1 Motivation und Einführung

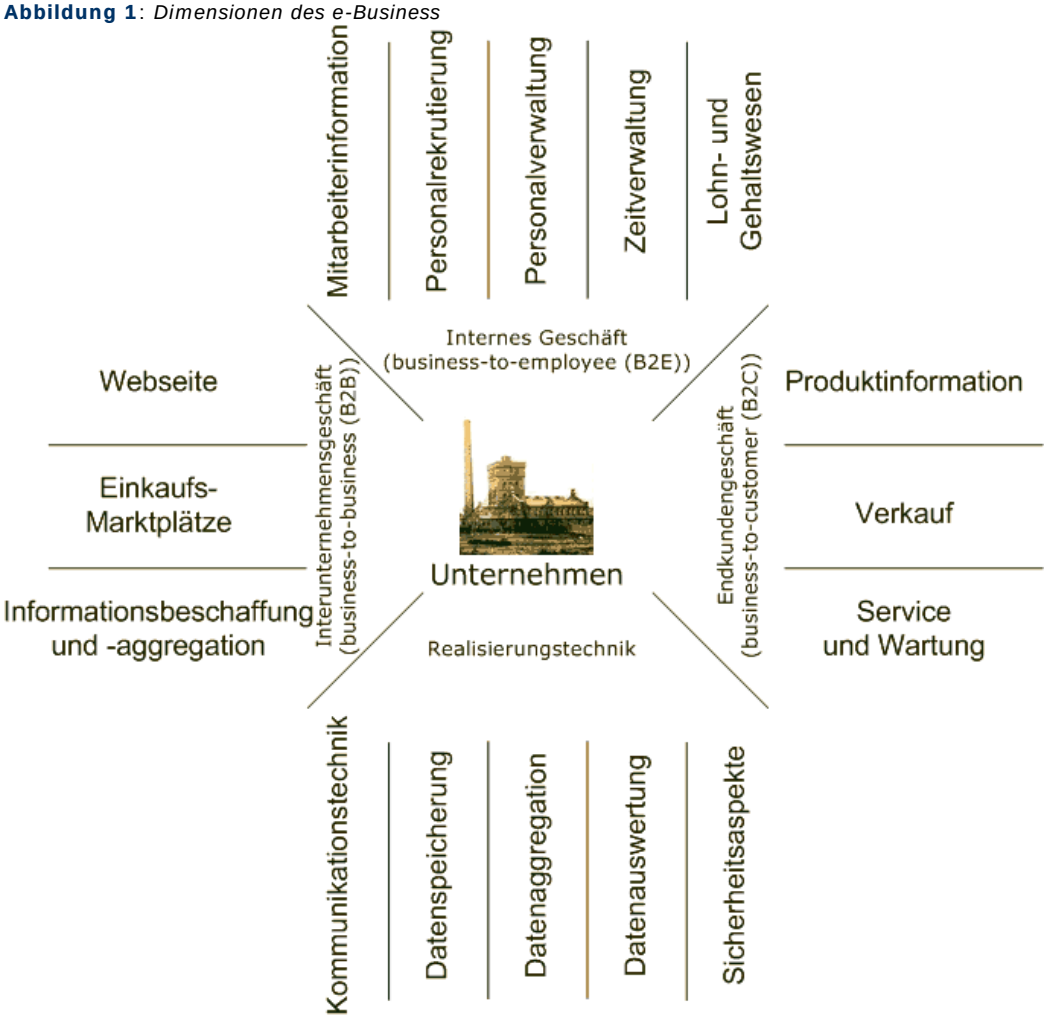
1.1 Was ist e-Business?

Der Begriff des *e-Business* als Abkürzung des englischsprachigen *electronic Business* hat sich inzwischen als Subsumption aller für ein Unternehmen wertschöpfenden Aktivitäten im Internet eingebürgert.
Die Sinngebung greift damit weiter als der historisch ältere Begriff *e-Commerce*, welcher ursprünglich ausschließlich Verkaufsaktivitäten bezeichnete. Inzwischen werden beide Terme jedoch nahezu synonym verwendet. Teilweise findet sich für den Teilbereich des internetgestützten Verkaufs von Waren und Dienstleistungen an Endkunden auch die Bezeichnung *e-tailing* (für *electronic retailing*) welcher jedoch nur einen Teilaspekt des e-Commercebegriffes abzudecken vermag.

Definition 1: e-Business
Electronic Business ist die Gesamtheit aller unternehmerischen Aktivitäten im Internet.

Gemäß dieser allgemeinen Definition werden sämtliche auf das Unternehmensziel gerichtete nach außen wirkende Aktivitäten als e-Business eingeordnet.
Gleichzeitig ergibt sich aus der Abstützung auf der Realisierungstechnik des Internets auch eine interne Sichtweise, sobald diese Technik innerhalb des Unternehmens zum Einsatz kommt.
Die Darstellung der Abbildung 1 unternimmt den Versuch der Einordnung der sich ergebenen Anwendungsdimensionen des e-Businessbegriffs.

Abbildung 1: *Dimensionen des e-Business*



(click on image to enlarge!)

