenz war und folglich, in der Interpretation des Anwenders, dort ein Schwesterknoten addressiert wird, von dem aus erneut rekursiv abgestiegen wird :

```
private void depthFirst(TreeNode node, int depth) {
 visitor.visit(node, depth);
 if (node.getFirstDescendant( ) != null)
  for ( TreeNode child = node.getFirstDescendant( );
        child != null;
        child = child.getFollowingSibling( ) ) {
   depthFirst(child, ++depth); --depth;
  }
}
```

Was bei der Verarbeitung eines Knotens mit diesem geschieht, wird auch hier mit einer Schnittstelle spezifiziert. Das Interface `data.util.`*`Visitor`* beschreibt die Signatur einer Methode *`visit( )`*, wie sie von jeder Klasse implementiert werden muß, deren Instanzen die Knoten eines Baums im Tiefendurchlauf verarbeiten sollen :

```
public interface Visitor {
  public void visit(TreeNode node, int depth);
}
```

Der `TreeTraverser` Konstruktur ist durch einen `data.tree.Tree` und ein Objekt mit der Schnittstelle *`Visitor`* parametrisiert und speichert Referenzen darauf in seinen `private` Attributen `tree` und `visitor`. Die öffentliche Methode `start( )` startet einen Tiefendurchlauf durch einen Aufruf von `depthFirst(tree.getRoot( ), 0)`. Folglich startet dadurch die Rekursion, so daß für jeden Knoten des Baums, in der Reihenfolge des Tiefendurchlaufs einmal die Methode `visit( )` mit einer Referenz auf diesen Knoten und mit der Anzahl der Knoten auf dem Pfad zurück bis zur Wurzel (also dessen Tiefe im Baum) als Parameter aufgerufen wird, deren konkrete Implementierung durch die Klasse des bei der Instanziierung angegebenen *`Visitor`*-Objekts beschrieben wird.

Der Parser In den Tabellen `functions` und `queries` werden die Pfade der Anfrage- und Parameterbäume als ':'-separierte Zeichenketten gespeichert. Zusätzlich befindet sich in beiden Tabellen die Spalte `owner`, die Gruppen von Pfaden zusammenfasst, die einen Anfragebaum eines Benutzers bilden. Die Tabelle `queries` hat zusätzlich die Spalte `parameter`, um einzelne Bäume im Parameterwald eines Benutzers zusammenzufassen.

Die Methoden `findBranchesByOwner( )` der Klasse `FunctionDAF` und `findParametersByOwner( )` der Klasse `QueryDAF` liefern *`Collection`*-Referezen auf die Anfragebäume eines Benutzers in der bekannten *`Map`*-Darstellung.

Der Pfad-Parser wird dafür verwendet, aus *`Map`*s von Pfaden Instanzen von `data.tree.Tree` zu erzeugen.

Im Paket `parser` ist in der Klasse `Parser.Transformer` für die Klasse `Parser` das *`Visitor`*-Interface so implementiert, daß nach einem Tiefendurchlauf die Position feststeht, an der neue Knoten in einen Baum eingefügt werden

müssen, um einen Pfad hinzuzufügen. Der Pfad muß dafür in Form eines Arrays, dessen Elemente die einzelnen Knoten speichern vorliegen und mindestens ein Pfad muß bereits als `Tree` Instanz gepeichert sein. Die Methode `visit( )` speichert für jeden neuen Pfad den Index im Pfad-Array, ab dem neue Knoten gespeichert sind, und eine `TreeNode` Referenz auf den Knoten, an dem der erste neue Kindknoten angehängt werden muß. So können Pfade beliebiger Länge in beliebige Bäume eingefügt werden, vorausgesetzt alle haben den gleichen Wurzelknoten. Da dabei etliche Fallunterscheidungen getroffen werden müssen, ist die Implementierung der Methode `visit( )` für den Parser etwas kompliziert und wird hier nicht im Detail besprochen.

Auf der Präsentationsschicht wird ohnehin niemals direkt mit der Klasse `Parser` gearbeitet.
Die Klassen `data.model.QueryFunction` und `data.model.QueryParameter` stellen einfachere Methoden der gemeinsamen Basisklasse `data.model.TreeQuery` bereit, mit deren Hilfe sich Repräsentationen von Anfrage- und Parameterbäumen erzeugen lassen.

`QueryFunction` und `QueryParameter` unterscheiden sich nur dadurch, daß `QueryParameter`-Objekte ein `name`-Attribut haben, dessen Belegung mit der des Wurzelknotens übereinstimmen muß. Eine Instanz einer Baumrepräsentation durch ein Objekt dieser Klassen kann mit deren parameterlosen Konstruktoren angelegt werden. Mit der Methode `addCommand( )` wird ein neuer Pfad hinzugefügt, der dabei in Form einer ':'-separierten Zeichenkette als Parameter übergeben wird. Nachdem alle Pfade eines Baums so für eine `QueryFunction`- oder `QueryParameter`-Instanz gepeichert wurden, ruft man nacheinander zwei ihrer Methoden auf, um die interne Baumdarstellung zu erzeugen : Durch die Methode `graft( )` wird der Parser initialisiert.
Danach kann mittels `pullUp( )` der Baum generiert werden. Eine Referenz auf den Baum läßt sich anschließend mit der Methode `getQueryTree( )` ermitteln.

```
data.tree.Tree parameterTree = null;
QueryParameter model = new QueryParameter( );
parameterBranches = (Map)parameterMap.get(parameter);
model.setName(parameter);
if (parameterBranches != null) {
  Collection pathExpressions = parameterBranches.values( );
  Iterator branches = pathExpressions.iterator( );
  while (branches.hasNext( ))
    model.addCommand((String)branches.next( ));
  model.graft( );
  model.pullUp( );
  parameterTree = model.getQueryTree( );
}
```

Wenn `parameterMap` eine *`Map`*-Darstellung des Parameterwalds ist und `parameter` ein Anfrageparameter, kann obiges Codefragment verwendet werden, um den zugehörigen Parameterbaum zu bestimmen.

3.3.2 Die grafische Darstellung der Bäume

Um die im Baumeditor angezeigten Bilder der Funktions- und Parameterbäume zu erzeugen, wird die Klasse `graphics.TreeDiagram` verwendet. Beim Anlegen einer Instanz von `graphics.TreeDiagram` müssen die Koordianten der linken oberen Ecke des Wurzelknotens, eine `data.tree.Tree`-Referenz auf den darzustellenden Baum und eine Referenz auf eine `data.model.NodeMap` angegeben werden.

`data.model.NodeMap` ist eine *`Collection`* von `data.Model.Node` Objekten. Ein `data.Model.Node` Objekt speichert die Koordinaten des Rechtecks, das einen Knoten darstellt, dessen Namen und seine Tiefe im Baum. WebTNDB verwendet `data.Model.Node` Objekte zur Pufferung der Daten eines Knotens, die für die Präsentationsschicht relevant sind. Dort dient z.B. eine Instanz von `data.model.NodeMap` zum Speichern von Pfaden, die im Pfadeditor bearbeitet werden.

`TreeDiagram` trägt beim Erstellen des Baumdiagramms die Koordinaten und Namen jedes dargestellten Knotens in die bei der Instanziierung angegebene `NodeMap` ein. Mit deren `containsNode( )` Methode läßt sich später feststellen, ob bei einem Mausklick auf ein Diagramm ein im darin dargestellen Baum enthaltener Knoten selektiert wurde. Die Parameter für `containsNode( )` sind die x- und y-Koordinate des Mausklicks und eine Referenz auf ein `data.Model.Node` Objekt. Wurde durch den Klick ein Knoten angewählt, liefert die Methode `true` zurück und speichert alle präsentationsrelevanten Daten über den Treffer im ihr übergebenen `Node`-Puffer.

Die in `TreeDiagram` gekapselte innere Klasse `TreeDiagram.Renderer` implementiert das *`Visitor`*-Interface zum Zeichnen einzelner Knoten.

Die Klasse `graphics.util.DiagramToolkit` enthält neben einigen Klassenmethoden zur Bestimmung der Abmessungen eines Baumdiagramms die statische Methode `drawImage( )`, die zur Generierung von Baumdiagrammen `TreeDiagram` Instanzen anlegt und damit Offscreen-Images von Bäumen erstellt.

Das Java Image Managing Interface JIMI ([Jim03]) stellt einen Encoder bereit, der solche Offscreen-Images durch JPeg-Kompression kodieren kann. `drawImage( )` verwendet diesen Encoder, um JPeg-kodierte Bilder von Baumdiagrammen auszugeben. Der JIMI-JPeg-Encoder ist auch Bestandteil der Java Image-I/O-Tools von denen zum Zeitpunkt der Erstellung dieser Arbeit ein 1.0-Release-Candidate verfügbar war und die voraussichtlich Bestandteil der Java Advanced Imaging API für Java 1.4 sein werden ([Jai03]).

In den auf der Präsentationsschicht generierten HTML-Formularen sind die grafischen Baumdarstellungen HTML `input`-Elemente vom Typ `image`, deren `src`-Attribut durch eine Referenz auf `service.TreeServlet/TreeDiagram.jpg` belegt ist. Als Quelle der Bilddaten ist dort also das Servlet `TreeServlet` im `service` Paket angegeben.

Dabei handelt es sich um ein sehr einfaches Servlet, das von allen anderen, in das Jakarta Struts Framework eingebetteten Präsentationskomponenten völlig unabhängig ist und daher bereits vorab gezeigt werden kann. Alle HTTP `GET`- oder `POST`-Requests an dieses Servlet führen zu einem Aufruf, der Methode

`drawImage( )`, die darauf ein Jpeg-kodiertes Baumdiagramm in den Client-`OutputStream` schreibt. Eine Instanz von `data.tree.Tree` muß dafür bereits im sogenannten Session-Scope referenziert sein. D.h. daß eine solche Referenz im `session`-Objekt, das der Container für jede Client-Sitzung anlegt, als Attribut gespeichert sein muß, damit das Servlet Zugriff auf die Daten hat, die den für den jeweiligen Client zu zeichnenden Baum beschreiben.

```
public class TreeServlet extends HttpServlet {
 Tree tree;
 public void init(ServletConfig config)
  throws ServletException {
   super.init(config);
 }
 public void doGet( HttpServletRequest request,
                    HttpServletResponse response )
  throws ServletException, IOException {
   doPost(request, response);
 }
 public void doPost( HttpServletRequest request,
                     HttpServletResponse response )
  throws ServletException, IOException {
   HttpSession session = request.getSession( );
   tree = (Tree)session.getAttribute("tree");
   response.setContentType("image/jpg");
   DiagramToolkit.initializeRenderingEngine( );
   Graphics context = DiagramToolkit.getContext(10, 10);
   Dimension size = DiagramToolkit.getSize(tree.getModel( ), context);
   DiagramToolkit.drawImage( tree.getModel( ), tree.getNodeMap( ),
                             size.width, size.height,
                             response.getOutputStream( ) );
 }
}
```

Eine Variante dieses Servlets ist als eigenständige Applikation zusätzlich im WebSpace des Testsystems unter dem Namen `testapp` verfügbar. Hier können testweise beliebige ':'-separierte Pfadausdrücke in ein Textfeld eingegeben werden. Das Servlet gibt diese an den Parser weiter und schickt die von den hier besprochenen Grafik-Utilities generierte Darstellung an den Client-Browser zurück.

3.3.3 Die Selektionsfilter

Die TNDB API enthält vier Typen von Selektionsfiltern, mithilfe derer das Web-Interface und der Proxy-Server gespeicherte Bäume und Anfrageintervalle auswerten. Dabei handelt es sich um die im Paket `data.filter` definierten Klassen `FunctionFilter`, `ParameterFilter`, `LocationFilter` und `IntervalFilter`.

`Function`- und `ParameterFilter` übersetzen die gespeicherten Bäume dabei in „`WHERE`“-Klauseln, die sich mithilfe der in der Klasse `rdbaccess.daos.EntryDAF` dafür vorgesehenen Methode `findAllMatches( )` zu SQL-`SELECT`-Statements für die Selektion der entsprechenden Einträge konkatenieren und auswerten lassen.

`Location`- und `IntervalFilter` filtern Listen von Einträgen, die bereits in der Datenbank selektiert wurden, und wählen davon nur die für den gespeicherten *location*-Parameterbaum und das gespeicherte Anfrageinterval gültigen Treffer aus.

Die Methode `findEntries( )` in `FunctionFilter` nimmt einen `ParameterFilter` und einen `LocationFilter` als Parameter und wertet dafür den zugehörigen Anfragebaum und alle Parameterbäume, also eine durch $f(p_1, ..., p_n)$ beschriebene Anfragefunktion, auf der Datenbank aus und liefert eine `java.util.`*`List`*-Referenz auf die gefundenen Ergebnisse zurück. Übergibt man diese Referenz an einen `IntervalFilter`, filtert dieser davon zusätzlich nur die Einträge heraus, die innerhalb des zugehörigen Intervals liegen.

Hat man Zugriff auf eine Methode `getDAF( )`, die zu einer Kennung die Data Access Façade zu der angeforderten Tabelle liefert, läßt sich wie in folgendem Code-Fragement gezeigt für den Benutzer `user` eine Selektion mithilfe der Filter-Klassen durchführen :

```
EntryDAF entryDAF = (EntryDAF)getDAF(ENTRIES);
DataAccessFacade valueDAF = getDAF(COLUMNVALUES);
FunctionFilter functionFilter =
  new FunctionFilter( user.getFunction( ),
          entryDAF, valueDAF, getDAF(TYPES) );
ParameterFilter parameterFilter =
  new ParameterFilter( user.getParameters( ),
          valueDAF, getDAF(SEASONS), getDAF(SPECIFIERS) );
LocationFilter locationFilter =
  new LocationFilter( getDAF(CONTINENTS), getDAF(COUNTRIES),
                      getDAF(STATES), getDAF(CITIES) );
IntervalFilter intervalFilter =
  new IntervalFilter(interval, Interval.NARROW);
List matches = functionFilter.findEntries(
                 parameterFilter, locationFilter );
List queryResults = intervalFilter.filterElements(matches);
```

Die Referenz `user` zeigt dabei auf ein Value Object vom Typ `UserVO`, repräsentiert also ein Benutzerkonto und ermöglicht mit den Methoden `getFunction( )` und `getParameters( )` *`Map`*-Darstellungen der damit assozierten Anfragebäume zu lesen, wie sie im vorherigen Abschnitt erklärt wurden.

`interval` referenziert eine Instanz der Klasse `data.model.Interval`, die zur Pufferung von Intervallen dient, wie sie im Intervaleditor durch Auswahl eines Start- und Endzeitpunkts angegeben werden können. Intern speichert ein `data.model.Interval` diese Zeitpunkte dafür in Objekten vom Typ

`java.util.Date`, also unabhängig von der Repräsentation, in dem bei ihrer Auswahl verwendeten Kalendersystem ([Lin98]).

Wie `getDAF( )` für die Komponeten der Präsentationsschicht implementiert ist, wird im folgenden Unterkapitel erklärt.

Hier wird abschließend noch auf verschiedene implementierungstechnische Details der einzelnen Filter eingegangen, um das Verständnis der Quelltexte zu erleichten :

In der Oberklasse `data.filters.Filter` ist die Methode `getEntryMap( )` implementiert. Diese liest eine ***Map*** in der Anfragepfade mit ihrem Zeilenindex als Schlüssel gespeichert sind, wie sie `getFunction( )` für ein `UserVO` liefert. Als Ergebnis wird eine ***Map*** zurückgegeben, in der Startknoten von Pfaden als Schlüssel für ***Map***s verwendet werden, in denen die Folgeknoten zusammengefasst sind. Für die Anfragepfade *event* : *exposition*, *event* : *concert* und *phenomenon* : *eclipse* bekäme man beispielswiese eine ***Map,*** die zum Schlüssel *event* eine weitere ***Map*** mit den Einträgen *exposition* und *concert* gespeichert hat, und zum Schlüssel *phenomenon* eine ***Map*** mit dem Eintrag *eclipse*. Die Methode `constructWhereClause( )` des `FunctionFilters` baut aus ***Map***s mit diesem Aufbau SQL-„`WHERE`“-Klauseln für `SELECT`-Statements zusammen. Für das Beispiel würde die Methode den String

```
„WHERE (type=1 AND firstinfocolumn=1)
     OR (type=1 AND firstinfocolumn=2)
     OR (type=3 and firstinfocolumn=6)“
```

zurückgeben. In der Basisklasse Filter sind auch die `encoder`-Methoden, die verwendet werden, um Selektor und Attributbelegungen numerisch zu kodieren. Der Parameter `offset` gibt an, wieviele führende Knoten in den Pfaden ignoriert werden sollen. `FunctionFilter` ruft `getEntryMap( )` mit dem offset `1` auf, um den gemeinsamen Wurzelknoten *entry* zu übergehen, der bei der Selektion von Einträgen keine Rolle spielt.

Der Ergebnisstring der Methode `constructWhereClause( )` wird als Parameter an die Methode
`appendConditionals( )` übergeben, die die zugehörigen Attributselektoren aus den Parameterbäumen hinzufügt. `appendConditionals( )` stellt dazu als erstes fest, welche Parameter für die Anfragefunktion relevant sind. Das sind immer die globalen Parameter *season* und *location* und weiterhin alle Bäume zu parametrisierten Selektortypen im Anfragefunktionsbaum. Im obigen Beispiel also die Parameterbäume mit den Wurzeln *eclipse*, *exposition* und *concert*.

Die für eine Anfrage relevanten Parameterbäume ergeben sich durch einen Join der Tabellen `functions` und `queries` mit der Join-Bedingung, daß Selektorknoten in Anfragepfaden aus der Tabelle `functions` mit dem Wert in der Spalte `parameter` der Tabelle `queries` übereinstimmen. Diese Vereinigung der beiden Tabellen kann als Selektion auf deren kartesischem Produkt verstanden werden. Die interne Darstellung der Anfragebäume und Parameterwälder durch Java-***Collection***s vom Typ ***Map*** erlaubt es, das Ergebnis dieser Vereinigung durch ein effizientes Verfahren zu bestimmen.

Anstatt jeden Selektor mit jedem Parameter zu vergleichen, kann man hier davon profitieren, daß Java für die Speicherung der *Collections* HashMaps verwendet. Der Parameterbaum zu einem Selektor wird bei einem Aufruf der Methode `get( )` also intern durch ein Hashing-Verfahren lokalisiert. Eine solche Join-Implementierung wird daher manchmal auch Hash-Join ([SH99, S. 337]) genannt :

```
private Map joinRelevants( ) {
 HashMap results = new HashMap( );
 Iterator selectors = relevants.iterator( );
 while (selectors.hasNext( )) {
  Object key = selectors.next( );
  results.put(key, query.get(key));
 }
 return results;
}
```

Für die einzelnen Parameterbäume wird im ParameterFilter wieder die Methode getEntryMap( ) aus der Basisklasse Filter verwendet, um eine Zerlegung zu erzeugen. getEntryMap( ) speichert zu jedem Knoten auch seinen Pfadindex, so daß die Belegungen des ersten und zweiten Attributs bei nicht-exklusiven Spezifizierern voneinander unterschieden werden können.
Wird beispielsweise durch Angabe des Parameterpfads *eclipse* : *lunar* : *total* spezifiziert, daß nur nach totalen Mondfinsternissen gesucht werden soll, kann die Methode appendConditionals( ) die WHERE-Klausel aus obigem Beispiel wie folgt umbauen :

```
„WHERE (type=1 AND firstinfocolumn=1)
   OR (type=1 AND firstinfocolumn=2)
   OR (type=3 AND firstinfocolumn=6
       AND firstcolumnfirstspecifier=13
       AND firstcolumnsecondspecifier=11)“
```

Die generierten SQL-Statements, die schnell ziemlich komplex werden können, werden auch alle ins Log-File des Servers geschrieben und können dort nachvollzogen werden. Dort wird auch für jede Anfrage mitprotokolliert, in welchen Regionen gesucht werden soll.

Die Klassen LocationFilter und IntervallFilter sind deutlich einfacher implementiert. Hier werden die gültigen Einträge in der Ergebnisliste anhand ihrer Attributwerte aussortiert. Da diese Selektion erst nach der Auswertung des SQL-Statements durchgeführt wird, ist zu beachten, daß für alle Einträge, die erst hier ausgefiltert werden, bereits Instanzen angelegt wurden. FunctionFilter und ParameterFilter sind daher die effizienteren Filter.

3.3.4 Zusammenfassung

Die Business-Tier oder Anwendungsschicht ist für die Verarbeitung von Anfragebäumen zuständig.

Intern werden dabei verschiedene Repräsentationen von Bäumen und ihren Knoten u.a. durch mehrfach verkettete Listen, *Map*s und *Collection*s von Zeichenketten verwendet. Für darüberliegende Schichten ist nur die logische Baumsicht auf die Daten relevant. Diese zu kennen genügt, um Bäume eindeutig referenzieren zu können, grafische Darstellungen davon zu generieren und die dadurch beschriebenen Anfragefunktionen auszuwerten.

Speziell für komfortable Benutzerobflächen werden auf der Business-Tier Instanzen der Klasse `NodeMap` verwaltet, die es ermöglichen zu überprüfen, ob durch einen Mausklick auf ein Baumdiagramm ein Knoten selektiert wurde.

Vier verschiedene Filterklassen ermöglichen es, Datenbank-Selektionen anhand von Funktionsbäumen, Parameterwäldern, Location-Bäumen und Intervallen durchzuführen. Die Filter für Funktionsbäume und Parameterbäume können von der internen Repräsentation der Daten profitieren und dynamisch SQL-Statements generieren.

Da unterschiedliche Typen der Regionalspezifikation möglich sind (Speicherung mit unterschiedlicher Präzision), steht für den *location*-Parameter ein eigener Selektionsfilter bereit. Dieser ermöglicht die Selektion unterschiedlicher Typen (Polymorphismus in dem zuvor erklärten Sinne), ist wesentlich einfacher implementiert als die beiden anderen Baumfilter, aber auch nicht so effizient, da eventuell unnötige Instanzen angelegt werden.

Die Bedienung der Filter ist sehr einfach, für alle Filter prinzipiell gleich und es ist dazu nicht nötig, die jeweilige interne Vorgehensweise zu kennen. Die Business-Tier abstrahiert hier also von den Selektionsverfahren, so daß sich daran Veränderungen durchführen ließen, ohne daß die Präsentationsschicht davon betroffen wäre.

3.4 Die Präsentationsschicht

Die Erstellung der Präsentationsschicht für eine Web-Applikation kann sehr viel Aufwand erfordern. Das Erscheinungsbild einer Anwendung trägt entscheidend zum Erfolg einer Anwendung bei. Außer einem ansprechendem Design, ist es wichtig, daß die Bedienung der Benutzerobfläche möglichst intuitiv ist.

Standalone Anwendungen können dazu auf reichhaltigen Bibliotheken aufbauen, mit deren Hilfe sich Benutzerschnittstellen erstellen lassen, deren Bedienelemente (Widgets) dem Anwender vom Umgang mit dem Window-Manager auf der verwendeten Plattform bereits vertraut sind z.B. Cocoa (Apple), MFC (Microsoft), JFC (Java), Motif (Solaris) oder QT (KDE). Für Benutzerinteraktionen stehen hier sehr komfortable Ein- und Ausgabemöglichkeiten, wie z.B. Scroll- and Slidebars, Tabs, Progress Indicators, Drag and Drop, etc. zur Verfügung.

Die Oberfläche einer Web-Applikation besteht dagegen aus einer Folge von dynamisch generierten HTML-Seiten. Folglich ist ein Konzept notwendig, das es erlaubt, mithilfe der rudimentären Elemente von HTML, wie Eingabefeldern, Auswahlfeldern und Schaltflächen eine möglichst flüssige Benutzerführung zu ermöglichen.

Bestimmte Interaktionen mit WebTNDB, wie die Registrierung eines neuen Benutzers oder das Bearbeiten von Anfragebäumen, Intervallen und Datenbankeinträgen, sind, was die Darstellung angeht, prinzipiell unabhängig voneinander. Da jeder Einzelschritt innerhalb einer solchen Interaktion auf eine neue Seite führt, sollte bei der Generierung dieser Seiten eine so konsistente Darstellung erreicht werden, daß der Seitenfluß den Anwender nicht bei seinen Eingaben stört. Im Idealfall entsteht dadurch die Illusion einer lokalen Anwendung, so daß der Anwender ein entferntes System bedienen und alle dort verfügbaren Informationen nutzen kann, ohne dabei Einschränkungen hinnehmen zu müssen.

Die Basisfunktionalität von WebTNDB, Anfragefunktionen, die durch Bäume spezifiziert sind auf temporalen Datenbeständen auszuwerten, wurde nicht in das Web-Interface integriert. Folglich kann auch unabhängig davon darauf zugegriffen werden, wie z.B. durch den Proxy-Server.

Daß die Trennung von Anwendungs- und Präsentationslogik die Flexibilität und Wiederverwendverwendbarkeit von Software fördert, ist inzwischen allgemein bekannt. Das architectural pattern MVC beschreibt prinzipiell genau diese Struktur. MVC steht für Model-View-Controller und bezeichnet eines der ersten dokumentierten Patterns für objektorientiere Software, das ursprünglich als Paradigma für eine geschickten Aufteilung verschiedener Klassen beim Entwurf von Benutzerstellen für Smalltalk-80 diente ([GHJ97, S. 4]) : Beim MVC-Design dienen View-Komponenten ausschließllich zur Darstellung und der Kontrollfluß innerhalb von Views wird durch Controller-Komponenten gesteuert. Controller und View müssen grundsätzlich unabhängig von der Komponente Model sein, wodurch im MVC-Design die interene Datenrepräsentation und Business-Logik einer Anwendung gemeint sind.

Die zuvor beschriebene Business- und Integration-Tier machen die Model-Komponente von WebTNDB aus. Eine ausführliche Diskussion verschiedener Ansätze für den Entwurf einer Web-Tier für J2EE-Anwendungen findet sich im 4. Kapitel der offiziellen Design-Guidelines von Sun Microsystems, „Web-Tier Application Framework Design“ [SSJ02].

Für die Realisierung von Views und Controller-Komponenten stellt die J2EE-Technologie unterschiedliche Möglichkeiten bereit. Die bewährteste Variante für MVC-Implementierungen auf J2EE-Basis ist dabei der sogenannte „Model 2 Approach“ ([ebenda]).

Dieser Ansatz geht von der Verwendung eines Servlet als Controller und von Java Server Pages als Views aus. Weiterhin beschreiben die Design-Guidelines zahlreiche Presentation-Tier Patterns, wie z.B. Front Controller, Composite View, Service To Worker, etc. also erprobte Entwurfsmuster zur Implementierung von Präsentationsobjekten nach dem „Model 2 Approach“, die Strategien beschreiben, um beispielsweise mittels sogenannter Templates ein konsistentes Erscheinungsbild zu erreichen.

Da die grundlegenden Problemstellungen einer Web-Schnittstelle unabhängig von den Aufgaben einer konkreten Anwendung gelöst werden können, ist es nicht nötig für jede Web-Applikation eine neue Model 2 Implementierung zu schreiben. Stattdessen verwendet man auf der Präsentationsschicht oft so-

genannte Front-End Frameworks[13], durch die eine solche Architektur bereits auf der Basis standardisierter Patterns implementiert ist. Erweiterungen der Basisklassen eines solchen Frameworks und verschiedene Konfigurationsdateien ermöglichen es, ein solches Framework soweit anzupassen, daß sich damit eine sehr professionelle Präsentationsschicht für die meisten Anwendungsfälle erstellen läßt.

Die Präsentationsschicht von WebTNDB wurde daher auf Basis des Frameworks Jakarta Struts implementiert. Im Folgenden wird vorausgesetzt, daß der Leser mit diesem Framework vertraut ist. Eine Einführung in die Grundlagen von Jakarta Struts anhand einfacher Beispiele ist im Anhang B zu finden.

3.4.1 Struts und WebTNDB

Die Presentation-Tier von WebTNDB ist vollständig in das Struts Framework eingebettet. Alle Komponeten dieser Schicht sind daher im Paket `struts` gespeichert. Im Subpackage `actions` sind die Actions von WebTNDB implementiert. Aus Platzgründen können die ca. 50 dort vereinbarten Klassen hier nicht einzeln dokumentiert werden.

Welche Aufgabe die einzelnen Actions haben, läßt sich jedoch, nachdem das System im 1. Kapitel dieser Arbeit ausführlich beschrieben wurde, einfach aus ihren Klassennamen ablesen. Das kurze Struts Tutorial in Anhang B vermittelt die nötigen Grundlagen, um die Implementierung der Actions anhand der Quelltexte nachvollziehen zu können. Für Detailfragen zur Implementierung der einzelnen Actions wird hier auf die Struts Online-Dokumentation hingewiesen, die bei Unklarheiten meist schnell weiterhelfen kann ([AJS03]).

Außer Action-Klassen sind im Paket `struts` zusätzlich Komponeten deklariert, die das Standard-Framework erweitern bzw. für die Applikation WebTNDB anpassen. Dabei handelt es sich um die Klassen `WebTNDBRequestProcessor`, `WebTNDBValidator` und `WebTNDBActionConfig`, deren Funktion im folgenden besprochen wird.

WebTNDBRequestProcessor Diese `RequestProcessor`-Subklasse überschreibt die Methode `init( )` des Standard `RequestProcessors`. Dabei handelt es sich um eine Lifecycle-Methode, die automatisch vom `ActionServlet` bei der Initialisierung des Frameworks aufgerufen wird.

Die `WebTNDBRequestProcessor` Implementierung ruft hier eine Klassenmethode der `DataSourceFactory` aus dem Persistence-Framework auf, damit diese die in der Struts-Kunfigurationsdatei vereinbarte Datenbank als Standard-Datenbank einrichten kann. Anschließend ruft sie die geerbte `init( )`-Methode aus der Elternklasse auf, um auch die Standardinitialisierungen vorzunehmen.

Will man mit einer Installation des Web-Interface Zugriff auf mehrere Datenbanken erlauben, so ist es denkbar, mehrere Datenbanken in der Datei `struts-config.xml` zu konfigurieren. Diese könnten dann in der `init( )`-Methode des `WebTNDBRequestProcessors` zusätzlich beim Verzeichnisdienst an-

[13] manchmal auch UI-Frameworks (User Interface) genannt.

gemeldet werden, damit WebTNDB diese via JNDI orten kann ([Lee02]). Eine entsprechende Subklasse `JNDIDataSourceFactory` ist dafür bereits im Persistence Framework von WebTNDB enthalten. Ansonsten ist es nicht nötig hier Veränderungen vorzunehmen.

WebTNDBValidator Standardeingabevalidierungen sind alle deklarativ in der Datei `validaton.xml` vereinbart. Falls bei einer Validierung die Werte mehrerer Felder, die in einer Beziehung zueinander stehen, überprüft werden sollen oder zusätzlich der Systemzustand berücksichtigt werden muß, ist eine rein deklarative Validierung nicht möglich.

Ein Validator ist ein Toolkit, das Klassenmethoden für solche Validierungen bereitstellt. Die Klasse WebTNDBValidator enthält bislang nur eine boolsche Methode, die überprüfen kann, ob die Werte zweier FormBean-Properties übereinstimmen. Diese wird verwendet, um zu überprüfen, ob der Benutzer für das gewünschte Passwort und die Bestätigung desselben den gleichen Wert eingegeben hat.

Zusätzlich könnte hier z.B. eine Methode bereitgestellt werden, die es unterbindet, Anfrageintervalle zu spezifizieren, deren Endzeitpunkt vor deren Startzeitpunkt liegt.

Zur Vorgehensweise bei der Implementierung solcher Custom-Validators und deren Deployment wird hier auf das 12. Kapitel von Ted Husteds Buch „Struts in Action“ verwiesen ([HCW02, S. 365]).

WebTNDBActionConfig Bei der Vorstellung der Controller-Komponenten von Struts wurde bereits dargestellt, wie Actions in der Konfigurationsdatei durch `<action>`-Elemente vereinbart werden, die zur Laufzeit durch Instanzen der Klasse `ActionMapping` repräsentiert werden. Die Definition des `<action>`-Elements sieht dabei auch optionale Eigenschaften vor, die durch Subelemente des Typs `<set-property>` Bestandteil einer Action-Deklaration sein können.

Zur Laufzeit dient dann eine erweiterte Subklasse von `org.apache.struts.config.ActionConfig` zur Repräsentation der Action-Konfiguration inklusive aller optionalen Elemente.
Bei `WebTNDBActionConfig` handelt es sich um eine solche Klasse.

In der Datei `struts-config.xml` gibt das `type`-Attribut des `<action-mappings>`-Elements, das alle `<action>`-Deklarationen umschließt, an, daß diese Klasse verwendet werden soll :

```
<action-mappings
  type="nu.mine.webtndb.struts.WebTNDBActionConfig">
```

`WebTNDBActionConfig` stellt außer den geerbten Attributen zur Repräsentation der üblichen Deployment-Konfiguration Felder namens `loginRequired` und `accessLevel` zur Verfügung. Weiterhin handelt es sich bei `WebTNDBActionConfig` nicht um eine direkte Erweiterung der `ActionConfig` Klasse des Frameworks, sondern bereits um eine Erweiterung der Subklasse

`SecureActionConfig` aus der SSL-Extension, die bereits ein zusätzliches Attribut namens `secure` hat.

Eine Action-Deklartion, die solche optionalen Eigenschaften für eine Action von WebTNDB vereinbart, sieht folgendermaßen aus :

```
<action path="/Delete"
  type="nu.mine.webtndb.struts.actions.DeleteEntryAction"
  validate="false">
    <set-property property="loginRequired" value="true"/>
    <set-property property="accessLevel" value="1"/>
    <set-property property="secure" value="true"/>
    <forward name="notauthorized" path="/Query.do"/>
    <forward name="success" path="/Query.do"/>
    <forward name="error" path=".webtndb.Logon"/>
</action>
```

Obige Deklaration gibt an, daß für die erfolgreiche Ausführung der `DeleteEntryAction`, die zum Löschen eines Datensatzes aus der Datentabelle dient, die Attribute des zugehörigen `ActionConfig`-Objekts folgende Belegung haben müssen : `loginRequired` speichert `true`, `accessLevel` speichert (mindestens) `1` und `secure` speichert `true`.

Gemeint ist damit, daß der Benutzer beim System angemeldet sein muß, bevor für ihn ein „/Delete.do“-Request bedient werden kann. Weiterhin muß er beim System mit einem Benutzerkonto bekannt sein, dem eine Rolle mit mindestens der Autoritätsstufe `1` zugeordnet ist, und die Action darf nur durch einen HTTPS-Request, also via SSL-Verschlüsselung, angefordert werden.

Ein Adminstrator könnte durch Änderung der optionalen `<set-property>`-Elemente in der Datei `struts-config.xml` also eine neue Rechtevergabe oder Sicherheitskonfiguration vornehmen.

Überprüft werden kann diese Konfiguration, indem man beispielsweise die auf /Delete.do endende Request-URL eines dazu autorisierten Benutzers ins Adressfeld eines anderen Browsers kopiert, über den entweder gar kein Benutzer angemeldet ist oder ein Benutzer der nicht über die nötigen Berechtigungen verfügt. Oder man versucht, den gleichen Request unverschlüsselt über HTTP zu senden. Der `RequestProcessor` wird den Benutzer darauf an den im `<forward>`-Element `error` bzw. `notauthorized` angegebenen Request-Pfad weiterleiten.