Die naheliegendste Form des e-Business ist der Geschäftsverkehr mit dem (End-)Kunden, als dem typischen Konsumenten der durch ein Unternehmen zur Verfügung gestellten Güter und Dienstleistungen. Dieser Teilbereich wird mit dem Begriff *Business-to-Customer* (B2C) belegt.
In diese e-Businessvariante fallen alle Interaktionen zwischen Kunde und Unternehmen während des gesamten Lebenszyklus des angebotenen Produkts, angefangen von verkaufsfördernden Maßnahmen (Marketing) über den Verkaufs- bzw. Dienstleistungserbringungsakt selbst bis hin zur Abwicklung der Wartung, soweit nach Art des angebotenen Gutes elektronisch überhaupt möglich.

Entgegengesetzt zum durch ein Unternehmen produzierten ausgehenden Güter- und Dienstleistungsstrom verläuft die Beschaffung von nicht-menschlichen Produktionsfaktoren wie Roh-, Hilfs- und Betriebsstoffen sowie die Interunternehmenskommunikation. Dieser Teilbereich wird mit dem Begriff *Business-to-Business* (B2B) belegt.
In diese e-Businessvariante fallen die zwischen Unternehmungen ablaufenden elektronischen Kommunikationen. Die Spannbreite reicht hierbei von der kostenfrei nutzbaren statischen Präsentation des Güter- und Dienstleistungsangebots im Stile eines Katalogs über spezialisierte Marktplätze mit Angebots- und Nachfragefunktionalitäten bis hin zu Informationsdienstleistungen welche Zugriff auf die datenhaltenden Systeme des Geschäftspartners gewähren.

Die umfassende Betrachtung der zuvor ausgeklammerten Kommunikation mit potentiellen und bestehenden Mitarbeitern konstituiert die dritte Klasse der e-Businessanwendungen, welche auf die unternehmensinterne Kommunikation mit den Mitarbeitern fokussieren. Dieser Teilbereich wird mit dem Begriff *Business-to-Employee* (B2E) belegt.
Dieser Sparte werden alle elektronischen Informationsangebote an den Mitarbeiter, wie Auskunft über den aktuellen Gleitzeitstand, Adressstamm- sowie Gehaltsdaten, zugeordnet.

1.2 Relevante Techniken und ihre Einordnung

Orthogonal zu den drei Anwendungsdimensionen verdient die ebenfalls in Abbildung 1 dargestellte Realisierungstechnik Betrachtung.
Hierunter fallen gemäß Definition 1 alle sog. *Internettechniken*.

Dieser, in der Praxis nicht klar definiert und trennscharf gebrauchte Begriff umfaßt sowohl die Internetbasistechniken zur Datendarstellung und -übertragung als auch verschiedene Techniken zur Realisierung von Anwendungen, die über das Internet angesprochen und benutzt werden können.

Im wesentlichen zielen die eingesetzten Techniken auf die Lösung spezifischer Problemstellungen. Tabelle 1 stellt die im Rahmen der Vorlesung behandelten Techniken nebst den durch sie betrachteten Problemgebieten und einer Kurzcharakteristik zusammen.

Tabelle 1: Techniken: Einordnung und Kurzcharakterisierung

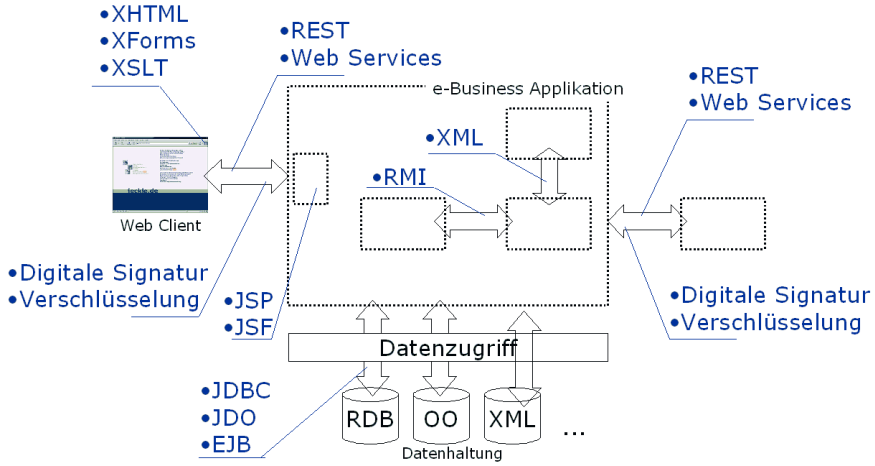
Problemdomäne	Technik	Charakteristik
Datendarstellung und -zugriff	XML	Generische Auszeichnungssprache zur Darstellung beliebiger Daten.
	XML-Namensräume	Syntaxmechanismus zur Unterscheidung von XML-Vokabularen.
	XML-Schema	Grammatiksprache zur Formulierung von XML-Vokabularen.
	XPath	Lokatorsprache zur Identikation von Knotenmengen in XML-Dokumenten.
Datenbankzugriff	JDBC	Durch SUN erarbeiteter Ansatz für den Zugriff auf tabellenartige Datenquellen. Zumeist für den Zugriff auf relationale Datenbanken benutzt.
	JDO	Mechanismus zur transparenten Persistierung von Java-Objekten in verschiedenen Datenspeichern.
	EJB	Durch SUN erarbeitete Komponententechnik. Hauptfocus in dieser Veranstaltung: Realisierung von Peristenz durch *Entity Beans*.
Funktionsintegration	JMS	Durch SUN für Java entwickelte Schnittstelle zur Verarbeitung asynchroner Nachrichten.
	RMI	Durch SUN für Java adaptierte Variante entfernter Funktionsaufrufe.
	REST	Interpretations- und Nutzungsvariante des HTTP-Protokolls zur Realisierung einfacher Web-Dienste.
Web Services	Ansatz zur Bereitstellung von Funktionalität über das Web mittels Nachrichtenaustausch und entfernter Funktionsaufrufe.	
Präsentationsaspekte	XHTML und XForms	Bekannteste Hypertextsprache und Ansatz zur Realisierung einfacher Web-basierter Eingabeoberflächen.
	JSP	Durch SUN erarbeiteter Ansatz zur dynamischen serverseitigen Erzeugung von Webseiten.

	JSF	Durch SUN erarbeiteter Ansatz zur vereinfachten Erstellung von GUI-basierten Web-Dialoganwendungen.
	XSLT	W3C-Standard zur Transformation von XML-Inhalten.
Sicherheitsaspekte	Schlüsselaustausch	Erzeugung und Verteilung geheimer und öffentlicher Daten, die den Zugriff auf gesicherte Daten gestatten.
	Leitungssicherheit	Bereitstellung transparenter Verbindungssicherung im Internet.
	Digitale Signatur	Sicherung von Datenkonsistenz, Glaubwürdigkeits des Ursprungs, Verbindlichkeit und Berechtigung.
	Verschlüsselung	Sicherung von Vertraulichkeit.

1.3 Architektur moderner e-Business Applikationen

Abbildung 2 ordnet die zuvor eingeführten Techniken in ein Architekturmodell für e-Business Applikationen ein.

Abbildung 2: *Architektur moderner e-Business Applikationen*



(click on image to enlarge!)

Das Architekturmodell zeigt die im Rahmen der Vorlesung behandelten Techniken als Bestandteil einer hypothetischen Architektur. Sie zeigt die bevorzugten Einsatzbereiche der Einzeltechniken und gibt damit bereits einen Ausblick auf die gegenwärtig in der Praxis etablierte Pragmatik.
Besonders deutlich wird dies anhand der dargestellten Positionierung des Remote-Method-Invocation-Mechanismus. Zwar kann dieser grundsätzlich auch zur systemübergreifenden Kommunikation herangezogen werden. Jedoch wird RMI aktuell vorwiegend für die Realisierung systeminterner Kommunikationsbeziehungen, beispielsweise innerhalb J2EE-basierter Applikationsserver, herangezogen. Dies liegt in zwei Grundfaktoren begründet. Zum einen ist nur ein Teil der verfügbaren e-Business-Systeme unter Nutzung der Programmiersprache Java realisiert, worauf die RMI-Anwendbarkeit faktisch beschränkt ist. Zum anderen ist der RMI inhärent zugrundeliegende Zugriff auf binäre Applikationsschnittstellen unter Sicherheitsrestriktionen als problematisch anzusehen.

2 Datendarstellung und -zugriff

2.1 Extensible Markup Language -- Strukturelle Grundkonzepte

Im Grunde besitzt die Geschichte der eXtensible Markup Language zwei Anfänge. Einerseits stellt XML die evolutionäre Fortentwicklung existierender generischer Auszeichnungssprachen dar; andererseits sind die

Hintergründe der Sprache XML so eng mit dem Aufkommen des World Wide Webs (WWW) verwoben, daß die Geschichte auch hier ihren Anfang nehmen könnte...

Der chronologischen Ordnung folgend sei zunächst die Entwicklung aus der Idee des *Hypertext* aufgerissen.
Die ersten Ideen zum Konzept des Hypertexts, als Plan zur Überwindung der Beschränkungen und Unzulänglichkeiten des klassischen textbasierten Publikationsmediums Papier, datieren zurück bis in die 1950er Jahre. Sie postulieren neben der nichtsequentiellen Organisation des Mediums auch zentrale Begriffe wie *Knoten*, *Link*, *Anker* und *Netz*. Ziel dieser Überlegungen war es, den auszudrückenden Inhalt von editorieller- und Präsentationsinformation wie Seitenzahlen, Fußnoten, Paginierung usw. zu trennen. Durch die nichtlineare Organisation soll es dem Leser freigestellt werden, auf welchen Pfaden er sich durch das Dokument bewegt.

Zur Realisierung dieser Bemühungen wird das Dokument mit weiteren Informationen angereichert, die jedoch für den Leser unsichtbar bleiben. Dieser Gedanke reicht zurück bis in die Anfänge des Buchdrucks. Dort sind formatierungsorientierte Auszeichnungssymbole, etwa für Fettdruck oder Unterstreichung, seit jeher bekannt. Vor dem Aufkommen der *what you see is what you get* Textverarbeitungssysteme waren diese bildlichen Symbole die einzige Möglichkeit zur Kommunikation präsentationsorientierter Information an den Schriftsetzer und Drucker.
Jedem Schüler ist bereits ein weiteres Beispiel einer editoriellen Auszeichnungssprache bekannt: Die graphischen Korrekturzeichen der Deutschlehrer. Auch sie liefern Informationen über den Inhalt, die nicht Bestandteil des Dokuments sind.

Voraussetzung für die angestrebte Flexibilisierung der Struktur eines Textes ist eine -- wie auch immer geartete -- technische Unterstützung. Seit den 60er Jahren wurden hierfür die aufkommenden elektronischen Rechenanlagen herangezogen. Eine der ersten Aktivitäten hierzu ist das von Ted Nelson initiierte (inzwischen legendäre) Xanadu-Projekt.