Überraschenderweise ist im obigen Beispiel für die Delete-Anfrage durch einen nicht dazu berechtigten Anwender der gleiche Folgepfad spezifiziert wie für eine erfolgreiche Ausführung : Beide gelangen über „/Query.do“, nach einer erneuten Auswertung der Anfragefunktion für die eventuell geänderten Datensätze, zurück zur Datentabelle.

Dort wird, falls keine Berechtigung dazu besteht, nichts geändert und stattdessen eine Fehlermeldung angezeigt. Die `execute( )`-Methode der `DeleteEntryAction` greift dazu auf geerbte Methoden der Klasse `WebTNDBAction` zu.

`WebTNDBAction` ist die Superklasse aller Actions von WebTNDB und beinhaltet u.a. Methoden zur Auswertung von Action-Konfigurationen vom Typ `WebTNDBActionConfig` inklusive deren zusätzlicher Attribute, um für optionale Elementdeklarationen das typische Verhalten zu bewirken.

Diese allgemeine `WebTNDBAction`-Klasse stellt auch eine WebTNDB-spezifische Erweiterung des Struts Frameworks dar. Die Implementierung dieser Klasse und die Nutzung ihrer Methoden durch davon abgeleitete Actions wird nun abschließend besprochen.

WebTNDBAction Die Klasse `WebTNDBAction` enthält die Methoden `getDAF( )` und `authorize( )`. Diese sind in allen Erbenklassen und damit in jeder Action verfügbar. `authorize( )` dient zur Autorisierung von Anfragen und `getDAF( )` ermöglicht den Zugriff auf das PersistenceFramework.

Der Methode `authorize( )` müssen für die Autorisierung einer Anfrage nur die Parameter `mapping` und `request` weitergereicht werden, die der `RequestProcessor` jeder `execute( )`-Methode beim Aufruf übergibt. Intern castet die Methode das Mapping zu einer Instanz der zuvor erklärten Klasse `WebTNDBActionConfig`. Da `WebTNDBActionConfig` im `type` Attribut des `<action-mappings>`-Elements angegeben ist, ist sichergestellt, das die `mappings` alle diesen Typ haben. Folglich gelingt der Cast und ergibt eine Referenz vom Typ `WebTNDBActionConfig` anhand der die Methode `authorize( )` die Autorisierung durchführen kann. Stellt ein Benutzer eine Anfrage, zu der er nicht autorisiert ist, kann die `WebTNDBAction` den Request über die `request`-Referenz als ungültig markieren und dazu eine Fehlermeldung speichern. Die Tiles-Definitionen enthalten für alle Views unterhalb des Logos den Custom-Tag `<html:errors>`, an dessen Stelle Struts bei der Ausgabegenerierung solche, mit dem `request` gespeicherte, Fehlermeldungen ausgibt.

Die Methode `getDAF( )` bekommt als Parameter eine numerische ID und gibt eine Referenz auf die zugehörige Fassadenklasse des Persistence Frameworks zurück. Intern wird dabei der `ServiceLocator` des Persistence-`FrameWorks` verwendet. Die Methode `getDAF( )` erspart folglich den `Action`-Implementierungen, selber auf den `ServiceLocator` zuzugreifen und für jeden Datenbankzugriff den gleichen Code zu wiederholen. Darüberhinaus stehen für alle `DAF`-Implementierungen in der Klasse `WebTNDBAction` gleichnamige Konstanten bereit. Es ist folglich nicht nötig, die numerischen IDs der `DAF`s zu kennen. Stattdessen liefert beispielsweise der Aufruf `getDAF(USERS)` eine Referenz auf die Fassadenklasse zum Zugriff auf persistent gespeicherte Benutzerkonten.

Als typisches Beispiel, wie die Methoden `authorize( )` und `getDAF( )` benutzt werden, kann die Klasse `DeleteEntryAction` dienen, deren Konfiguration gerade gezeigt wurde.

```
public class DeleteEntryAction extends WebTNDBAction {

 private void deleteEntry(String entryID)
  throws ServiceLocatorException, DataAccessException {
   EntryDAF entryDAF = (EntryDAF)getDAF(ENTRIES);
   EntryVO entry = entryDAF.findByPrimaryKey(
                     new Long(Long.parseLong(entryID)) );
   entryDAF.delete(entry);
 }

 public ActionForward execute( ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response )
  throws IOException, ServletException {
   String target = "success";
   try {
    if (authorize(mapping, request))
     deleteEntry(request.getParameter("entryid"));
    else
     target="notauthorized";
   } catch (ServiceLocatorException sle) {
    System.err.println(sle);
    sle.printStackTrace( );
    target = "error";
    ActionErrors errors = new ActionErrors( );
    errors.add( ActionErrors.GLOBAL_ERROR,
                new ActionError( "errors.server.internal",
                                 sle.getMessage( ) ) );
                saveErrors(request, errors);
   } catch (DataAccessException dae) {
    System.err.println(dae);
    dae.printStackTrace( );
    target = "error";
    ActionErrors errors = new ActionErrors( );
    errors.add( ActionErrors.GLOBAL_ERROR,
                new ActionError( "errors.database.error",
                dae.getMessage( ) ) );
    saveErrors(request, errors);
   }
   return (mapping.findForward(target));
 }

}
```

Im Quelltext der `DeleteEntryAction` ist auch zu sehen, wie eine Action selber den Request durch das Speichern von `ActionErrors` als ungültig markieren und dazu eine Fehlermeldung aus den Resource-Bundles selektieren kann, falls nach der Autorisierung, z.B. beim Datenbankzugriff, noch Fehler auftreten.

3.4.2 Zusammenfassung

Die Presentation-Tier von WebTNDB besteht aus View Komponenten und Controller Komponeten, die abgegrenzt von den anderen Schichten, dem Model, alle im Paket `struts` gespeichert sind. Es handelt sich dabei um Klassen, die in das Front-End Framework Jakarta Struts eingebettet sind. Jakarta Struts stellt eine MVC-Implementierung nach dem „Model 2 Approach“ dar und dient WebTNDB als Grundlage einer modularen, flexiblen und ansprechenden Präsentationsschicht.

Die View Komponeten sind spezielle JSP-Seiten, sogenannte Tiles, aus denen die einzelnen Bildschirmseiten zusammengesetzt werden. Alle Tiles sind im Verzeichnis `pages/tiles` gespeichert.

Die Controller Komponeten sind sogenannte Actions, die beim Übergang von einer Seite zur nächsten den zugehörigen Request verarbeiten. Die Actions von WebTNDB sind im Subpackage `actions` des Pakets `struts` gespeichert.

Actions und JSP-Definitionen sind Komponenten mit einer einfachen Struktur, die sich schnell nachvollziehen läßt. Da WebTNDB verschiedene Basiskomponenten von Struts erweitert, ist es für die Integration neuer Anwendungsfälle ausreichend, Action-Klassen und JSP-Definitionen hinzuzufügen. Das typische Verhalten bezüglich der Datenzugriffsmechanismen kann, wie auch das typische Erscheinungsbild für View Komponenten, von den Basisdefinitionen abgeleitet werden.

Veränderungen an der Konfiguration des Frameworks ermöglichen sehr umfangreiche Anpassungen. So könnten beispielsweise bei der Initialisierung mehrere verteilte Datenbanken eingerichtet werden oder zusätzliche Sprachen für die Benutzerschnittstelle vereinbart werden. Solche Änderungen setzten jedoch umfassende Kenntnisse der Bestandteile des Frameworks und deren Funktion voraus. In einem Schnelldurchgang wurden in diesem Abschnitt die prinzipiellen Mechanismen zur Anfragebearbeitung durch Struts z.T. durch einfache Beispiele und z.T. direkt anhand einzelner Komponenten von WebTNDB erklärt. Im Quellenverzeichnis ist zusätzliche Literatur aufgelistet, die dabei helfen kann, diese Kenntnisse zu vertiefen.

4 Ausblick

Als Anwendungen einer aktiven temporalen Datenbank sind natürlich die gleichen denkbar, in denen herkömmliche temporale Datenbanken eingesetzt werden. In dieser Arbeit wurde ein Prototyp eines solchen Systems vorgestellt.

Verschiedene mögliche Erweiterungen der Software wurden dabei auch schon angesprochen : Speziell für temporale Daten wurden effiziente Zugriffsmechanismen bereits intensiv untersucht, deren Integration in WebTNDB zur Optimierung dienen könnte.

Als nächstes sollte das System Zugriffe, die Berechnungen erfordern, nicht nur von regulären unterscheiden, sondern in einem solchen Fall auch tatsächlich auf das entsprechende Verfahren, beispielsweise aus der Astronomie, zurückgreifen. Der Algorithmus zur Bestimmung des Datums für das Osterfest ist z.B. in [RD97] dargestellt und könnte als erstes Experiment dazu in den WebCAL Server implementiert werden.

Auch an der Benutzerschnittstelle sind Erweiterungen denkbar. Mehr Komfort würde es bringen, wenn sich alle Meta-Datentabellen, also z.B. auch die Schemata, durch das Web-Interface berarbeiten ließen, so daß dafür kein direkter Zugriff auf die Datentabellen mit einem anderen Client-Programm mehr erforderlich ist. Wenn auch eine Benutzerverwaltung, mit entsprechenden Managementtools für Berechtigungen, in das Web-Interface integriert würde, könnte auf direkte Tabellenzugriffe bei der Administration vollständig verzichtet werden.

Falls bei zukünftigen Anwendungen mit einer Steigerung der Systemlast zu rechnen ist, könnten die Business- und Integration-Tier zu einem EJB-Server portiert werden, der Load-Balancing Verfahren bereitstellt. Dadurch könnte auch das etwas komplizierte Mangagement von Transaktionen vereinfacht werden. Zugleich würden dabei auch Transaktionen über mehrere verteilte Datenbanken ermöglicht. Bei einem EJB-Server handelt es sich aber i.a. um ein kommerzielles System, für das Lizenzgebühren anfallen. Weiterhin müßten Entwickler dazu natürlich die Prinzipien des umfangreichen Enterprise Java Beans Frameworks kennen.

Auch konzeptuelle Erweiterungen des Systems könnten Gegenstand weiterer Forschungsarbeit sein : Insbesondere die Anfragedefinition durch Bäume, deren Einsatzgebiet prinzipiell nicht auf temporale Daten beschränkt ist, führt zu weiteren Fragestellungen.

Interessant wäre es beispielsweise, ob sich die tabellarische Darstellung der Daten durch Einfügen einer zusätzlichen Schicht über der Integration-Tier vollständig vor den höheren Schichten verbergen ließe, so daß dort nur noch mit Bäumen gearbeitet wird.

Anstatt die Bäume gemäß verschiedener Meta-Datentabellen auf Datentabellen abzubilden, könnte diese Zwischenschicht XML-Dokumente zur Repräsentation der Daten verwenden. Die Baumstruktur dieser Dokumente wäre natürlich ideal zur Speicherung von Daten, auf die auch durch Baumstrukturen zugegriffen wird.

Anstelle der Meta-Datentabellen könnten DTDs oder besser noch XML-Schemata treten. Die Baumstruktur der Anfragebäume könnte dann der Dokumentstruktur eines solchen XML-Schemas entsprechen. Solche Dokumenttypdefinitionen unterliegen auch nicht den Einschränkungen, die bisher für Meta-Daten aufgrund der statischen Tabellenschemata berücksichtigt werden müssen.

Falls intern einmal XML-Daten generiert werden, könnten auch Anfragemechanismen, wie sie der XQuery Entwurf des W3C darstellt, brauchbar werden. Zum Zeitpunkt der Erstellung dieser Arbeit sind beim W3C als offizielle Recommandations zu XQuery nur „Working Drafts“ erhältlich. Sobald daraus ein Standard für den Zugriff auf baumartig strukturierte Daten entstanden ist, wäre es natürlich interessant, zu untersuchen, ob sich aus den Anfragefunktionen von WebTNDB auch XQuery-Anfragen generieren ließen.

Die Spezifikation der Anfrageintervalle könnte auch um weitere Features ergänzt werden. Bisher läßt sich nur nach Daten suchen, die innerhalb eines Intervalls liegen. Für viele Anfragen wäre aber auch die Spezifikation mehrerer Intervalle sinnvoll, die in einer Beziehung zueinander stehen. Das könnten z.B. periodische Intervalle sein, die der Anwender durch Begriffe wie „werktags“ oder „am Wochenende“ angibt.

Präzise Definitionen für solche Begriffe zu finden, ist Thema des Artikels [Ohl00], in dem dazu eine Time-Term-Language entworfen wird. Eine solche Sprache könnte auch unscharfe Begriffe zulassen, so daß z.B. nach Konzerten gesucht werden kann, die „kurz vor Sonnenuntergang“ stattfinden.

Die Integration eines Interpreters für eine solche Time-Term Language in ein aktives temporales Datenbanksystem stellt ein weiteres interessantes Thema für zukünftige Forschungen auf diesem Gebiet dar.

A Die Installation von WebTNDB Servern

Um die volle Funktionalität von WebTNDB zu nutzen, müssen mindestens ein Datenbank-Server, ein WebCAL-Server, ein WebCAL-Proxy und ein Applikations-Server bereitstehen. Diese Komponenten können grundsätzlich alle auf dem gleichen oder alle, möglicherweise auch mehrfach, auf verschiedenen Rechnern installiert sein. Welche Verteilung dabei am sinnvollsten ist, hängt von den Datenquellen und der Anwendung ab. Die folgende Installationsanleitung behandelt die Installation der einzelnen Komponenten unabhängig voneinander und stellt dar, wie eine dem Testsystem entsprechende, einsatzfähige Installation durchgeführt werden kann. Weitere Optionen zur Konfiguration eines solchen Systems werden im Zusammenhang mit dem Deployment der Applikation, am Ende des Implementierungskapitels, gezeigt.

Auf den Server-Rechnern werden verschiedene zusätzliche Programme benötigt. Dieser Abschnitt gibt dazu jeweils die vom Testsystem verwendeten Versionsnummern und Internet-Quellen dazu an. Diese Quellen sollten vor einer neuen Installation auf jeden Fall überprüft werden, da möglicherweise bereits neuere Versionen verfügbar und Teile der folgenden Informationen bereits veraltet sind.

A.1 Die Datenbank

Für alle Datenbankinteraktionen greift WebTNDB auf relationale Datenbankmanagementsysteme zurück. Prinzipiell kann dafür jedes System verwendet werden, für das ein JDBC-Treiber[14] verfügbar ist, also z.B. ein kommerzielles Produkt wie Informix, Access oder Oracle. Das Testsystem verwendet die über mysql.com verfügbare Open-Source-Software MySQL. Als JDBC Treiber kann der Open-Source Treiber von GJT[15], den Sun-Microsystems im Internet unter der URL http://industry.java.sun.com/products/jdbc/drivers zum Download bereitstellt, oder der unter http://www.caucho.com/projects/jdbc-mysql verfügbare Treiber der Firma Caucho Technologies, der etwas schneller und für den nicht kommerziellen Einsatz ebenfalls frei ist, verwendet werden.

WebTNDB führt auf der Datenbank Transaktionen durch, die von den Tabellen des Typs MyISAM ([WA02, Section 7]), die MySQL gewöhnlich verwendet, nicht unterstützt werden. Die installierte Version des MySQL Servers muß Taballen des Typs InnoDB[16] kennen. InnoDB-Tabellen wurden von der Firma „Innobase Oy“ entwickelt und erlauben MySQL außer Transaktionen auch die

[14] Die JDBC-API wird auch als „Java Database Connectivity“ bezeichnet und enthält Klassen sowie vor allem Schnittstellen, die Java-Programmen den Zugriff auf tabellarisch strukturierte Daten ermöglichen. Um damit auf die Tabellen einer Datenbank zuzugreifen, benötigt man eine Bibliothek von Klassen, die die in der API beschriebenen Schnittstellen implementieren, einen sogenannten JDBC-Treiber. Weitere Informationen zu JDBC folgen im Abschnitt über das Persistenz-Framework im Implementierungskapitel. Ein ausführliche Einführung findet sich z.B. in [WFCHH99].

[15] **G**iant **J**ava **T**ree vertreibt über das Internet kostenlos Open-Source Java-Software. Aktuelle Entwicklungen können über www.gjt.org eingesehen werden.

[16] Die einzige Alternative für MySQL, sind Tabellen des Typs BDB, wie sie die Version BerkeleyDB unterstützt, die von http://www.sleepycat.com/ heruntergeladen werden kann.

Vergabe von Zeilensperren ([Tuu03]). Die Unterstützung von InnoDB Tabellen ist in MySQL-Server-Versionen $\geq$4 bereits enthalten. Zum Zeitpunkt der Erstellung dieser Arbeit stand MySQL 4 kurz vor der final-Release. Das Testsystem nutzt die Version 4.0.9-γ, wie sie unter http://www.mysql.com/downloads/mysql-4.0.html für viele Rechner-Architekturen und Betriebssysteme binär oder auch im Quelltext heruntergeladen werden kann.

In der Datei `INSTALL_SOURCE`, die in allen MySQL-Distributionen enthalten ist, finden sich detailierte Instruktionen, wie das Betriebssystem zu konfigurieren ist, um das Script `mysql_install_db`, das den Server installiert, ausführen zu können. Nach der Installation des Datenbankservers muß auch dieser noch konfiguriert werden. Dazu müssen in der Datei `my.cnf`, die auch Bestandteil der MySQL-Distribution ist, die Kommentarzeichen vor den mit `innodb_` beginnenden Zeilen entfernt werden und die dort gespeicherten Informationen angepasst werden. Dabei handelt es sich unter anderem um den Ort der Dateien, in denen MySQL InnoDB Tabellen speichert, und deren maximale Größe. Genauere Informationen hierzu finden sich ebenfalls in der Datei `INSTALL_SOURCE`.

Nach der Installation und Konfiguration des MySQL-Servers kann dieser durch Aufruf des Skripts `mysqld_safe` ausgeführt werden.