Zunächst erforderte die maschinelle Verarbeitung die Überarbeitung des Auszeichnungssymbolvorrates. Dies wurde notwendig, da eingesetzte Technik keine Unterstützung der alt-hergebrachten graphischen Auszeichungssymbole bot.
In einem ersten Entwicklungsschritt wurden daher die vormalig bildhaften Zeichen durch textuelle Pendants ersetzt und verallgemeinert. Beispielsweise: `Überschrift` zur inhaltlichen Kennzeichnung einer entsprechenden Textzeile.
Mit diesem Schritt erfolgte auch der Übergang zur formatierungsunabhängigen Auszeichnung, die bewußt auf die Beschreibung des späteren visuellen Aussehens der Information zugunsten einer neutralen deskriptiven Beschreibung der Semantik verzichtete.

In den 60er und 70er Jahren werden verschiedene Weiterentwicklungen der generischen Auszeichnungssprachen betrieben; u.a. bei der IBM durch das Team um Goldfarb, Mosher und Loire. Sie stellen 1969 unter dem Namen *Generalized Markup Language* einen Sprachvorschlag zusammen, der in der Folgezeit durch IBM kommerziell vermarktet wird.

Aus den GML-Aktivitäten bei IBM entwickelt sich die internationale Standardisierungsbewegung der *Standard GML* (SGML).
Durch sie wird eine Sprache festgelegt, welche die Definition eigener Sprachen erlaubt; daher auch der Begriff *Metasprache*. SGML bietet somit keinen feststehenden problemspezifischen Sprachumfang an, sondern eine Menge verschiedenster struktureller Konstrukte zur Formulierung von Dokumentgrammatiken.
In der Praxis wird der Einsatz einer mit Hilfe von SGML definierten Sprache oftmals plakativ zum *Einsatz von SGML* verkürzt, obwohl diese Begrifflichkeit lediglich den Erstellungsprozeß der Grammatik bezeichnet.

Mittels SGML definiert Tim Berners-Lee Mitte der 80er Jahre eine eigene Sprache zur vereinfachten Formulierung von Dokumenten, die er *HyperText Markup Language* (HTML) nennt. Hauptbeweggrund seiner Aktivitäten ist der Versuch den Dokumentenaustausch am Europäischen Kernforschungszentrum CERN rechnergestützt zu vereinfachen.
Die Eingangs erwähnten zentralen Hypertextkonzepte finden sich bereits in seinem ersten Sprachvorschlag wieder. Zur technischen Realisierung der Verknüpfung zwischen den Dokumenten mittels Ankern und Links definiert er den *Uniform Resource Locator* (URL), eine global eindeutige Adresse für beliebige Inhalte.

Seine Aktivitäten in Genf bilden die Keimzelle des Web.

In der Folgezeit, insbesondere im Zuge der Kommerzialisierung des Word Wide Web, entstehen verschiedene Revisionen der ursprünglichen HTML. Einige der Erweiterungen werden durch die beiden großen Web Browser Hersteller Microsoft und Netscape proprietär vorgenommen, um ihre Position am Markt zu stärken.
In der Konsequenz entstehen während des oft apostrophierten *browser war* teilweise inkompatible HTML-Dialekte. (Man denke nur an die Tags: `marquee` (nur Microsoft Internet Explorer) oder `layer` (nur Netscape Navigator))
Darüberhinaus entwickelt sich HTML zunehmend von einer Präsentations-orientierten Auszeichnungssprache zu einer semantischen. Dies bedeutet: während HTML in der ersten Grundform

zunächst überwiegend Elemente bot, durch die die Präsentation der Inhalte am Bildschirm festgelegt wurde (Beispiele: `b` für Fettdruck, `u` für Unterstreichungen oder `i` für Kursivschreibung), wurden später zunehmend semantische Elemente eingeführt. Durch sie wird die Bedeutung der ausgezeichneten Information ausgedrückt (Beispiele hierfür: `acronym` zur Kennzeichnung von Abkürzungen, `address` für Adressen oder `strong` zur besonderen Betonung einer Textpassage).

So wünschenswert die sukzessive Umgestaltung der HTML an die veränderten Bedürfnisse war, so aussichtslos waren die Bemühungen dennoch. Während bei den Präsentations-orientierten Elementen zunehmend Vollständigkeit hinsichtlich der Anwenderwünsche erzielt werden konnte, offenbaren sich die bisher erfolgten semantischen Erweiterungen als permanent inadäquat.
Letztlich war der Versuch, durch Standardisierung, semantische Erweiterungen in HTML einzubringen in doppelter Hinsicht zum Scheitern verurteilt:
1. birgt der Ansatz die Gefahr, die Elementmenge in unbekannte Größen zu erweitern
2. muß die Semantik jedes Tags definiert, abgestimmt und verabschiedet werden.

Aus diesen Gründen wurde seitens des W3C nach einer tragfähigeren Lösung gesucht. Unter Rückgriff auf die HTML-Wurzeln (als Anwendung der Metasprache SGML) wurde das Projekt *SGML for the Web* initiiert. Der letztendlich verabschiedete Vorschlag zur *eXtensible Markup Language* (XML) bildet konzeptionell eine Untermenge der Sprachmöglichkeiten von SGML. Konsequenterweise ist jedes XML-Dokument auch ein gültiges SGML-Dokument.