Zum Quelltext von WebTNDB gehört ein SQL-Skript, das verwendet werden kann, um die vom System benötigten 5 Daten-Tabellen und 11 Meta-Datentabellen auf einem MySQL-Server anzulegen. Für bestimmte adminstrative Aufgaben kann es sinnvoll sein, die interne tabellarische Darstellung der Daten zu kennen und direkt daran Veränderungen vorzunehmen. Die MySQL-Distribution enthält dazu das Client-Programm `mysql`. Im Folgenden wird klar, daß bestimmte Änderungen, die sich so durchführen lassen, zu Datenzuständen führen können, die die Funktionaltät des Systems beeinträchtigen. Mithilfe des Skripts ließe sich dann zwar der ursprüngliche Zustand wieder herstellen, alle zwischenzeitlichen Änderungen gingen dadurch aber verloren.

Die Datentabellen

Der Name einer Datenbank, auf der WebTNDB arbeitet, ist `tndb`, daher ist das erste Kommando, das ein Administrator in `mysql` eingibt

```
use tndb
```

zur Selektion dieser Datenbank. Eine wichtige Tabelle heißt `users` und dient WebTNDB zur Verwaltung der Benutzerkonten. Mit dem Kommando

```
select username, role from users;
```

kann ein Administrator sich dort die Namen und Rollen aller registrierten Benutzer anzeigen lassen. Das Kommando

```
select rolename, accesslevel from roles;
```

zeigt die zu den einzelnen Rollen in der Tabelle `roles` gespeicherten Autoritätsstufen. Die nötigen Schritte, um die für eine Aktion vorrausgesetzte Autoritätsstufe festzulegen, folgen im Deployment-Abschnitt am Ende des Folgekapitels. Eine sinnvolle Operation, könnte ein Administrator in der `users` Tabelle beispielsweise mit dem Kommando

```
update users set role="administrator" where username="xy";
```

durchführen. Die mit der Rolle `"administrator"` assoziierten Berechtigungen würden dem Benutzer `"xy"` zugeteilt.
In der Tabelle `users` trägt das Initialisierungsskript auch ein Benutzerkonto mit dem Namen `default` ein. Die zu diesem Konto gespeicherten Daten über den Funktionsbaum und die Parameterbäume werden kopiert, um einem neu registrierten Benutzer nach seiner ersten Anmeldung eine Anfragefunktion zuzuteilen. Dieses Konto sollte daher nie gelöscht oder umbenannt werden, da WebTNDB sonst keine neuen Benutzer mehr registrieren kann. Mittels
`select password from users where username="default";`
könnte sich ein Administrator aber das Passwort zu diesem Konto beschaffen, um sich damit als default-User anzumelden und die Anfragefunktion, die neue Benutzer anfänglich zugeteilt bekommen, zu bearbeiten.

Die Datentabellen `entries, functions` und `queries` dienen zur persistenten Speicherung der temporalen Einträge und der Funktions- und Parameterbäume der registrierten Benutzer. Diese Tabellen mit `mysql`-Kommandos zu bearbeiten, ist wenig nützlich, da das Web-Interface hierfür weit komfortabler ist und sich alle sinnvollen Änderungen an diesen Tabellen von dort durchführen lassen.

Die Meta-Datentabellen

Die Tabelle `types` speichert mögliche Typen von temporalen Einträgen und hat nach Ausführung des Initialisierungsskripts folgenden Inhalt :

typeid	type	schema	tablecolumns	active	rowversion
1	event	event	1	0	0
2	appointment	appointment	1	0	0
3	phenomenon	phenomenon	1	1	0
4	holiday	holiday	1	1	0

Die `typeid` ist dabei der numerische Primärschlüssel. Ein Typbezeichner könnte folglich mehrfach mit unterschiedlichen Schemata hier gespeichert werden. In der `schema` Spalte ist ein Fremdschlüssel gespeichert und referenziert eine Zeile in der gleichnamigem Meta-Datentabelle. Der Wert in der Spalte tablecolumns gibt an, wieviele zusätzliche Selektoren zur weiteren Kategorisierung eines Typs gespeichert werden. Mögliche Belegungen für solche Selektoren sind in der Tabelle `columnvalues` gespeichert. In den Meta-Datentabellen ist die Möglichkeit vorgesehen, zwei solche Selektoren zu speichern. Das Testsystem berücksichtigt für alle Typen aber nur einen Selektor.

`active` ist ein boolsches Attribut, das in der Datenbank durch eine Spalte vom Typ BIT mit dem Wert 1 für true und dem Wert 0 für false repräsentiert wird. Ist in der `active` Spalte der Wert 1 gespeichert, stellt WebTNDB beim Speichern eines Eintrags dieses Typs keine Möglichkeit zur Verfügung, einen Termin anzugeben, da davon ausgegangen wird, daß ein Algorithmus zur Bestimmung des Zeitpunkts nötig ist. Erweitert man den Zugriffsmechanismus so, daß bei einer entsprechenden Suche alle relevanten Treffer durch den jewei-

ligen Algorithmus automatisch mitberechnet werden, kann auf die persistente Speicherung von Einträgen solcher Typen auch ganz verzichtet werden. Zur Erprobung des Zugriffsmechanismus lassen sich beispielsweise Sonnenfinsternisse im Testsystem noch explizit in die Datentabelle `entries` eintragen. Genauer wird darauf noch bei der Erklärung der `specifiers`-Tabelle eingegangen.

Die Spalte `rowversion` ist in jeder nichtstatischen Tabelle enthalten. Im Folgekapitel wird erklärt, wie ein optimistisches Sperrverfahren mithilfe der dort gepeicherten Werte zeitgleiche Zugriffe mehrerer Benutzer auf die gleichen Datensätze synchronisiert. Für die Administration mit dem `mysql`-Client-Programm ist es nur nötig zu wissen, daß man bei Änderungen die dort gespeicherten Werte beibehalten muß und neue Zeilen immer mit dem `rowversion` Wert `0` einfügt.

Die Tabelle `schema` hat sechs Spalten :

schema-name	firstcolumn-typed	secondcolumn-typed	firstcolumn-name	secondcolumn-name	row-version
appoint-ment	1	0	sort	NULL	0
event	1	0	subject	NULL	0
holiday	0	0	institution	NULL	0
pheno-menon	1	0	identifier	NULL	0

Der in der Spalte `schemaname` gespeicherte Name eines Schemas ist der Primärschlüssel der Tabelle `schema`. Die boolschen Werte `firstcolumntyped` und `secondcolumntyped` geben an, ob für die Selektoren zusätzliche Attribute gespeichert werden. Die Namen dieser Selektoren, die das Web-Interface auf der Editor-Seite vor dem entsprechenden Auswahlfeld anzeigt, sind in den Spalten `firstcolumnname` und `secondcolumnname` gespeichert. Da das Testsystem nur einen solchen Selektor berücksichtigt, ist hier jeweils nur die Spalte `firstcolumntyped` und `firstcolumnname` relevant.

In der Tabelle `columnvalues` sind anfangs folgende mögliche Selektoren gespeichert :

valueid	type	columnnumber	value	rowversion
1	1	1	concert	0
2	1	1	exposition	0
3	1	1	lecture	0
4	2	1	arrangement	0
5	2	1	meeting	0
6	3	1	eclipse	0
7	3	1	sunrise	0
8	3	1	sundown	0
9	4	1	school	0
10	4	1	college	0
11	4	1	university	0
12	4	1	company	0

Der Primärschlüssel in der Spalte `valueid` ist hier wieder der Zeilenindex. In der Spalte `type` steht der Typ der Einträge, die durch den in dieser Zeile gespeicherten Selektor kategorisiert werden können. Angegeben ist der Typ dort jeweils durch einen Fremdschlüssel, der eine Zeile in der bekannten Tabelle `types` referenziert. Für das Testsystem ist der in der Spalte `columnnumber` gepeicherte Selektorindex immer 1. Der Name des Selektors, mit dem er als Knotenlabel und im Elementeditor angezeigt wird, ist in der Spalte `value` gespeichert. In der Tabellendefinition ist diese Spalte als `unique` vereinbart. Jeder Wert kann hier also nur einmal vorkommen und läßt sich folglich zur eindeutigen Referenzierung eines Selektors verwenden.

Zur weiteren Selektion von Einträgen können für die vereinbarten Kategorien zusätzlich Attribute verwendet werden, wenn der entsprechende Selektor in der Tabelle `schema` als typisiert gekennziechnet ist. Mögliche Belegungen für Attribute sind in der Tabelle `specifiers` gepeichert :

specifierid	specifier	specifieditem	exclusive	rowversion
1	business	arrangement	1	0
2	private	arrangement	1	0
3	binding	meeting	1	0
4	voluntary	meeting	1	0
5	official	exposition	1	0
6	regular	exposition	1	0
7	classic	concert	1	0
8	modern	concert	1	0
9	diverting	lecture	1	0
10	scientific	lecture	1	0
11	total	eclipse	0	0
12	partial	eclipse	0	0
13	lunar	eclipse	1	0
14	solar	eclipse	1	0
15	started	sunrise	1	0
16	completed	sunrise	1	0
17	started	sundown	1	0
18	completed	sundown	1	0

Spezifizierer bilden die Knoten der Parameterbäume. Auch hier ist der in der Spalte `specfierid` gespeicherte Zeilenindex der Primärschlüssel. Einen weiteren Schlüssel bilden die Spalten `specifier` und `specifieditem`, welche jedoch einzeln keine Schlüssel sind. Der Wert *started* kann zum Beispiel z.B. verwendet werden, um den Selektor *sunrise* oder *sundown* zu spezifizieren. *sunrise* und *sundown* können aber beide auch durch *completed* spezifiziert werden. Die Spalte `exclusive` speichert einen boolschen Wert, der aussagt, ob ein Spezifizierer nur allein zur Spezifizierung eines Selektors verwendet werden darf. Das ist der Fall, wenn der `exclusive` BIT Wert für alle Zeilen zur Spezifikation eines Selektors auf 1 gesetzt ist. Ein *event* Eintrag der Kategorie *meeting* kann z.B. nicht durch die Spezifizierer *binding* und *voluntary* zugleich spezifiziert werden. Sind in der Tabelle für einen Selektor zusätzlich Spezifizierer, deren `exclusive` BIT auf 0 gesetzt ist, gepeichert, muß zu dessen Spezifizierung ein zweites Attribut gespeichert werden. Diese Spezifizierer werden dann als mögliche Belegungen des zweiten Attributs verwendet und im Parameterbaum als Kindknoten an das erste Attribut angehängt. Spezifizierer mit dem `exclusive` Wert 1 sind immer Belegungen für das erste Attribut. Ausschließlich Spezifizierer mit dem `exclusive` Wert 0 zu vereinbaren, ist bei diesem Selektionsmechanismus nicht sinnvoll.

Das Testsystem verwendet nur bei *phenomenon* Einträgen der Kategorie *eclipse* ein zweites Attribut. Außer dem Spezifierer *lunar* oder *solar* können dafür im Parameterbaum zusätzlich die Spezifizierer *partial* und *total* zur Selektion verwendet werden. Bei der Speicherung oder Bearbeitung eines solchen Eintrag müssen die Belegungen für beide Attribute ausgewählt werden.

Die Typen dieser Einträge sind in der `types`-Tabelle als „active“ gekennzeichnet. Zur Bestimmung der tatsächlichen Termine hierfür sollen also einmal zusätzliche Algorithmen aufgerufen werden. Das gleiche gilt für Einträge vom Typ *holiday* zur Speicherung von Feiertagen, deren Selektoren durch kein weiteres Attribut gekennzeichnet sind. Die Verarbeitung solcher Einträge ist, wie ihre Selektion, durch Parameterbäume mit weniger oder mehr als einem Kindknoten, bislang nur experimentell implementiert : WebTNDB legt solche Termine immer mit dem Startzeitpunkt 30.04.2003, 9 Uhr mitteleuropäischer Zeit bis zum Endzeitpunkt um 10 Uhr des gleichen Tages in der Datenbank ab. Die Selektion solcher Einträge mit einem Selektor und bis zu zwei Parametern, die einmal zur Auswahl des entsprechenden Algorithmus dienen soll, ist bereits voll funktionsfähig und kann, wenn dieser Zeitraum in das Anfrageintervall aufgenommen wird, auf dem Testsystem erprobt werden.

Für die Unterstützung eines in den Meta-Datentabellen bereits vorgesehenen zweiten Selektors sowie auch für die Integration astronomischer Algorithmen, wie sie zukünftige Versionen von WebCAL anbieten werden, sind Erweiterungen an der im Folgekapitel detailiert beschriebenen Implementierung notwendig.

Die möglichen Knoten der Parameterbäume mit der Wurzel *season* und *location* sind in eigenen Metadatentabellen gespeichert, da zu ihrer Auswertung ein anderer Zugriffsmechanismus verwendet wird.

`season` ist eine statische Tabelle, d.h. Änderungen sind hier nicht vorgesehen. In der Tabelle ist zu jeder der vier Jahreszeiten eine numerische ID gespeichert. Für welchen Terminzeitraum einem Eintrag die ID welcher Jahreszeit zugewiesen wird, ist programmatisch festgelegt und auf die Nordhalbkugel der Erde bezogen.

Parameterbäume mit dem Wurzelknoten *location* können eine Tiefe von bis zu fünf Knoten haben. Die möglichen Belegungen für die Kindknoten müssen dabei in den Tabellen `continents`, `countries`, `states` oder `cities` gespeichert sein. Bis auf die statische Tabelle `continents` haben diese Tabellen alle das gleiche Spaltenschema : Die erste Spalte `countryid`, `stateid` und `cityid` ist der Primärschlüssel, ein numerischer Zeilenindex. In der zweiten Spalte `countryname`, `statename` bzw. `cityname` ist der Name des Eintrags gepeichert. Die dritte Spalte `countrycode`, `statecode` bzw. `citycode` speichert eine Abkürzung des Namens der verwendet wird, um Pfade und Knotenlabel mit möglichst wenig Zeichen darzustellen. Dieser Code ist als `unique` vereinbart, muß also im Gegensatz zum Namen ein Schlüssel sein. Will man z.B. die Städte *Frankfurt am Main* und *Frankfurt an der Oder* in der `cities` Tabelle speichern, kann für beide der Name *frankfurt* gespeichert werden, als Code müssten aber unterschiedliche Zeichenketten wie z.B. *frm* und *fro* verwendet werden. In der vierten Spalte ist ein Fremdschlüssel, der die übergeordnete Region in der entsprechenden Tabelle angibt, gespeichert. Für `cities` also eine

Referenz auf einen Eintrag in der Tabelle `states`, für `states` eine Referenz auf einen Eintrag in der Tabelle `countries` und für `countries` eine Referenz auf einen Eintrag in der Tabelle `continents`, die als einzige *location*-Tabelle selber keine solchen Referenzen speichert.

Die Meta-Datentabelle `counter` dient WebTNDB für interne Mechanismen. Hier sind für alle nichtstatischen Tabellen bis zu zwei Sequenznummern gespeichert. Bei der einen handelt es sich um einen Zähler, der inkrementiert wird, um den Primärschlüssel für einen neuen Eintrag in einer Tabelle zu erzeugen, sofern die Tabelle einen numerischen Zeilenindex als Primärschlüssel verwendet. Die zweite Nummer ist auch ein Zähler, der zur Versionierung von Zeilen verwendet wird. Versionsnummern in der `rowversion` Spalte einer Tabelle sind für die von WebTNDB verwendete optimistische Locking-Strategie von Bedeutung. Die Counter-Tabelle wird von WebTNDB automatisch aktualisiert. Ein Admistrator, der mit dem `mysql`-Client Programm direkt auf den Tabellen arbeitet, muß beachten, daß der Wert des Indexzählers immer mindestens so hoch sein muß wie die Anzahl der in der entspechenden Tabelle gespeicherten Zeilen. Beim Einfügen neuer Zeilen, z.B. in eine der Meta-Datentabellen, mit dem `insert`-Kommando muß der zugehörige Zähler also gegebenenfalls durch ein `update`-Kommando angepasst werden.

A.2 Der WebCAL Server

Der WebCAL Server ist von allen Komponeten am einfachsten zu betreiben. Dazu muß nur das Programm `server` von der Kommandozeile gestartet werden.

Dabei handelt es sich jedoch um nativen Binärcode, der von den C++-Quellen speziell für die Rechnerarchitektur des Server-Computers kompiliert sein muß. Um den WebCAL-Server von den C++-Quellen für eine neue Architektur zu übersetzen, müssen dort bestimmte Bibliotheken verfügbar sein. Da der Wertebereich des in C für Zeitkoordinaten üblichen Datentyps `time_t` nur von 0 bis 2^{31} reicht, verwendet WebCAL das GNU-Multiple-Precision-Arithmetics Package. 2^{31} Sekunden sind seit dem 1.1.1970, wo sich der Ursprung der Referenzzeitline befindet, nämlich bereits am 19.1.2038 abgelaufen. Weiterhin wird die Existenz bestimmter I/O-Stream-Libraries, wie z.B. `sstream`, die in `gcc` Versionen $\leq$2.95 nicht enthalten ist, vorrausgesetzt. Bisher ist nur für Intel-Architekturen eine ausführbare Version verfügbar. Daher wird ein solches System zum Betrieb des WebCAL-Servers empfohlen.

A.3 Der Proxy-Server

Ein Proxy-Server ist ein Vermittler, der Client-Anfragen an einen weiteren Server, in diesem Fall den WebCAL-Server, weiterleitet und dessen Antworten an den Client zurückschickt. Für den Client sieht die Kommunikation mit dem Proxy-Server dabei so aus, als würde er mit dem WebCAL-Server selber kommunizieren. Der WebTNDB Proxy-Server stellt dem Client jedoch verschiedene zusätzliche Kommandos zur Verfügung, um auch auf Datenbanken zugreifen zu können.

Der Client bekommt durch den ProxyServer direkten Zugriff auf die im WebCAL Server verwalteten Intervalle. Weiterhin ist es mit dem Proxy Server möglich, auf temporale Daten zuzugreifen, die über mehrere Datenbanken auf unterschiedlichen Server-Rechnern verteilt sind.