Die Abweichung zu SGML wird besonders aus den Entwicklungszielen für XML deutlich:

1. Einfache Nutzung im Internet.
   In Abkehr von den Hauptnutzung SGMLs als offline Dokumentationsformat wird die Untermengenbildung *XML* für die primäre Nutzung im Internet vorgenommen.
2. Unterstützung eines breiten Anwendungsspektrums.
   Auch hier soll die Untermengenbildung das Einsatzspektrum über die Hauptnutzung SGMLs als Format der technischen Dokumentation hinaus befördern.
3. SGML Kompatibilität.
   XML bildet eine echte Untermenge des ISO-Standards SGML, durch diesen Schritt kann jedes XML-Dokument auch als gültiges SGML-Dokument interpretiert und durch die entsprechenden SGML-Werkzeuge verarbeitet werden.
4. Einfache Applikationsentwicklung.
   Die Untermengenbildung wird im Hinblick auf eine gegenüber SGML deutlich vereinfachte Entwicklung von XML verarbeitenden Applikationen vorgenommen.
5. Minimierung optionaler Sprachmerkmale -- Idealerweise gleich Null.
   Auch dieses Ziel ist im Hinblick auf eine vereinfachte Applikationsentwicklung, aber auch eine einfachere Benutzbarkeit durch Menschen auf dem Wege der Komplexitätsreduktion zu interpretieren.
6. Lesbarkeit.
   Das entstehende Textformat soll für Menschen und Maschinen gleichermaßen les- und verstehbar sein.
7. Kompakte Spezifikation.
   Die erstehende XML-Spezifikation sollte deutlich weniger Umfang aufweisen als der SGML-Vorgängerstandard. Letztlich konnte die reine Seitenzahl von über 600 Seiten für die SGML-Spezifikation auf ungefähr 30 Seiten für XML reduziert werden.
8. Formaler und präziser Sprachentwurf.
   Um die schnelle Akzeptanz seitens der Anwender zu forcieren erachteten die Mitglieder der XML-Arbeitsgruppe die schnelle Verfügbarkeit von XML-Werkzeugen für essentiell. Aus diesem Grunde sollte der XML-Sprachentwurf möglichst leicht und eindeutig in XML-Werkzeuge zu implementieren sein.
9. Leichte Dokumenterstellung.
   Die Erstellung von korrekten XML-Dokumenten sollte idealerweise so einfach sein, daß hierfür keine speziellen Werkzeuge benötigt werden.
10. Nicht notwendigerweise knappes Markup.
    Kompaktheit und Effizienz hinsichtlich des Volumens eines XML-Dokuments war zu keinem Zeitpunkt eines der Hauptentwicklungsziele. Auf der Basis des XML-Information Sets ist es jedoch möglich beliebig kompakte Binärformate identischer Mächtigkeit zur die in der XML-Spezifikation vorgestellten Textnotation zu definieren.

XML stellt jedoch keine echte semantische Auszeichnungssprache dar, da durch die Metasprache lediglich eine Möglichkeit zur Formulierung eigener Syntax gegeben ist. Die Bedeutung der Elemente bleibt jedoch unberücksichtigt, und kann mittels XML nicht ausgedrückt werden.

Tabelle 2: Einige chronologische Eckdaten

Jahr	Ereignis
1945	Vannevar Bush diskutiert in seinem Artikel As We May Think ein persönliches Informationssystem mit Kommunikationsmöglichkeiten und Zugriff auf Bücher, Tonaufnahmen, etc. unter dem Namen *Memex*.
1967	William Tunnicliffe (Chairman des Graphic Communications Association (GCA) Composition Committee) schlägt aus seinen Erfahrungen bei der wiederholten Erstellung von Telephonkatalogen (*yellow pages*) vor, häufig auftretende strukturelle Elemente zu standardisieren.
September 1967	William Tunnicliffe (Vorsitzender der Graphic Communication Association) spricht sich auf einer Konferenz des *Printing Office* der Regierung von Kanada für die Separierung von Inhalt und Format aus.
Ende der 1960er Jahre	Stanley Rice, ein New Yorker Schriftsetzer, schlägt *editorial structure tags* vor. Der CGA-Direktor Norman Scharpf initiiert das Projekt *GenCode*.
1969	Charles Goldfarb, Edward Mosher und Raymond Lorie entwickeln bei der IBM die *Generalized Markup Language* (GML). Anwendungshintergrund war ein Projekt zur Integration von Informationssystemen für Anwaltskanzleien.
1970	Goldfarb formuliert zwei Grundprinzipien generalisierter Auszeichungssprachen: 1) Auszeichnungssprachen beschreiben die Dokumentstruktur, nicht die physischen Charakteristika wie Präsentation 2) Die Struktur der Auszeichnungssprache soll so gewählt sein, daß sie sowohl von Menschen als auch Maschinen interpretiert werden kann
1978	ANSI ruft *Computer Languages for the Processing of Text*-Komitee ins Leben. Ziel ist die Weiterentwicklung der GML zu einem nationalen US-Standard.
1980	• ANSI veröffentlicht ersten Entwurf einer standardisierten GML (SGML). • Tim Berners-Lee tritt seine Arbeit am Europäischen Kernforschungszentrum CERN an. Dort entwickelt er in der Folgezeit die (niemals veröffentlichte) Hypertextanwendung *Enquire*.
1983	Der International Revenue Service (IRS) und das US Verteidigungsministerium (DoD) übernehmen den sechsten Entwurf zur SGML (auch bekannt als GCA 101-1983).
1984	Die SGML-Arbeitsgruppe nimmt unter Schirmherrschaft der International Standardization Organization (ISO) als ISO/IEC JTCI/SC18/WG8 ihre Arbeit auf. Goldfarb dient als *technical leader* der ISO-Gruppe, sowie dem umorganisierten ANSI-Komitee X3V1.8.
1985	Norm-Entwurf zu SGML veröffentlicht.
15. Oktober 1986	ISO verabschiedet SGML als ISO 8879:1986.
März 1989	Berners-Lee schlägt mit dem Dokument *Information Management: A Proposal* ein SGML-basiertes Hypertext-System zum Informationsaustausch vor.
1990	Am Weihnachtstag nimmt das World Wide Web seinen Betrieb mit zwei Maschinen am CERN auf. Die notwendigen Implementierungen von HTML, HTTP und URL erfolgten durch Berners-Lee. Die erste WWW-Verbindung wird zwischen Berners-Lees Workstation und Robert Cailliaus' NeXT-Rechner aufgebaut. Ein Screenshot des ersten Web-Browsers NeXTStep-Implementierung des Browsers
1991	Beginn der turnusmäßigen Überarbeitungsphase von ISO 8879.
3. November 1992	Erster Entwurf zu HTML
Juni 1993	Einreichung des ersten HTML Entwurfs bei IETF.
Oktober 1994	Gründung World Wide Web Consortium
14. November 1996	Erster Entwurf zu XML vorgestellt
14. Januar 1997	Verabschiedung der HTML v3.2
1998	W3C gibt die erste Version von XML als Recommendation frei.
2000	• W3C gibt XHTML v1.0 -- die Reformulierung von HTML v4.01 zu einer XML-Anwendung -- frei. • W3C verabschiedet XML 2nd edition; sie integriert u.a. die XML Namespaces und behebt einige editorielle Fehler.
2. Mai 2001	Das W3C verabschiedet den XML Schema-Standard. Er geht an vielen Stellen deutlich über die ererbten SGML-Möglichkeiten hinaus, und markiert den Übergang von Präsentations-orientierten Strukturen hin zu Datenstrukturen.