Für den Betrieb des WebTNDB Proxy-Servers benötigt man eine Java-Virtual-Machine ≥ 1.3 und Zugriff auf einen sogenannten Verzeichnisdienst, für den ein JNDI-Treiber verfügbar ist. Ein Verzeichnisdienst stellt einen Name-Service bereit, bei dem zu einem Namen Referenzen eingetragen und wieder angefragt werden können. Ein solcher Service funktioniert also ähnlich wie ein Adressbuch. Darüber hinaus stehen für bestimmte Dienste meist Suchoptionen mit zusätzlichen Attributen bereit, so daß z.B. nach allen Druckern, die in Farbe drucken können, innerhalb einer Abteilung gesucht werden kann. Von solchen Features wird hier kein Gebrauch gemacht. Oft verwendete Verzeichnisdienste sind z.B. DNS, LDAP, Cos-Naming oder die RMI-Registry, für die JNDI-Treiber inklusive der zugehörigen Interface-Definitionen von
http://java.sun.com/products/jndi heruntergeladen werden können.

JNDI steht für **J**ava **N**aming and **D**irectory **I**nterface und vereinbart u.a. die Schnittstelle `javax.naming.`*`Context`*, über die Java Programme auf Verzeichnisdienste zugreifen können, nachdem ein JNDI-Treiber geladen wurde. Der Treiber ist dabei ein Paket von Klassen, die die in den JNDI-Schnittstellen vereinbarten Methoden für den Zugriff auf einen speziellen Verzeichnisdienst implementieren müssen. Für den Zugriff auf Verzeichnisdienste ist JNDI also das, was JDBC für den Zugriff auf Datenbanken darstellt.

Um sich auch beim Verzeichnisdienst nicht an ein spezielles Produkt zu binden, verwendet WebTNDB JNDI um Datenbanken zu lokalisieren. Die Methode *`bind( )`* der Schnittstelle *`Context`* dient in JNDI dazu, einen neuen Eintrag hinzuzufügen, und mit der Methode *`lookup( )`* kann nach Einträgen gesucht werden. Die weiteren Möglichkeiten von JNDI werden in [Lee02] vorgeführt. Für das Verständnis der Quellen des Proxy-Servers ist es jedoch nicht nötig, mehr von JNDI zu kennen als die hier erklärten Grundlagen.

Das dort registrierte Objekt, mit dem WebTNDB arbeitet, ist eine Instanz vom Typ `javax.sql.DataSource`. `DataSources` sind Factories ([GHJ97, S. 87]) für Datenbankverbindungen. Die Klasse `javax.sql.DataSource` ist Bestandteil des JDBC 2.0 Optional-Package (früher JDBC Standard Extension), die wiederum Bestandteil von J2EE ist und von http://java.sun.com/products/jdbc heruntergeladen werden kann[17]. `DataSources` sind inzwischen der bevorzugte Weg, Datenbankverbindungen aufzubauen. Von der Verwendung der JDBC Klasse `DriverManager`, die in der Einführungs-Literatur oft noch vorgeführt wird (z.B. [Paw00, S. 71]), wird inzwischen i.a. abgeraten, da hier zum einen der verwendete JDBC-Treiber hartkodiert wird und weiterhin *`Connection`* Typen wie `XAConnections` und `PooledConnections`, die auch verteilte Transaktionen erlauben und effiziente Zugriffsstrategien nutzen, nicht unterstützt werden ([WFCHH99. S. 467]).

[17] ab J2SE Version 1.4 ist das JDBC 2.0 Optional-Package auch dort enthalten.

Kommerzielle Applikationsserver wie WebSphere von IBM oder WebLogic von BEA[18] bieten meist eine grafische Benutzerschnittstelle wie z.B. Die „WebLogic Server Console“ ([GWE02, S. 607]), um DataSources unabhängig vom Quelltext für JNDI konfigurieren zu können.

Für die Konfiguration einer `DataSource` müssen natürlich, genau wie beim `DriverManager` auch, der Datenbankrechner, der Name der Datenbank, der Port, ein Beutzername und dessen Passwort bekannt sein. Folgendes Codefragment aus dem Proxy-Server zeigt, wie die Datenbank des Testsystems als DataSource konfiguriert und beim Verzeichnisdienst unter dem Namen „`tndb`“ registriert wird :

```
DataSource ds = new MysqlConnectionPoolDataSource( );
ds.setUrl("jdbc:mysql://webtndb.mine.nu:3306/tndb");
ds.setDatabaseName("tndb");
ds.setPort(3306);
ds.setServerName("webtndb.mine.nu");
ds.setUser("username");
ds.setPassword("password");
this.dataSource = ds;
env = System.getProperties( );
env.put( Context.INITIAL_CONTEXT_FACTORY,
         "com.sun.jndi.rmi.registry.RegistryContextFactory" );
try {
   ctx = new InitialContext(env);
   ctx.bind("tndb", ds);
} catch (javax.naming.NameAlreadyBoundException nabe) {
} catch (Exception e) {
   throw new DataAccessException(
        "error binding database in jndi registry : " + e );
}
```

Kommandos, die mit dem #-Zeichen beginnen, gibt der Proxy-Server nicht an den WebCAL-Server weiter sondern versucht sie als Datenbank-Kommandos zu interpretieren. Das Kommando `#tndb`, das mit einem Datenbanknamen parametrisiert wird, dient zur Auswahl einer Datenbank. Der Proxy-Server ruft dafür seine `tndb`-Methode auf, die wie folgt aussieht :

[18] ein solches grafisches Application Deployment Tool ist auch Bestandteil der kostenlosen J2EE-Referenzimplementierung von Sun.

```
private void tndb(String dbid)
 throws DataAccessException {
   try {
      this.dataSource = (DataSource)ctx.lookup(dbid);
   } catch (Exception e) {
      throw new DataAccessException(
           "error retrieving datasource "
           + dbid +" via jndi : " + e );
   }
}
```

Innerhalb des `try`-Blocks ist zu sehen, wie der Proxy-Server die Datenquelle beim Verzeichnisdienst anfordert.

Weitere Datenbank-Kommandos, die der Proxy-Server interpretiert, sind `#user`, `#details`, `#url`, `#start`, `#end`, `#get` und `#help`.

`#help` listet genau diese Kommandos auf. Mit `#get` wird eine Anfrage abgesetzt. Die Bäume für die Beschreibung der Anfragefunktion sind über das Web-Interface zu spezifizieren. Eine spezielle Anfragefunktion kann beim Proxy-Server mit dem Kommando `#user username` für einen bekannten Benutzer ausgewählt werden. Als Antwort gibt der Proxy-Server die Titel der gefundenen Einträge und für jeden Eintrag dessen numerische ID zurück. Diese ID wird als Parameter für die Kommandos `#details`, `#url`, `#start` und `#end` verwendet, um die mit einem Eintrag gespeicherte Beschreibung, die URL oder den Termin von Start- oder Endzeitpunkt aus der Datenbank zu lesen.

Für Termine wird dabei jeweils ein WebCAL-Interval-Identifier zurückgegeben. Dieser kann mit den von WebCAL bereitgestellten Kommandos weiterverarbeitet werden ([Ohl02, S. 11]).

Da das Testsystem die RMI-Registry[19] als Verzeichnisdient nutzt, muß dieses Programm vor dem Proxy-Server gestartet werden. Dazu gibt man das Kommando

`rmiregistry &`

auf der Kommandozeile ein und startet den Proxy-Server anschließend selbst durch das Kommando

`java WebTNDBServer <host> <port>`

Die Parameter `host` und `port` geben dabei Rechnernamen und Portnummer des Serverrechners an, auf dem der WebCAL Server läuft.

A.4 Der Applikations-Server

Die Web-Applikation basiert auf sogenannten Java-Web-Komponenten, genauer Java-Servlets und JSPs. Zu deren Bereitstellung ist ein sogenannter Servlet Container notwendig. Das Testsystem verwendet hierzu den Open-Source Container Tomcat 4.1.18 von der Jakarta Organization. Um Kontrollfluß und Inter-

[19] Das Programm `rmiregistry` gehört zu jeder J2SE Distribution und wird i.a. zur Lokalisierung von verteilten Objekten, die Java **R**emote **M**ethod **I**nvocation für den entfernten Methodenaufruf nutzen, verwendet.

aktionen zwischen den Web-Komponten zu steuern, setzt WebTNDB außerdem das Front-End Framework Jakarta Struts ein.

Die Installationsanleitung geht hier knapp auf Komponenten-Frameworks und Applikations-Server ein, um die im folgenden verwendeten Begriffe vorab zu erklären. Weitere Ausführungen über Jakarta Struts folgen im Abschnitt über die Präsentationsschicht im Implementierungsteil, bevor diese Technologie verwendet wird.

Serverseitige Anwendungen mit Datenbanknutzung haben oft eine sehr komplexe mehrschichtige Architektur. Es sollen alle Komponenten auf Erweiterbarkeit und Anpassungen, speziell eine zunehmenden Verteilung auf immer mehr Server-Rechner bei steigender Systemlast, ausgelegt werden.

Das für die J2EE Plattform definierte Komponentenmodell Enterprise JavaBeans, kurz EJB, beschreibt verschiedene Schnittstellen und Basisklassen serverseitiger Komponenten. Unter einem J2EE-Application-Server oder EJB-Server versteht man i.a. Software, die auch einen EJB-Container umfasst. Dieser stellt die nötigen Infrastrukturdienste für den Betrieb von EJB-Komponenten auf mehreren verteilten Server-Rechnern bereit. Ein großer Vorteil eines solchen Systems ist, daß der Programmierer von geschickten Lastenausgleichsverfahren profitieren kann, die für seine Komponenten die höchst mögliche Verfügbarkeit gewährleisten, ohne daß er diese komplizierten Verfahren selber kennen muß. Außerdem lassen sich Transaktionsgrenzen bei Datenbankzugriffen schichtübergreifend außerhalb des Quelltextes in sogenannten Application-Deployment-Deskriptoren festlegen.

EJB-Server sind meist kommerzielle Produkte wie z.B. BEA WebLogic, IBM WebSphere oder Sun One, für deren Betrieb die Hersteller hohe Lizenzgebühren verlangen. Die Verwendung der kostenlosen Referenzimplementierung von Sun hat erhebliche Einschränkungen zur Folge. Da die zu erwartende Serverlast, die WebTNDB zu tragen hat, vorerst nicht übermäßig hoch sein wird und sich auch die Komplexität des Systems dadurch verringern ließ, wurde auf den Einsatz eines EJB-Servers verzichtet.

Mit Applikations-Server ist hier also die Standalone-Anwendung Jakarta Tomcat gemeint, die einen Container ausschließlich für Web-Komponenten beinhaltet.

Die Wahl dieser Software hat verschiedene weitere Design-Entscheidungen bedingt. So mußten Standard-Entwurfsmuster (J2EE-Patterns) z.T. abweichend von den offiziellen Dokumentationen implementiert werden, da diese von der Verwendung eines EJB-Containers ausgehen. Insbesondere bei der Handhabung von Datenbanktransaktionen wurde ein eigener Lösungsansatz gewählt. Die Komponenten von WebTNDB sind jedoch so strukturiert, daß die Portierung zu einem EJB-Server mit nicht allzu großem Aufwand machbar sein sollte.

Tomcat kann von http://jakarta.apache.org/tomcat heruntergeladen werden. Alle Binaries sind Java-Klassen und daher für jede Plattform gleich. Ein Download und eine Kompilierung der Quellentexte ist auch möglich aber nicht nötig. Zusätzlich müssen noch die von Jakarta Struts benötigten Pakete von http://jakarta.apache.org/struts heruntergeladen und installiert werden. WebTNDB wurde mit der Struts Version-1.1-β3 entwickelt und getestet. Frü-

here Versionen sind auch noch im Umlauf, können aber nicht verwendet werden. Ist also eine bereits ältere Version von Struts 1.1 oder sogar noch Struts 1.0 installiert, muß diese aktuallisiert werden. Sobald die 1.1 final-Release verfügbar ist, sollte diese auch problemlos verwendet werden können. Für die Installation müssen einige Klassenbibliotheken in die dafür vorgesehenen Verzeichnisse kopiert werden. Im Fall von Struts sind das alle Archive mit der Dateiendung `jar`, sowie die JSP-Tag-Library-Deskriptoren mit der Dateiendung `tld`, die aus dem `lib`-Verzeichnis der Struts-Distribution in den WebSpace unserer Applikation kopiert werden müssen.

Der WebSpace ist ein Verzeichnis, aus dem Tomcat auch die übrigen Komponenten einer Applikation lädt. Die kompilierten Java-Klassen von WebTNDB inklusive zusätzlicher Bibliotheken, wie z.B. Treiber für JDBC und JNDI, müssen also auch alle in dieses Verzeichnis. Außerdem die Konfigurationsdateien `web.xml` und `struts-config.xml`, auf deren Inhalt im Folgekapitel eingegagen wird. Der Ort dieses Verzeichnisses wird im `path`-Attribut eines `Context`-Elements, das in der Tomcat-Konfigurations-Datei `server.xml` definiert wird, gespeichert. Einfacher hat man es, wenn man ein neues Unterverzeichnis des Verzeichnisses `webapps` verwendet, das bereits Bestandteil der
Tomcat-Distribution ist und verschiedene Beispielapplikationen enthält. In diesem Fall muß an der Konfiguration in der Datei `server.xml` nichts verändert werden und alles wird automatisch geladen.

Die genaue Directory-Struktur für den WebSpace, die Einsteigern oft etwas kompliziert erscheint, ist online auf der Seite
http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/deployment.html dokumentiert, und z.B. auch in der Tabelle 1.2 in [Goo02, S. 6] dargestellt. Ist alles einmal wie dort beschrieben eingerichtet, läßt sich Tomcat mit dem Skript `startup.sh` starten und die Applikation ist über Port 8080 verfügbar.

Soll dabei wie im Testsystem HTTPS unterstützt werden, muß zusätzlich SSL konfiguriert werden. Als erstes müssen dafür ein paar Ergänzungen für den Container eingerichtet werden. U.a. werden die **J**ava **S**ecure **S**ockete **E**xtension JSSE und ein sogenannter keystore, der sich mit dem Programm `keytool` erzeugen läßt und zur Verwahrung der Schlüssel verwendet wird, benötigt. Eine detailierte Auflistung der einzelnen Schritte zum Containter-Setup für SSL kann auf der Seite http://jakarta.apache.org/tomcat/tomcat-4.1-doc/ssl-howto.html nachgelesen werden.

Außerdem wird die SSL-Extension `sslext` von Steve Ditlinger benötigt, die von http://sourceforge.net/projects/sslext heruntergeladen werden kann und ebenfalls in den WebTNDB WebSpace kopiert werden muß. Eine Erklärung der dort enthaltenen Klassen kann in [Dit02] nachgelesen werden. Leider müssen für die Zusammenarbeit mit der Struts Version ab 1.1-β3, wie WebTNDB sie benötigt, dort auch noch verschiedene Anpassungen vorgenommen werden : Das Paket `sslext` muß entpackt und in allen Quellen die Klasse `ApplicationConfig` in `ModuleConfig` umbenannt werden. Die Konstante
`action.Action.APPLICATION_KEY` muß durch die Konstante
`Globals.MODULE_KEY` ersetzt werden. Die `extends`-Vereinbarung der Klasse `SecureRequestProcessor` muß angepasst werden, so daß sie nicht von der Klas-

se `RequestProcessor` sondern von `TilesRequestProcessor` abgeleitet wird[20]. Die SSL-Extension wird anschließend rekompiliert und das entsprechende Plugin muß konfiguriert werden. Dazu wird abschließend folgender Eintrag in die Datei `struts-config.xml` geschrieben :

```
<plug-in className="org.apache.struts.action.SecurePlugIn">
       <set-property property="httpPort" value="8080"/>
       <set-property property="httpsPort" value="8443"/>
  </plug-in>
```

[20]In einer kurzen Email-Korrespondenz mit Steve Ditlinger konnte ich klären, daß er plant, diese Anpassungen in der nächsten Version von `sslext` auch vorzunehmen. Weiterhin wird es dort wahlweise zwei Varianten der Klasse `SecureRequestProcessor` geben : Eine, die von `RequestProcessor` erbt und eine für den Betrieb mit Tiles, die von `TilesRequestProcessor` erbt. Diese `sslext` Version wird vorraussichtlich auch Bestandteil der final Release von Struts 1.1 sein, so daß überhaupt keine Änderungen am Quelltext mehr nötig sind, sobald diese verfügbar ist.

B Jakarta Struts

Struts, heißt zu deutsch „Verstrebungen“ oder „Stützbalken“ und bezeichnet das bekannteste und am meisten verbereitete Front-End Framework für Java Web-Applikationen. Struts ist ein Open-Source Projekt, das 2000 von Craig R. McClanahan, Ted Husted und anderen Programmierern gestartet wurde, und umfaßt inzwischen 8 Top-Level Packages und ungefähr 250 Klassen und Schnittstellen, die frei von Lizenzgebühren von
http://jakarta.apache.org/struts heruntergeladen werden können.

Neben Struts gibt es verschiedene andere UI-Frameworks wie z.B. Expresso, Barracuda, Cocoon, Velocity, Turbine etc. die alle ihre eigenen Vorzüge haben.

Struts ist mit durchschnittlich mehr als 1000 neuen Downloads täglich das führende Produkt seiner Art. Die Gründe dafür sind, daß Struts als das ausgereifteste UI-Framework gilt, und, daß ein für die Zunkunft angekündigter Standard, wie ihn Sun mit den „**J**ava **S**erver **F**aces“ (JSF) verspricht, auf Struts basieren wird[21], so daß Struts-erfahrene Programmierer sich dort schnell zurecht finden werden. In einem Artikel auf dem in Expertenkreisen vielbeachtetem Online-Forum http://theserverside.com wurde Struts vor kurzem bereits als „de facto standard for java web applications“ bezeichnet.

Da bereits zu Beginn der Entwicklung von WebTNDB klar wurde, daß das Web-Interface ein komplexes Session- und View-Management erfordert, fiel die Entscheidung zur Verwendung von Struts bereits sehr früh. Ohne ein solches Framework hätte für die Präsentationsschicht eine eigenes
MVC-Komponentenmodell programmiert werden müssen. Der dadurch verursachte Aufwand wäre deutlich höher ausgefallen als der für die Einarbeitung in das standardisierte Framework Struts. Zudem lohnt sich die in Struts investierte Zeit auch für zukünftige Projekte.

Die ausführliche, frei verfügbare Online-Dokumentation ([AJS03]) sowie zahlreiche sehr aktive Newsgroups und Foren im Internet können den Einstieg in die Technologie erleichtern. Auch im Buchhandel sind inzwischen sieben Bücher über Struts erhältlich, von denen „Struts In Action“, das von den Hauptautoren der Software selbst geschrieben wurde, sicher am ehesten empfehlenswert ist ([HCW02]).

Solche Literatur ist hilfreich, wenn man selbst Struts-Applikationen schreiben möchte. Für das Verständnis der Presentation-Tier von WebTNDB ist sie aber nicht notwendig. Dieser Anhang enthält eine Einführung in Struts, in der die Grundlagen für die Arbeit mit dem Framework anhand einfacher Beispiele erklärt werden. Weiterführende Konzepte von Struts werden direkt anhand von Komponenten der Presentation-Tier von WebTNDB vorgeführt. Danach ist der Leser in der Lage, die Arbeitsweise der übrigen Komponenten durch Einsicht der Quelltexte zu verstehen. Weiterhin werden in diesem Anhang einige der wichtigsten Features von Struts wie z.B. Internationalization-Support, Custom-Tag-Libraries etc., von denen auch WebTNDB profitiert, besprochen.

[21] Craig R. McClanahan ist der Kopf der Gruppe, die derzeit bei Sun die JSF-Spezifikation ausarbeitet.

Den Code einzelner Struts-Komponenten zu verstehen, ist relativ einfach, da diese meist kleine, klar definierte Aufgaben erledigen. Um solche Komponenten in des Framework einzubetten, ist aber weiterhin nötig, dessen komplexe Architektur vollständig zu verstehen.

Die folgende Einführung beinhaltet außer Code-Beispielen daher auch eine ausführliche Darstellung der Mechanismen die Innerhalb des Frameworks ablaufen. Dabei wird insbesondere auf die Bedeutung der einzelnen Konfigurations-Elemente detailiert eingegangen. Um die Einführung dennoch möglichst knapp zu halten, kommt sie nicht ohne Fachterminologie aus, mit der man sich möglicherweise erst vertraut machen muß. Um eine konkrete Vorstellung von den Vorgängen bei der Anfragebearbeitung zu entwickeln, empfiehlt es sich daher, die folgenden Beschreibungen gründlich durchzusehen und anhand der Beispiele nachzuvollziehen.

B.1 Struts Komponenten

Die Model Komponenten einer Struts-Applikation können beliebige Java-Klassen, EJBs oder Komponenten anderer Frameworks sein. Im Fall von WebTNDB handelt es sich dabei um die zuvor besprochenen Klassen der Bussiness- und Integration-Tier, die sich in keiner Weise von Java-Applikationen mit anderen Benutzerschnittstellen unterscheiden.

Struts beschreibt die View- und Controller-Komponenten einer Applikation.

Java Server Pages : Die Views der Struts Applikation Die View Komponenten einer Struts Applikation sind **J**ava **S**erver **P**ages (JSPs). JSPs sind eine Ergänzung zur Java-Servlet Technologie, die es erlauben HTML-Tags (oder XML) und Java-Code im Quelltext einer Seite zu mischen. Eine solche Seite wird mit der Dateiendung `jsp` im Server-Webspace gespeichert wie eine normale HTML-Seite. Beim ersten Aufruf dieser Seite, übergibt der Server diese an eine sogenannte JSP-Engine, wie sie z.B. in Tomcat integriert ist. Die Engine übersetzt die JSP und erzeugt daraus ein Servlet, das alle zukünftigen Client-Anfragen an die JSP bedient, indem es die Java-Logik aus den JSP-Tags ausführt und den zugehörigen HTML-Code ausgibt ([Pek00, S. 15]). Eine Beispiel JSP-Seite könnte z.B. wie folgt aussehen :

```
<HTML> <BODY>
<B>Hello Client !</B><BR/>
   I am a Web-Server and my name is
<%= request.getServerName( ) %><BR/>
<%! java.util.Date now = new java.util.Date( );
    String currentTime =
    new java.text.SimpleDateFormat("HH:mm:ss").format(now);
%>
   The current time is <%= currentTime %>
</BODY> </HTML>
```

Auf der Client-Seite sieht die erzeugte Ausgabe im Browserfenster z.B. so aus

```
Hello Client !
I am a Web-Server and my name is www.webtndb.mine.nu
The current time is 18:25:06
```

Die Ansicht des Seitenquelltextes im Browser läßt nicht mehr erkennen, daß die Seite dynamisch erzeugt wurde. Spürbar wird das jedoch nach einem erneuten Laden der Seite, bei dem die ausgegebene Uhrzeit jedesmal angepasst wird. Außerdem fällt dabei auf, daß die Seite deutlich schneller geladen wird als beim ersten Mal, da das erzeugende Servlet nicht neu generiert werden muß.

Auf der Beispielseite kann man sehen, wie sich Java-Ausdrücke in sogenannten JSP-Expressions, die durch `<%= %>`-Tags markiert werden, serverseitig auswerten lassen, um deren Ergebnisse in die HTML-Ausgabe zu übernehmen ([Pek00, S. 27]). Unabhängig von der Ausgabe lassen sich Blöcke von Java-Code innerhalb sogenannter JSP-Scriptlets, die durch `<%! %>`-Tags markiert werden, ausführen, wie es im Beispiel zur Ermittlung und Speicherung der aktuellen Uhrzeit getan wird. Innerhalb von Scriptlets und JSP-Expressions kann auf bestimmte, implizite Objekte zugegriffen werden, die nicht zuvor deklariert werden müssen ([Pek00, S. 30]). Im Beispiel handelt es sich bei der Variable `request` um ein solches implizites Objekt.

In JSPs stehen weiterhin verschiedene Tags zur Verfügung, die mit dem Präfix `jsp` beginnen ([Pek00, S. 263]). Besonders erwähnenswert ist der Tag `<jsp:useBean>`, der dazu dient, in JSPs auf JavaBeans zuzugreifen.

Um Skriptletcode in JSPs weitgehend zu vermeiden, da dies die Gefahr birgt Business-Logik in der Präsentationsschicht unterzubringen, lassen sich die verfügbaren JSP-Tags durch sogennante Custom-Tags erweitern ([Pek00, S. 256]). Struts stellt 6 Tag-Libraries zur Verfügung, die eine Vielzahl solcher Custom-Tags bereitstellen.

WebTNDB verwendet davon die HTML, die Bean, die Logic und die Tiles-Library, sowie zusätzlich die SSL-Extension-Library und kommt nahezu ohne Skriptletcode aus. Um die Custom-Tags dieser Bibliotheken verfügbar zu machen, muß eine JSP die zugehörigen **T**ag-**L**ibrary-**D**eskriptoren (TLDs) einbinden, was durch Voranfügen der folgenden Direktiven geschieht :

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"   %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"   %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/sslext.tld" prefix="sslext"%>
```

Ähnlich wie bei einem XML-Namespace, werden dabei Präfizes durch Angabe einer URI eindeutig gebunden, so daß die JSP-Engine im folgenden weiß, daß es sich z.B. beim Tag `<html:submit>` um den in der Struts HTML-Tag-Library

definierten `submit`-Tag handelt, und beim Tag `<logic:forward>` um den in der Struts Logic-Tag-Library definierten `forward`-Tag.

Die Namen der Tags sind dabei so gewählt, daß das Verständnis oft sehr intuitiv ist. Andernfalls wird bei der Erklärung nachfolgender Codebeispiele auch auf die verwendeten Custom-Tags eingegangen.

Speziell die Tiles-Library stellt eine Eigenheit von Struts dar und beschreibt einen Template-Mechanismus, der dazu dient, kleinere JSPs zu größeren zusammenzufügen. Dadurch entsteht ein konsistentes Erscheinungsbild, ohne daß Code kopiert und ständig wiederholt werden muß, was die Anpassungsarbeit bei Änderungen deutlich reduziert. Tiles ermöglichen im Gegensatz zu anderen Technologien, wie SSI oder JSP-Includes, die dem gleichen Zweck dienen, Seitendefinitionen in einem objektorientierten Stil. Auf die Verwendung von Tiles in WebTNDB wird später noch detailierter eingegangen. Ein vollständiger Überblick über alle anderen Struts Tag-Libraries ist in der Online-Dokumentation ([AJS03]) oder im Referenzteil ab Kapitel 13 in [Goo02, S. 249] zu finden.

Alle JSPs aus denen die Views von WebTNDB zusammengesetzt werden, sind im Unterverzeichnis `pages` im WebSpace der Applikation zu finden.

RequestProcessor, ActionServlet und Actions : Die Struts Controller Komponenten Das Kontrollflußmanagement ist eine der schwierigsten Aufgaben, die auf der Präsentationsschicht zu bewältigen sind. Struts nimmt WebTNDB hier die meiste Arbeit ab, da die Verwaltung von Systemzuständen innerhalb des Frameworks geschieht.

Die zentrale Controller-Komponente von Struts ist das `org.apache.struts.action.ActionServlet`. Hierbei handelt es sich um die Struts Implementierung des durch den „Model 2 Approach“ beschriebenen Controller-Servlets. D.h. daß alle Anfragen zunächst von dieser zentralen Komponente entgegengenommen werden, und von dort an das zur Verarbeitung einer speziellen Anfrage konfigurierte Objekt weitergeleitet werden. Für jede Struts-Anwendung muß daher im Web-Application Deployment-Deskriptor ein Mapping für dieses Servlet vereinbart sein. Das ist die Datei `web.xml`, die Informationen über die Container-Konfiguration für eine Web-Applikation enthält.

Für WebTNDB sind dort die folgenden XML-Elemente eingetragen, die das benötigte Mapping beschreiben[22] :

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Das `<servlet>`-Element stellt das eigentliche Deployment dar. Es teilt dem Container mit, daß er ein Servlet unter dem Namen `action` bereitstellen soll, daß dieses durch die Klasse `ActionServlet` im `action` Paket der Struts-Libraries implementiert ist, und, daß diesem der Parameter `config` übergeben werden soll. Der Wert dieses Parameters gibt an, daß die Struts-Konfiguration in der Datei `WEB-INF/struts-config.xml` gespeichert ist.

Das zweite Element vereinbart ein Mapping, das in diesem Fall festlegt, daß WebTNDB alle Anfragen mit der Endung `do` an das Servlet weiterleitet, das unter dem Namen `action` deployed wurde.

Anhand der durch den `config`-Parameter spezifizierten Struts-Konfigurationsdatei kann das Objekt bestimmt werden, das zur Verarbeitung einer speziellen Anfrage vorgesehen ist. Bei diesem Objekt handelt es sich immer um eine sogenannte Action. Das sind Instanzen von Klassen, die von der Basisklasse `org.apache.struts.action.Action` abgeleitet wurden. Die Konfigurationsdatei enthält die Information, welche Action für die Verarbeitung welcher Anfragen zuständig ist. Bei der Implementierung einer Action überschreibt man die aus der Basisklasse geerbte `execute( )`-Methode, um eine eigene Verarbeitungsroutine bereitzustellen. Meist sind in der Struts-Konfigurationsdatei mehrere Actions innerhalb eines XML `<action-mappings>`-Elements vereinbart. Eine Struts-Applikation besteht i.a. aus einem `ActionSevlet`, einer Vielzahl von `Action`-Klassen und deren Konfiguration. Das `ActionServlet` ist innerhalb des Frameworks implementiert. Um ein neues Web-Interface auf Basis des Struts-Frameworks zu bauen, werden also `Action`-Klassen implementiert, Views durch JSP oder Tiles-Definitionen gestal-

[22] Die hier abgedruckte Information ist gekürzt, um das Verständnis zu erleichtern. Tatsächlich enthält die Datei `web.xml` auch die Information darüber, welche Seite die Startseite der Applikation ist, diverse Log-Level Einstellungen und zu jeder referenzierten Tag-Library-URI den Ort, an dem die zugehörige `tld`-Datei gespeichert ist.

tet und eine Struts-Konfigurationsdatei in XML geschrieben. Für WebTNDB sind in der Datei `struts-config.xml` mehr als 80 Actions vereinbart.

Struts liest bei der Initialisierung die Konfigurationsdatei und erzeugt aus den XML-Elementen Instanzen von Java-Klassen, deren Attribute danach die Informationen über das Laufzeitverhalten der Applikation enthalten. Diese Konfigurationsobjekte werden vom `org.apache.struts.action.RequestProcessor` verwaltet, der sie für die Anfrageverarbeitung nutzt. Das `ActionServlet` leitet alle Anfragen an den `RequestProcessor` weiter. Dieser ermittelt anhand der Struts-Konfiguration, welche `Action` für die Verarbeitung einer Anfrage zuständig ist und ruft ihre `execute( )`-Methode auf.

Der Methode `execute( )`wird dabei die Deployment-Information des zugehörigen `<action>`-Elements aus der Struts-Konfigurationsdatei als Objekt vom Typ `org.apache.struts.action.ActionMapping` als Parameter übergeben.

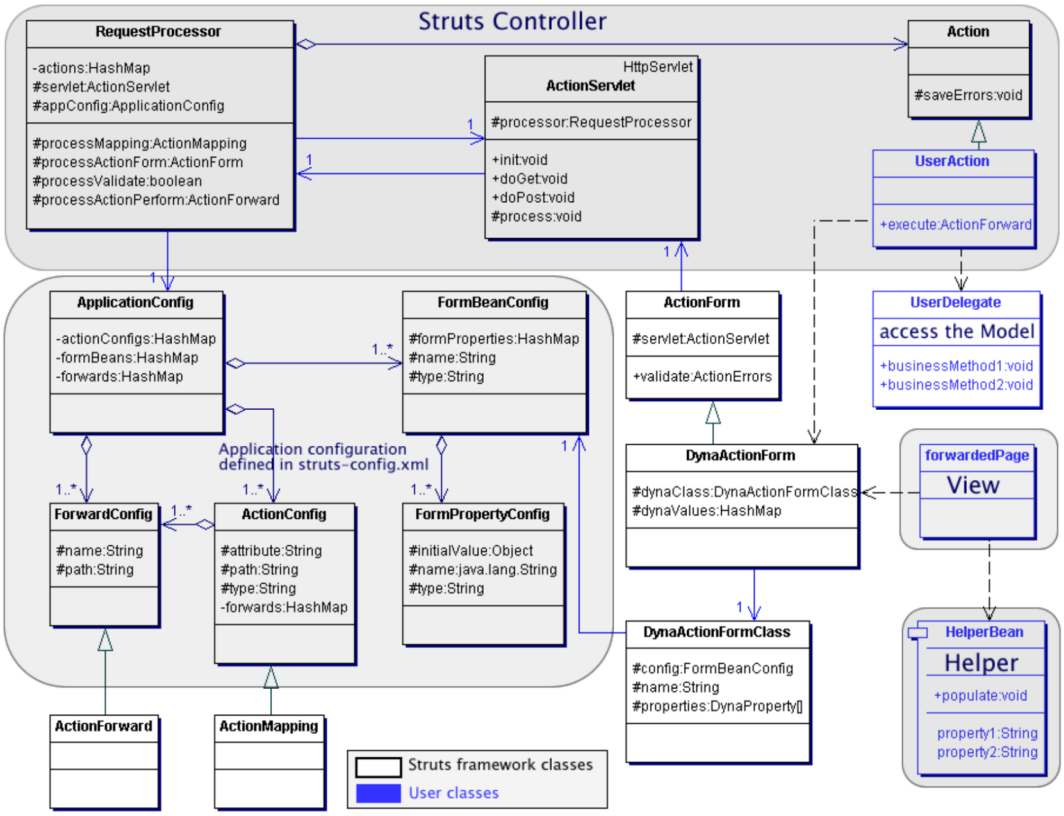
Die zugehörige DTD definiert die wichtigsten Bestandandteile eines `<action>`-Elements folgendermaßen[23] :