Zum Abschluß dieser Einführung seinen die zehn Punkte zusammengestellt und kommentiert, die durch das World Wide Web Consortium als plakative Kurzcharakterisierung von XML veröffentlicht wurden:

1. XML steht für strukturierte Daten.
   Diese Aussage betont die Rolle von XML als Sprache um Sprachen zu erzeugen. Nicht XML wird innerhalb verschiedenster Applikationen direkt verarbeitet, sondern XML basierte Formate. So steht nicht die XML selbst für all diese Anwendungsdomänen, sondern die jeweiligen problemspezifischen XML-basierten Sprachen. XML selbst dient lediglich der Strukturierung der verschiedensten darzustellenden Daten.
   Gleichzeitig rückt durch Aussage die Rolle der XML als Datenformat in den Vordergrund und läßt so die Weiterentwicklung gegenüber den präsentationsorientierten Vorläufern deutlich werden.
   Die Vorlesungskapitel Strukturelle Grundkonzepte und XML Schemasprachen vermitteln einen Eindruck dieses Wandels und dokumentieren die Grundlagen des gegenwärtigen datenorientierten Einsatzes der XML.
2. XML sieht ein wenig wie HTML aus.
   Diese Aussage soll offenkundig einerseits den bisherigen HTML-verwendenden Web-Autoren den Einstieg in die XML schmackhaft werden lassen. Dennoch führt sie ein wenig von der Grundidee XMLs als generischer Auszeichnungssprache für beliebigste Anwendungen weg, indem sie den Blick auf HTML focussiert.
   Die -- im Grunde der Verwandschaft zu SGML geschuldete -- offensichtliche syntaktische Ähnlichkeit zu HTML wird bereits bei der Betrachtung der strukturellen Grundkonzepte deutlich.
3. XML ist Text, aber nicht zum Lesen.
   XML-Dokumente können sicherlich im wörtlichen Sinne „gelesen“ werden ... Die Aussage zielt jedoch auf den intendierten Einsatzzweck von XML: der Darstellung von Daten für den Austausch zwischen Maschinen. Unbenommen dessen kann XML selbstverständlich auch von Menschen gelesen und verstanden werden, wenngleich dies bei umfangreicheren XML-Dokumenten durchaus mühsam werden kann.
   Aufschluß über die textuelle Natur XMLs, insbesondere im Hinblick auf die Verwendung unterschiedlicher Alphabete, liefert das Kapitel strukturelle Grundkonzepte.
4. XML ist vom Design her ausführlich.
   Hiermit wird versucht dem häufig geäußerten Kritikpunkt der Platzzunahme XML-codierter Inhalte gegenüber klassischen Darstellungsweisen etwas pauschal entkräftend entgegenzutreten. Sicherlich geht das W3C in dieser Aussage nicht fehl, wenn die Entwicklung der Netzwerkbandbreiten, der CPU-Leistung und der Speicherkapazitäten berücksichtigt. Andererseits ist die Aufblähung der XML-formatierten Inhalte im Vergleich zu optimierten Binärformaten nicht von der Hand zu weisen, wird jedoch durch die mit der Verwendung von XML einhergehenden Vorteile mehr als ausgeglichen.
   Einen ersten Eindruck der Natur XML-codierter Inhalte liefert das Kapitel strukturelle Grundkonzepte. Dort finden sich auch Ansätze die bekannte XML-Syntax kompaktifiziert darzustellen ohne die Vorteile der generischen Auszeichnungssprache aufgeben zu müssen.
5. XML ist eine Familie von Techniken.
   Eine Aussage, die durch alle drei Kapitel der Vorlesung unterstrichen wird, die deutlich zeigen, daß XML nicht als isolierte Idee oder Technik anzusehen ist -- sondern erst im Zusammenspiel mit anderen XML-Standards und eingebettet in Applikationen und Infrastrukturen -- seine volle Wirkungsmächtigkeit entfalten kann.
6. XML ist neu, aber nicht so neu.
   Diese Bezugnahme soll nochmals unterstreichen, daß XML keineswegs den Anspruch erhebt eine vollkommen neue technische Errungenschaft zu sein, sondern vielfach bekanntes und erprobtes aus der Informatik wiederverwendet und im neuen Verwendungskontext weiterentwickelt.
   Diese Aussage wird durch die in den einzelnen Kapiteln dargebotenen Rückbezüge auf bereits bekannte Techniken und Lösungsformen untermauert.
7. XML überführt HTML in XHTML.
   Diese Aussage greift nochmals die Beziehung zwischen XML und HTML auf. Diesmal soll die Rolle von XML im Bezug auf die Weiterentwicklung von HTML zum XML-basierten Vokabular *XHTML* unterstrichen werden. So löst XML die Abhängigkeit zwischen SGML und HTML auf und reformuliert HTML auf der Basis von XML.
   Das Kapitel XHTML führt kurz in die Entwicklung der neuen HTML-Varianten auf Basis der XML ein und skizziert die vorgenommen Änderungen und zukünftige Erweiterungen dieser Hypertextsprache.
8. XML ist modular.
   Hierdurch wird unterstrichen, daß XML kein in sich geschlossenes monolithisches Gebilde darstellt, sondern einzelne Vertreter aus der Familie der XML-Sprachen wahlfrei zur Lösung konkreter Probleme herangezogen werden können. Ebenso wird die Sprachfamilie beständig an verschiedensten Stellen unabhängig voneinander weiterentwickelt, ohne einer zentralen Koordination zu bedürfen.
9. XML ist die Basis für RDF und das Semantic Web.
   Grundidee des Semantic Web ist die Weiterentwicklung des sichtbaren XHTML-basierten Webs unter Nutzung seiner datenorientierten Ergänzung XML zu einem Netz von Sinnzusammenhängen.
10. XML ist lizenzfrei, plattform- und herstellerunabhängig, und gut unterstützt.
    XML ist eine durch das World Wide Web Consortium herausgegebene Spezifikation, die kostenfrei über das Web bezogen werden kann und durch Interessierte ohne weitere Lizenzkosten in eigenen kommerziellen Produkten verwendet werden. Durch den Standardisierungsprozeß innerhalb des World Wide Web Consortiums wird sichergestellt, daß keine Ausführungsplattform bevorzugt wird und gleichzeitig keine Nachteile für Andere entstehen. Dies wird durch die herstellerunabhängige Organisation des Gremiums versucht zu garantieren, in dem zwar Hersteller Mitglied werden können,