```
<!ELEMENT action (set-property*,forward*)>
<!ATTLIST action name %BeanName; #IMPLIED>
<!ATTLIST action path %RequestPath; #REQUIRED>
<!ATTLIST action type %ClassName; #IMPLIED>
```

`type` spezifiziert dabei die `Action`-Klasse mit der `execute( )`-Methode zur Verarbeitung der Anfrage. `path` gibt den Request-Pfad der Anfragen an (ohne die `do`-Extension), die durch die beschriebene `Action` vearbeitet werden sollen. Im `name` Attribut kann die Klasse einer von `org.apache.struts.action.ActionForm` abgeleiteten JavaBean spezifiziert werden, deren Properties Struts die Werte in den Feldern eines HTML-Formulars zuordnet. Für `<forward>`-Subelemente wird jeweils ein Name und ein Pfad angegeben. Innerhalb der Actions dienen die `<forward>`-Elemente des zugehörigen Mappings zur Kontrollflußsteuerung. Die Actions rufen dazu die Methode `findForward( )` des `ActionMapping`-Objekts mit dem Namen eines solchen `<forward>`-Elements als Parameter auf, um den Pfad zu bestimmen, an den als nächstes verzweigt werden soll. Ein solcher Pfad führt entweder direkt zur nächsten `Action` („action chaining“, [Goo02, S. 198]) oder zum nächsten View, der durch eine JSP oder Tiles-Definition spezifiziert wird. Welche optionalen Properties Bestandteil eines WebTNDB Action-Elements sein dürfen, wird noch im Abschnitt über die Anpassungen des Frameworks an WebTNDB besprochen, der dieses Kapitel abschließt.

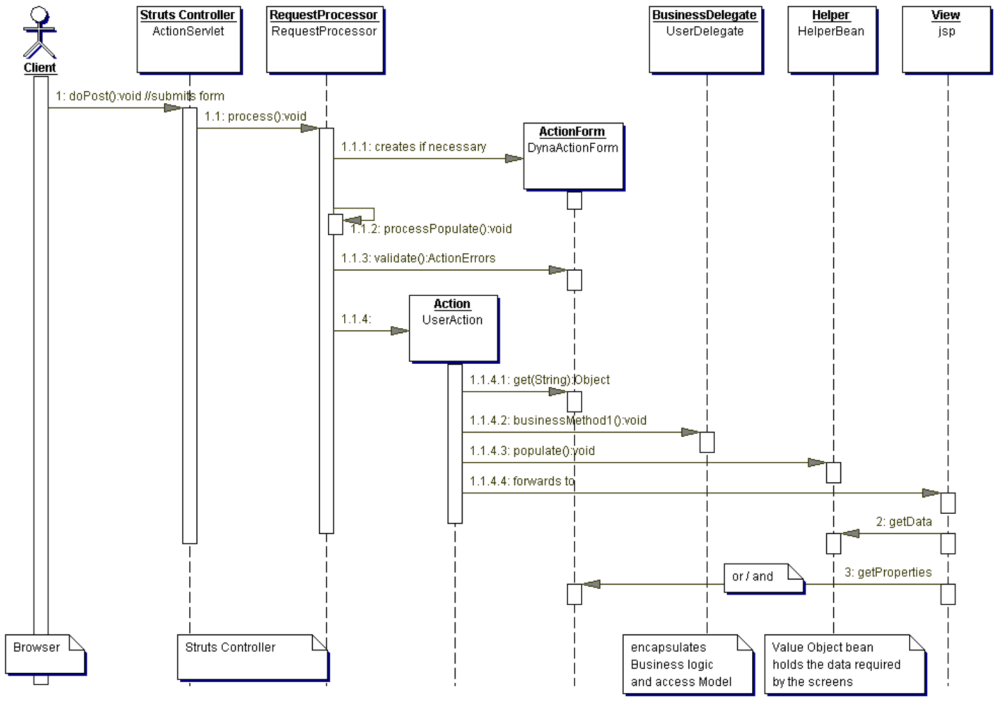
[23] Die vollständige und ausführlich kommentierte Version dieser DTD ist unter http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd verfügbar.

Auf der Web-Seite von Jean-Michel Garnier werden die Beziehungen der Komponenten von Struts durch UML-Diagramme grafisch veranschaulicht ([Gar02]). Folgendes Klassendiagramm zeigt alle Controller-Komponeten einer Struts Applikation :



Nur die blau dargestellten Klassen sind nicht bereits durch das Framework implementiert. Schon beim ersten Blick auf das Design ist zu erkennen, wie groß der Arbeitsanteil ist, den Struts dem Entwickler beim Entwurf eines Web-Interface abnimmt. Das Bild zeigt zwar stellvertretend für alle Actions nur eine Erbenklasse der `org.apache.struts.action.Action`. Bei der Implementierung weiterer Actions müssen aber keine neuen Design-Entscheidungen mehr getroffen werden.

Die Darstellung der statischen Struktur der Klassen, kann deren Funktionsweise nicht zeigen. Der Zweck der komplexen Struktur innerhalb des Frameworks ist daher in dem Klassendiagramm nicht zu erkennen. Das folgende Sequenzendiagramm stellt die Interaktionen zwischen den Objekten bei der Verarbeitung einer Anfrage dar.



Die Vorgänge innerhalb der Komponenten werden hier anhand eines konkreten Beispiels gezeigt.

Eine Beispielanwendung : ColorAction Eine JSP zur Generierung eines Formulars könnte z.B. so aussehen :

```
<html:form action="/Dispatch.do" method="post">
  <TABLE>
    <TR> <TD>next page : </TD>
         <TD><html:text property="path"/></TD>
    </TR>
    <TR><TD COLSPAN="2" ALIGN="center">
         <html:submit/></TD>
    </TR>
  </TABLE>
</html:form>
```

Die `html`-Custom Tags erzeugen ein `<FORM>`-Element, ein `<INPUT>`-Element mit `TYPE="text"` und `NAME="path"` und ein `<INPUT>`-Element mit `TYPE="submit"`, wie man es leicht an ihren Namen erkennen kann. Interessant ist das Attribut `action` des `<html:form>`-Elements. Der Wert dieses Attributs gibt die URL

an, an die das Formular übermittelt werden soll. Hier ist zu sehen, daß sich eine Action wie ein Servlet referenzieren läßt.

Für den Pfad „/Dispatch.do“ muß in der Struts-Konfigurationsdatei ein Mapping vereinbart sein, das man sich folgendermaßen vorstellen kann :

```
<action path="/Dispatch"
        type="com.somedomain.actions.ColorAction"
        name="pathForm"
        validate="true">
  <forward name="red" path="/redpage.jsp"/>
  <forward name="green" path="/greenpage.jsp"/>
  <forward name="error" path="/Logout.do"/>
</action>
```

Der Wert „pathForm“ im Attribut `name` referenziert eine `ActionForm` Subklasse, die in der Konfigurationsdatei durch ein `<form-bean>`-Element vereinbart sein muß. Am einfachsten hat man es, wenn man hier ein `DynaActionForm` verwendet, das einem erspart zur Speicherung der Felder des Formulars selber eine Bean-Klasse zu implementieren. Die Definition dafür sieht so aus :

```
<form-bean name="pathForm" dynamic="true"
           type="org.apache.struts.action.DynaActionForm">
  <form-property name="path" type="java.lang.String"/>
</form-bean>
```

Dadurch wird eine sogenannte dynamische FormBean deklariert. Diese stellt für jedes `<form-property>`-Element eine gleichnamige Property bereit, auf die mit den Methoden `get(„`*propName*`“)` und `set(„`*propName*`“,` *propValue*`)` zugegriffen werden kann.

Zusammen mit dem zuvor vereinbarten `<action>`-Element wird also festgelegt, daß der Wert *value*, den der Benutzer in das Textfeld „path“ eingibt, durch einen Aufruf von *pathForm*`.set(„path“,` *value*`)` als Property einer FormBean gespeichert werden soll, die durch die Klasse `DynaActionForm` definiert ist. Die FormBean wird vom `RequestProcessor` als Parameter an die `execute( )`-Methode eines Objekts der Klasse `ColorAction` übergeben und stellt somit deren Sicht auf das Formular dar. Die Bedeutung des `validate` Attributs wird später, bei der Auflistung spezieller Features von Struts, noch genau erklärt.

Nach einem Klick auf die „submit“-Schaltfläche des Formulars laufen die einzelnen Schritte zur Anfragebearbeitung ab, wie in obigem Sequenzendiagramm dargestellt. Was die dargestellten Interaktionen im Beispiel bewirken, wird hier schrittweise zusammengefasst :

- 1 : Der Browser schickt das Formular in einem HTTP-`POST`-Request (da `method="post"`) an die URL www.somedomain.com/context/Dispatch.do. Der Web-Container ordnet der URL-mit der `*.do`-Extension das

`org.apache.struts.action.ActionServlet` zu. Der Request wird folglich in ein Java-Objekt mit der Schnittstelle `javax.servlet.http.`*`HttpServletRequest`* gepackt, das der `service( )`-Methode des `ActionServlets` als Parameter übergeben wird. Diese leitet den POST-Request an die Methode `doPost( )` weiter.

- 1.1 : Die `doPost( )`-Implementierung des `ActionServlets` leitet den Request zur Verarbeitung an die `process( )`-Methode des `RequestProcessors` weiter, die anhand des Pfades (ohne die `do`-Extension) das zur Verarbeitung dieser Anfrage in der Struts-Konfigurationsdatei spezifizierte `ActionMapping` bestimmt.
  Das ActionMapping beschreibt das weitere Vorgehen des `RequestProcessors` :

- 1.1.1 : Zuerst fordert der `RequestProcessor` eine FormBean des vereinbarten Typs `DynaActionForm` mit der Property `path` an, die Struts entweder aus einem Pool vorgefertigter FormBeans entnimmt oder neu instanziiert.

- 1.1.2 : Die Methode `processPopulate( )` des `RequestProcessors` speichert den Request-Parameter „`path`“ als `path`-Property der FormBean.

- 1.1.3 : Die Methode `validate( )` des `DynaActionForm`-Objekts tut gar nichts. Wäre im Mapping anstatt der dynamischen FormBean aus dem Framework eine andere Subklasse von `ActionForm` angegeben, könnte diese `validate( )` überschreiben, um Eingaben des Benutzers auf Gültigkeit zu überprüfen. Wie man eine Validierung für dynamische FormBeans vereinbart, wird später erklärt.

- 1.1.4 : Der `RequestProcessor` fordert als nächstes eine Instanz von `ColorAction` an und ruft deren `execute( )`-Methode auf. Die FormBean und das `ActionMapping` werden dem `ColorAction`-Objekt bei diesem Aufruf als Parameter übergeben.

- 1.1.4.1 : Das `ColorAction`-Objekt liest die `path`-Property der FormBean, die ihm übergeben wurde.

- 1.1.4.2 : Im `ColorAction`-Beispiel kann die `execute( )`-Methode das Formular selber verarbeiten und das Versenden von Nachrichten an weitere Objekte daher entfallen. Im Sequenzendiagramm ist angedeutet, daß zur weiteren Anfrageverarbeitung Methoden eines Objekts `UserDelegate`, das eine Implementierung des J2EE-Patterns Business-Delegate ([SJC02, S. 248]) darstellt, aufgerufen werden können. Das Business-Delegate Entwurfsmuster beschreibt, wie Komponeten der Business-Tier lose an Komponeten der Presentation-Tier gekoppelt werden, um Abhängigkeiten zu minimieren. Manche Komponenten der Business-Tier von WebTNDB sind ähnlich gebaut. Für das Verständnis des `ColorAction`-Beispiels kann das Objekt `UserDelegate` aus dem Sequenzendiagramm ignoriert werden.

- 1.1.4.3 : Um Folge-Actions zu parametrisieren, verwendet man oft `ValueObjects`, die einfache Daten-Puffer darstellen, wie schon im Abschnitt über die Integration-Tier besprochen wurde. Das `ColorAction`-Beispiel kommt ohne solche Puffer aus.

- 1.1.4.4 : Die `execute( )`-Methode ruft die `findForward( )`-Methode des `ActionMapping`-Objekts auf, um ein Ziel zu ermitteln, das als `<forward>`-Element in der zugehörigen `<action>`-Deklaration vereinbart wurde. Für das Ziel mit dem Namen „`error`“ ist der Pfad „`/Logout.do`“ angegeben, zu dem man sich eine weitere Action vorstellen kann, die beispielsweise die Session invalidiert.

- 2, 3 : Die Seiten greenpage.jsp und redpage.jsp können mittels Skriptlet-Code oder Custom-Tags aus der Struts-Bean-Library auf Properties zugreifen, die von der vorgeschalteten Action für die FormBean oder für zusätzliche ValueObjects gespeichert wurden, um die Darstellung zu parameterisieren. Davon wird gerne Gebrauch gemacht, um die Felder in Formularen auf Folgeseiten vorab mit Default-Werten zu belegen. In diesem ersten Beispiel ist das nicht nötig.

Um die FormBean und die Action-Instanz anzufordern ruft der `RequestProcessor` verschiedene Klassenmethoden des Toolkits `org.apache.struts.util.RequestUtils` auf, das im Sequenzendiagramm der Übersicht wegen nicht dargestellt ist.

Eine weitere einfache Action ist in der Struts-Konfigurationsdatei von WebTNDB für den Pfad „`/Login`“ vereinbart. Diese kann als zusätzliches Beispiel eingesehen werden, um die oben dargestellten Schritte nochmals nachzuvollziehen. Die „`/Login`“-Action überprüft die Zugangsdaten bei der Anmeldung eines neuen Benutzers durch das Anmeldeformular `pages/tiles/login.jsp`. Dabei kommen aber bereits weitere Features zum Einsatz, die erst später besprochen werden.

Hier wird zunächst eine mögliche Implementierung für die `ColorAction`-Klasse aus obigem Beispiel angegeben, um die Vorstellung von der Anfragebearbeitung durch Struts weiter zu konkretisieren.

Die Implementierung von Struts-Actions Im Gegensatz zu den komplexen Vorgängen innerhalb des Frameworks sieht die Implementierung einer Action-Klasse erfreulich einfach aus :

```
package com.somedomain.actions;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.DynaActionForm;

public class ColorAction extends Action {

 public ActionForward execute( ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response )
 throws IOException, ServletException {
  String target = "error";
  String path =
   (String)((DynaActionForm)form). get(„path“);
  if (path.equalsIgnoreCase(„redpage“))
   target = „red“;
  else if (path.equalsIgnoreCase(„greenpage“))
   target = „green“;
  return (mapping.findForward(target));
 }

}
```

Der RequestProzessor gibt die Parameter `request` und `response`, die der Container dem `ActionServlet` übergibt, an jede Action weiter. HTTP-Anfragen können folglich in einer Struts-Action bedient werden, wie in einem gewöhnlichen Servlet. Hier werden diese Parameter nicht verwendet.

Stattdessen zeigt das einfache Beispiel bereits alle nötigen Schritte zur Verarbeitung einer Anfrage mithilfe der Stuts-spezifischen Objekte mit den Typen `ActionMapping`, `ActionForm` und `ActionForward`.

Die erste Zeile der `execute( )`-Methode speichert „`error`“ als Default-Ziel der `ColorAction`. Wenn also im folgenden keine gültige Auswahl einer Folgeseite festgestellt werden kann, wird der Client an die URL

„www.somedomain.com/context/Logout.do“ weitergeleitet, die für das Ziel „error“ im Mapping vereinbart wurde. Um diesen Request zu bedienen, muß für den Pfad „/Logout“ eine weitere Action vereinbart sein, die den Fehler verarbeitet, indem sie die Sitzung beendet.

Der form Parameter referenziert die FormBean pathForm, die zu einem DynaActionForm gecastet werden kann, da ihr Typ entsprechend vereinbart wurde. Mit der get( )-Methode wird die path-Property gelesen.

Hat der Benutzer in das path-Textfeld der Formularseite „redpage“ eingegeben, wird als Ziel „red“ gespeichert. Für die Eingabe „greenpage“ speichert die Action „green“.

mapping referenziert das ActionMapping Objekt mit den Deployment-Daten. Insbesondere kennt mapping die <forward>-Elemente und kann folglich für die Namen „red“ und „green“ das tasächlich gespeicherte Ziel ermitteln. Dabei handelt es sich um die Seiten „redpage.jsp“ und „greenpage.jsp“, für die kein Beispiel-Code angegeben wird, da man sich dafür eine sehr einfache Gestaltung vorstellen kann.

Die Methode findForward( ) bestimmt dieses Ziel als Instanz der Klasse ActionForward, die an den RequestProcessor zurückgegeben wird, damit dieser daraus den Folge-Request generiert.

Durch ein flexibles Action-Design kann ein und dieselbe Action-Klasse unterschiedliche Anfragen bedienen, und Veränderungen müssen nur an der Konfiguration vorgenommen werden. Alle der mehr als 80 Actions, die in der Struts-Konfigurationsdatei für WebTNDB vereinbart sind, können durch ca. 50 Klassen, die im Subpackage actions des Pakets struts vereinbart sind, verarbeitet werden.

B.2 Struts Features

Bei der Verwendung von Struts profitiert eine Anwendung von verschiedenen Features, wie geschickten Pooling-Mechanismen für FormBean und Action-Instanzen, die ein Performace-Plus bringen, ohne daß der Entwickler selbst etwas dazu tun muß. WebTNDB profitiert weiterhin von verschiedenen zusätzlichen Features von Struts, die einzeln konfiguriert werden müssen. Die wichtigsten davon werden im folgenden kurz vorgestellt.

Internationalization Support Eine wichtige Eigenschaft der aktiven temporalen Datenbank WebTNDB ist, daß unabhängig von bestimmten Zeit- und Kalendersystemen auf Zeitdaten zugegriffen werden kann. Um internationalen Anwendern den Zugang zum System zu erleichtern, ist die Benutzerschnittstelle auch nicht an eine bestimmte Sprache gebunden.

Komponeten einer Web-Applikation, die zur Speicherung von sprachspezifischen Informationen dienen, heißen auch i18n-Komponenten[24]. Bei einer Struts Anwendung sind die i18n-Komponeten sogenannte Resource Bundles.

[24]Diese beliebte Abkürzung geht darauf zurück, daß das englische Wort internationalization zwischen dem i und dem n aus 18-Buchstaben besteht ([Goo02, S.105]).

Resource Bundles sind Dateien, die Schlüssel- und Wertepaare für alle Textausgaben der Applikation speichern. Dabei handelt es sich um alle auf der Benutzeroberflache angezeigten Texte, inklusive der Beschriftungen der Schaltflächen etc. . Der Ort von Resource Bundles wird in <message-resources>-Elementen in der Struts-Konfiguration angegeben.

Für WebTNDB ist dort die Datei
WEB-INF/classes/i18n/ApplicationResources.properties angegeben. Hier sind alle Text-Ausgaben des Testsystems in englischer Sprache gespeichert. Ein Übersetzer kann den Inhalt dieser Datei ohne Programmierkenntnisse in eine andere Sprache übersetzen.

Der Server wählt für jeden Client das Resource Bundle in der Sprache, die in seinem Browser-Programm als bevorzugte Sprache voreingestellt ist. Folglich präsentiert sich die Applikation dem Benutzer automatisch in seiner Wunschsprache, sofern dafür ein Resource Bundle verfügbar ist.

Der Einführungstext, der Informationen über die Anwendung enthält, die auf der Anmeldeseite angezeigt werden, ist auch in der Datei ApplicationResources.properties gespeichert. Ein Server-Administrator kann diesen Text hier natürlich auch verändern und für eine neue Installation des Web-Interface dort bespielsweise eine spezifische Beschreibung der Daten eintragen, auf die durch das Web-Interface auf diesem Server zugegriffen wird.

Eingabevalidierung Ein Web-Interface sollte Benutzer-Eingaben prüfen, bevor sie an Komponenten tieferliegender Schichten übertragen werden oder gar persistent gespeichert werden. Bei WebTNDB betrifft das u.a. Eingaben bei der Registrierung wie zum Beispiel das gewünschte Passwort und die angegebene Email-Adresse. Das Passwort muß mit der Paßwortbestätigung übereinstimmen, damit sicher ist, daß sich der Benutzer hier nicht vertippt hat. Die Email-Adresse sollte einem Muster für gültige Email-Adressen entsprechen daß u.a. vorgibt, daß die Adresse ein @-Zeichen enthält. Bei der Bedienung der Anwendung sind weiterhin unterschiedliche Eingaben z.B. für die Spezifikation des Anfrageintervalls vorzunehmen, bei denen Benutzer, die aufgrund von Tippfehlern oder Mißverständnissen ungültige Daten eingeben, darauf aufmerksam gemacht werden müssen.

Es gilt allgemein als guter Stil solche Fehler bereits auf der Präsentationsschicht auszuschließen, damit Komponeten tieferliegender Schichten dafür keine speziellen Vorkehrungen mehr treffen müssen. Struts bietet dazu einen praktischen Validierungsmechanismus an :

Hat das validate Attribut eines `<action>`-Elements in der Struts-Konfigurationsdatei den Wert `true`, wird die Methode `validate( )` der im `name` Attribute referenzierten FormBean aufgerufen, nachdem der `RequestProcessor` die Attribute der FormBean, mit den Benutzereingaben aus dem Formular belegt hat. Eine eigene FormBean-Klasse für jedes Formular kann durch ihre eigene `validate( )`-Definition dann jeweils eine Validierung der Attributwerte beschreiben. Um nicht soviel Code schreiben zu müssen, ist es aber auch möglich, dynamische FormBeans des speziellen Typs `DynaValidatorForm`

zu vereinbaren, für die eine deklarative Validierung möglich ist.

Dazu wird der Struts-Konfiguarion eine Plugin-Vereinbarung hinzugefügt, die folgendermaßen aussieht :