die technischen Entscheidungen jedoch Arbeitsgruppen obliegen, die nicht durch eine Firma dominiert werden können.

Web-Referenzen 1: Vertiefende Informationen

- Artikel in der Online-Ausgabe des *Economist* über Ted Nelson -- *The Babbage of the web*
- COT1800 Public Networks, Lecture 8, Standard Generalised Markup Language
- Brief History of Document Markup
- XML, Element Types, DTDs, and All That
- Clark, J.: *Comparison of SGML and XML*

Web-Referenzen 2: Weiterführende Links

- Browser Timelines
- Browser Emulator

Definition 2: XML-Sprache

Eine Anwendung der Extensible Markup Language. Ein Vokabular, das aus Symbolen und der ihnen zugewiesenen Bedeutung (Semantik) gebildet wird, ergänzt um Regeln (grammatikalische Struktur und Gültigkeitsregeln für den Inhalt (z.B. Datentypen)) zur Kombination der Vokabularelemente.
Anwendungen einer so neu geschaffenen XML-Sprache *L* werden als XML-Dokumente, auch: *L*-Dokumente, bezeichnet.

Strukturelle Grundkonzepte

Die grundlegende XML-Syntax ist in der namensgebenden W3C-Recommendation der Extensible Markup Language definiert. Die Semantik der Metasprache wird hingegen durch den W3C-Standard des XML Information Set festgelegt.
Diese Spezifikationen beinhalten die grundlegenden Definitionen hinsichtlich Terminologie und Beziehung der verschiedenen möglichen Elemente eines XML-Dokuments. Im vorliegenden Teilkapitel werden beide Sprachaspekte grundlegend eingeführt und ein erstes Verständnis der XML vermittelt. Dabei wird in Form von Ausblicken auf nachfolgende Abschnitte der Bogen zu Grammatikdefinitionssprachen und weiterführenden Konzepten wie Namensräumen gespannt.
Zum leichteren Verständnis sind die aus der offiziellen Spezifikationen entnommenen formalen Grammatikdefinitionen der EBNF-Notation durch vereinfachte graphische Strukturdarstellungen ergänzt.

Definition 3: XML Dokument

Ein XML-Dokument ist ein Datenstrom (der nicht zwingend als Datei vorliegen muß), welcher den Strukturierungsprinzipien der eXtensible Markup Language genügt.

Definition 4: XML Information Set

Die Spezifikation des XML Information Sets definiert die Semantik der Metasprache XML, d.h. ihre zentralen Begriffe.
Gleichzeitig setzt es diese Begriffe in Beziehung und definiert so syntaxunabhängig die Struktur eines XML-Dokumentes.