```
<plug-in
  className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
      value="/WEB-INF/validation/validator-rules.xml,
             /WEB-INF/validation/validation.xml"/>
</plug-in>
```

Die `pathnames`-Property gibt dabei die Pfade der Dateien mit den Validierungsregeln an. `validator-rules.xml` ist eine Datei aus der Struts-Distribution mit vorgefertigten Definitionen für die Validierung von Standardfällen, wie z.B. einem Test auf rein numerische Eingaben. Die Datei `validaton.xml` wird beim Deployment an den angegebenen Ort kopiert und dient zur Spezifikation anwendungsspezifischer Validierungen. Für Eingabefelder in einem Formular kann dort in `<field>`-Definitionen beispielsweise ein Muster durch einen regulären Ausdruck vorgegeben werden :

```
<field property="email"
       depends="required,mask">
  <msg name="mask"
       key="app.registerForm.emailmsg"/>
  <var><var-name>mask</var-name>
       <var-value>${email}</var-value>
  </var>
  <arg0 key="app.registerForm.email"/>
</field>
```

Obige Definition aus der WebTNDB-Konfiguration gibt an, daß die Email-Adresse, die der Benutzer in das Registrierungsformular eingibt, dem regulären Ausdruck in der Variablen `email` entsprechen muß. Eine große Auswahl brauchbarer regulärer Ausdrücke für solche Definitionen steht z.B. auf http://www.regexlib.com bereit. Weiterhin wird ein Schlüssel angegeben, mit dem Struts eine Fehlermeldung in der Sprache des Benutzers aus dem zugehörigen Resource Bundle auswählen kann, falls eine ungültige Email-Adresse eingegeben wird.

Das 12. Kapitel im Struts Buch von Ted Husted enthält eine sehr ausführliche und einfach verständliche Erklärung aller weiteren Möglichkeiten zur deklarativen Validierung dynamischer FormBeans und kann kostenfrei von der Verlags-Hompage heruntergeladen werden ([HCW02, S. 365]).

Seitenlayout durch Tiles-Definitionen Das Erscheinungsbild aller Seiten von WebTNDB gleicht sich in vielen Bestandteilen. Hintergrundfarbe und Muster, Logo und Schriftfarben, die sich nie ändern, sollten möglichst nicht für jede Seite neu definiert werden. Auch das Layout aller Seiten gleicht sich in einigen

Elementen, wie z.B. der links positionierten Kontrollleiste und dem Logo am oberen Fensterrand.

Struts Applikationen steht in Form der Tiles-Libraries von Cedric Dumoulin ein fortschrittlicher Mechanismus für parametrisierte Views und Seitentemplates zur Verfügung ([Dum02]).

Tiles sind JSP-Quelltext Dateien, die einzelne Bestandteile der Views einer Web-Applikation beschreiben. Mithife der Tiles Custom-Tag Library lassen sich dort Attribute von Darstellungselementen parametrisieren und ein flexibles Template zusammenstellen.

Um Tiles zu verwenden, muß ein weiteres Plugin in der Struts-Konfigurationsdatei vereinbart werden :

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config"
                value="/WEB-INF/tiles-defs.xml" />
</plug-in>
```

Die `definitions-config`-Property gibt die Datei mit den View-Definitionen an. In der oben referenzierten Datei `WEB-INF/tiles-defs.xml` ist unter dem Namen `.webtndb.BaseDef` die Basisdefiniton aller WebTNDB-Views gespeichert[25] :

```
<definition name=".webtndb.BaseDef"
            path="/pages/tiles/layout/Template.jsp">
  <put name="header" value="pages/tiles/header.jsp"/>
  <put name="footer" value="pages/tiles/copyright.jsp"/>
  <put name="textColor" value="#c5def7"/>
  <put name="backgroundColor" value="#03274c"/>
</definition>
```

Wie man sieht, werden Darstellungseigenschaften, in Tiles-Definitionen, durch `<put>`-Elemente spezifiziert. Für die Eigenschaften „`textColor`“ und „`backgroundColor`“ sind deren Werte direkt angegeben. Anstelle dieser Wertangeben ließe sich hier natürlich auch ein Stylesheet für die Seitenformatierung spezifizieren. „`header`“ und „`footer`“ referenzieren zusätzliche Dateien. Dabei handelt es sich um Tiles, die in Subdefinitionen zu Views vereint werden. Das `path`-Attribute gibt dazu eine JSP-Seite an, die als Template dient. In dieser Template werden die Tiles, wie in der jeweiligen Subdefinition spezifiziert, angeordnet.
Die Datei `/pages/tiles/layout/Template.jsp` gibt für alle Seiten von WebTNDB eine Dreiteilung vor[26] :

[25] Die vollständige Definition kann in der Datei `/WEB-INF/tiles-defs.xml` eingesehen werden. Dort finden sich einige weitere `<put>`-Elemente zur Definition gemeinsamer Basiseigenschaften der Views.

[26] Einzelne `<put>`-Definitionen, die zusätzliche Eigenschaften der Basisdefinition an die eingefügten Tiles weitergeben, wurden hier (...) wieder ausgelassen.

```
<!-- header -->
  <tiles:insert name="header">
    ...
  </tiles:insert>
<!-- body -->
  <tiles:insert name="body">
    <tiles:put name="dispatch">
      <tiles:getAsString name="dispatch"/>
    </tiles:put>
    <tiles:put name="bodyFrame">
      <tiles:getAsString name="bodyFrame"/>
    </tiles:put>
    <tiles:put name="menu">
      <tiles:getAsString name="menu"/>
    </tiles:put>
    ...
  </tiles:insert>
<!-- footer -->
  <tiles:insert name="footer">
    ...
</tiles:insert>
```

Hier ist zu sehen, wie Tiles durch den Custom-Tag `<tiles:insert>` zu Views zusammengefügt werden können. Der Tag `<tiles:getAsString>` dient zum Zugriff auf Vereinbarungen der Basisdefinition und wird hier verwendet, um die dort deklarierten Eigenschaften an die eingesetzten Tiles weiterzugeben.

Für die einzelnen Views von WebTNDB sind in der Datei `tiles-defs.xml` Subdefinitionen der Basisdefinition angegeben. Die Tiles-Definition der Startseite sieht dort z.B. so aus :

```
<definition name=".webtndb.Welcome"
            extends=".webtndb.BaseDef">
  <put name="menu" value="pages/tiles/tohttps.jsp"/>
  <put name="content" value="pages/tiles/text.jsp"/>
  <put name="titleKey" value="app.webtndb.headline"/>
</definition>
```

Die Anmeldeseite ist folgendermaßen definiert :

```
<definition name=".webtndb.Logon"
            extends=".webtndb.Welcome">
  <put name="menu" value="pages/tiles/login.jsp"/>
  <put name="contentKey" value="app.intro.text"/>
  <put name="titleKey" value="app.intro.headline"/>
</definition>
```

Nicht nur die Basisdefinition, sondern auch Subdefinitionen können also durch weitere Subdefinitionen erweitert werden. Mithilfe einer derart definierten Vererbungshierarchie, lassen sich die Views einer Applikation sehr elegant beschreiben. Einzelne Eigenschaften, die in Subdefinitionen anzupassen sind, können dort durch neue `<put>`-Elemente gleichen Namens in einem objektorientierten Stil überschrieben werden.

Tiles-Definitionen wie `.webtndb.Welcome` und `.webtndb.Logon` sind durch einen führenden . gekennzeichnet und können nachdem das entsprechende Plugin geladen wurde, von JSP-Definitionen unterschieden werden. Das Plugin selektiert einen speziellen `RequestProcessor`, der Tiles-Definitionen als Ziel einer Action im `path`-Attribut von `<forward>`-Elementen verarbeiten kann, um den Seitenfluß zu den dadurch beschriebenen Views weiterzuleiten. Die `Plugin`- und `TilesRequestProcessor`-Klassen sind wie die Tiles Custom-Tag Library Bestandteil der Struts Standarddistribution, so daß für deren Verwendung nichts zusätzlich installiert werden muß.

Im Struts Buch von Ted Husted wird das Seitenlayout, durch Tiles-Definitionen im 11. Kapitel, das unter Mitwirkung von Cedric Dumoulin entstand, sehr ausführlich besprochen. Auch dieses Kapitel kann kostenfrei von der Verlagsseite heruntergeladen werden ([HCW02, S. 319]).

Literatur

[Ohl02] Hans Jürgen Ohlbach.
WebCal, An Advanced Calendar Server.
Institut für Informatik, Universität München, 2002

[Ohl00] Hans Jürgen Ohlbach.
Calendar Systems and Temporal Notions.
Department of Computing, Imperial College of Science,
Technology and Medicine, London UK 2000

[RD97] Edward M. Reingold, Nachum Dershowitz.
Calendrical Calculations.
Cambridge University Press, UK 1997

[Tan02] Andrew Tanenbaum.
Computer Networks.
Prentice Hall PTR, NJ USA 2002 (4th Edition)

[Nist03] National Institute of Standards and Technology
Role Based Access Control.
http://csrc.nist.gov/rbac/

[CWH00] Mary Campione, Kathy Walrath, Alison Huml,
The Java(TM) Tutorial :
A Short Course on the Basics.
Addison-Wesley Pub. Co., USA 2000 (3rd Edition)
Online Version : http://java.sun.com/docs/books/tutorial

[Paw00] Monica Pawlan,
Essentials of the Java Programming Language.
A Hands-On Guide.
Copyright © 2000 Sun Microsystems Inc. CA USA, published by
Addison-Wesley Longman,
Pearson Education, USA 2000

[Lin98] Peter Van Der Linden.
Java Programmer's FAQ - Part B - Dates and Times.
http://java.sun.com/people/linden/faq_b2.html#Date, 1999

[WA02] Michael Widenius, David Axmark.
MySQL Reference Manual.
O'Reilly & Associates, Incorporated, CA USA 2002,
Online Version : http://www.mysql.com/doc/en/

[Tuu03] Heikki Tuuri.
InnoDB Engine in MySQL: Reference Manual
http://www.innodb.com/ibman.html, 2003

[Sed92] Robert Sedgewick.
Algorithmen in C.
Addison-Wesley Deutschland GmbH,
München Deutschland, 1992

[SH99] Gunter Saake, Andreas Heuer.
Datenbanken: Implementierungstechniken
MITP-Verlag GmbH, Bonn, 1999

[Pek00] Larne Pekowsky,
Java Server Pages,
Addison-Wesley Longman, Massachusetts USA, 2000

[SSJ02] Inderjeet Singh, Beth Stearns, Mark Johnson.
Designing Enterprise Applications with the J2EE Platform.
Addison-Wesley Pub. Co., USA 2002. (Second Edition)
Online-Version :
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e

[MH01] Richard Monson-Haefel,
Enterprise Java Beans.
Developing Enterprise Java Components.
O'Reilly & Associates, CA USA 2001 (3rd Edition)

[GWE02] Michael Gridley, Rob Woollen, Sandra L. Emerson,
J2EE Applications and BEA WebLogic Server.
Prentice Hall PTR, NJ USA 2002

[GHJ97] Erich Gamma, Richard Helm, Ralph Johnson,
Design Patterns.
Elements of Reusable Object- Oriented Software.
Addison Wesley Longman, USA 1997

[SJC02] Sun Java Center.
Core J2EE Patterns:
Best Practices and Design Strategies.
Copyright © 2001-2002 Sun Microsystems Inc., published by
Prentice Hall PTR
Online Version : http://java.sun.com/blueprints/corej2eepatterns/index.html

[BCJKNR02] Craig A. Berry, John Carnell, Matjaz B. Juric, Meeraj Moidoo
Kunnumpurath, Nadia Nashi, Sasha Romanosky.
J2EE Design Patterns Applied.
Real World Development with Pattern Frameworks.
Wrox Press Ltd., Birmingham UK, 2002

[Cra02] William Crawford.
Essential Multitier J2EE Design Patterns.
Oracle Technology Network. Oracle Corporation 2002.
http://otn.oracle.com/oramag/oracle/03-jan/o13j2ee.html

[AJS03] Apache Software Foundation, The Jakarta Projekt.
The Struts User's Guide 2003.
http://jakarta.apache.org/struts/userGuide/index.html

[Dum02] Apache Software Foundation and Cedric Dumoulin.
Tiles Library Documentation. 2002
http://www.lifl.fr/~dumoulin/tiles/doc/tutorial.html

[Dit02] Steve Ditlinger.
Mix Protocols Transparently in Web-Applications
Java World - Fueling Innovation(TM), 2002,
http://www.javaworld.com/javaworld/jw-02-2002/jw-0215-ssl.html

[Goo02] James Goodwill.
Mastering Jakarta Struts.
A code-intensive guide to using the Jakarta Struts Framework to build Java Web applications.
Wiley Publishing, Inc., Indianapolis USA, 2002

[HCW02] Ted Husted, Craig R. McClanahan, David Winterfeldt.
Struts in Action:
A Practical Guide to the Leading Java Web Framework.
Manning Publications Co., CT USA 2002,
Online Sample Chapters : http://www.manning.com/husted

[Gar02] Apache Software Foundation and Jean-Michel Garnier.
Struts Controller UML-Diagrams.
http://rollerjm.free.fr/pro/Struts11.html, 1999-2002

[WFCHH99] Seth White, Maydene Fisher, Rick Cattell, Graham Hamilton, Mark Hapner.
JDBC API Tutorial and Reference,
Universal Data Access for the Java 2 Platform.
Copyright © 1999 Sun Microsystems Inc. CA USA, published by Addison-Wesley Longman,
Pearson Education, USA 1999 (Second Edition)

[Lee02] Rosanna Lee.
The JNDI Tutorial.
Building Directory-Enabled Java-Applications.
Copyright © 2002 Sun Microsystems Inc.
http://java.sun.com/products/jndi/tutorial

[Jim03] *JIMI Software Development Kit*
The Java Image Managing Interface.
Copyright © 1995-2003 Sun Microsystems, Inc.,
http://java.sun.com/products/jimi

[Jai03] *Java Advanced Imaging API. I/O Tools 1.0-RC*
Copyright © 1995-2003 Sun Microsystems, Inc.,
http://java.sun.com/products/java-media/jai

[Go99] Li Gong.
Inside Java 2 Platform Security.
Architecture, API Design and Implementation.
Copyright © 1999 Sun Microsystems Inc. CA USA, published by
Addison-Wesley Longman,
Pearson Education, USA 1999

[Tat02] Bruce A. Tate.
Bitter Java.
The lessons tought by server-side Java antipatterns.
Manning Publications Co., CT USA 2002

[KPS01] Hans-Peter Kriegel, Marco Pötke, Thomas Seidl.
Object-Relational Indexing for General Interval Relationships.
Institut für Informatik, Unversität München. 2001.
Proc. 7th Int'l Symposium on Spacial and Temporal Databases (SSTD01).
Lecture Notes in Computer Science,
Copyright © Springer Verlag 2001

[KPS01] Thomas Seidl, Hans-Peter Kriegel, Marco Pötke.
Interval Sequences: An Object-Relational Approach to Manage Spacial Data.
Institut für Informatik, Unversität München. 2001. Proc. 7th Int'l Symposium on Spacial and Temporal Databases (SSTD01).
Lecture Notes in Computer Science,
Copyright © Springer Verlag 2001

[KPS00] Hans-Peter Kriegel, Marco Pötke, Thomas Seidl.
Managing Intervals Efficiently in Object-Relational Databses.
Institut für Informatik, Universität München, Proceedings of the 26th International Conference on Very Large Databases,
Cairo, Egypt, 2000