Ausgehend von der Allgemeinheit der Aussage aus Definition 1 folgt, daß der Infoset neben seinem theoretischen Wert als Semantikdefinition zur XML auch zur Formulierung der Datenstrukturen, welche innerhalb eines XML-Prozessors vorliegen müssen, um beliebige XML-Dokumente verarbeiten zu können, herangezogen werden kann.
Daher läßt sich ein XML-Prozessor definieren als:

Definition 5: XML-Prozessor

Ein XML-Prozessor ist eine maschinelle Komponente (typischerweise: Software), die zum Lesen, Speichern und Verarbeiten eines XML-Dokuments eingesetzt wird.
Er erlaubt Zugriff auf den Inhalt und die Struktur des XML-Dokuments.

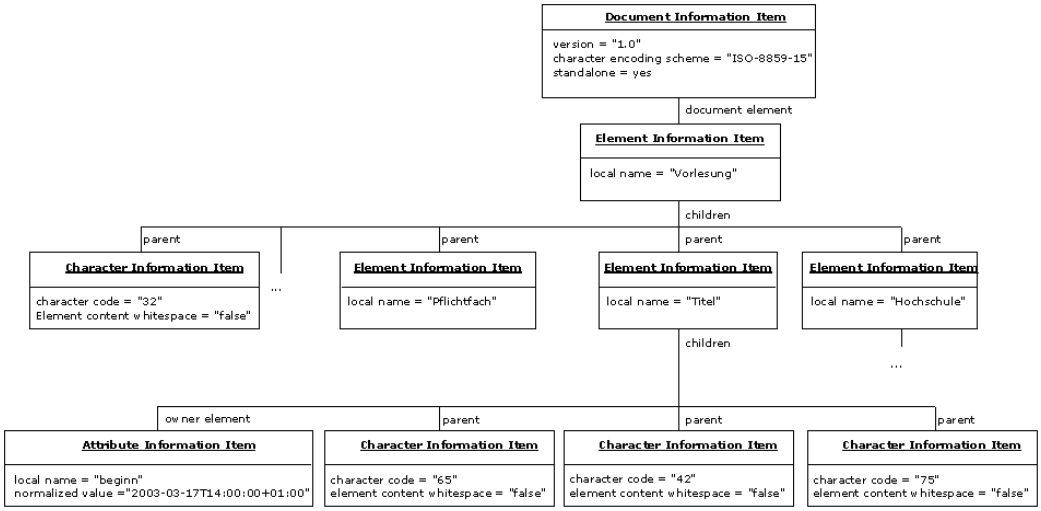
Die XML-Spezifikation faßt den XML-Prozessorbegriff etwas enger und beschränkt ihn lediglich auf Software-Module, die XML-Dokumente lesend verarbeiten. Konzeptionell spricht jedoch nichts gegen eine Umsetzung in Hardware, beispielsweise im Kontext eingebetter Systeme etc. (In XML-Spezifikation nachschlagen)
Ferner nimmt die XML-Spezifikation an, ein Prozessor operiere nicht eigenständig, sondern im integrierten Zusammenspiel mit einer Applikation.

Beispiel 1: Ein erstes XML-Dokument

```
(1)<?xml version="1.0" encoding="ISO-8859-15" standalone="yes"?>
(2)<Vorlesung>
(3)   <Pflichtfach/>
(4)   SS2003
(5)   <Titel beginn="2003-03-17T14:00:00+01:00">eBusiness-Engineering</Titel>
(6)   <Hochschule>Fachhochschule Furtwangen</Hochschule>
(7)   <Praktikum>Kein Übungsbetrieb</Praktikum>
(8)</Vorlesung>
```

Download des Beispiels

Das Beispiel zeigt ein erstes einfaches XML-Dokument, welches bereits die häufigst verwendeten Sprachelemente der XML versammelt.
Jedem XML-Dokument entspricht genau ein Information Set, der alle Informationselemente des Dokuments in Form einer Baumstruktur beinhaltet. Die nachfolgende Abbildung zeigt den Information Set des Beispiels in der Notation eines UML-Klassendiagramms. Dabei sind die einzelnen Knoten des Information Sets als Objekte (Klassensymbole mit unterstrichenem Klassennamen) und die Eigenschaften der Knoten als Attributwerte dargestellt.



Document Information Item

Jedes Information Set besteht genau aus einem *Document Information Item*. Dieses stellt den äußeren Rahmen des XML-Dokuments dar. Es beinhaltet dokumentbezogene Informationen, wie die verwendete XML-Version und das gewählte Codierungsschema innerhalb des Unicode-Systems.
Das Document Information Item enthält daher u.a. die Informationen des XML-Dokumentprologs in der erste Zeile jedes Dokuments. Das durch die öffnende Winkelklammer und ein Fragezeichen eingeleitete Konstrukt ist in der ersten Zeile des Beispiels 1 dargestellt. Innerhalb des Prologs findet sich die Zeichenkette `xml`, sowie die Bezeichner `version` und `encoding`. Beiden ist ein durch doppelte Hochkommata umschlossener Wert nachgestellt, `1.0` für `version`, bzw. `ISO-8859-15` für `encoding`. Beendet wird der Prolog wiederum durch ein Fragezeichen und die schließende Winkelklammer. Wird auf die Angabe des optionalen Prologs im Dokument verzichtet, so sind die daraus ableitbaren Angaben im Document Information Item nicht gesetzt.

Als weitere Eigenschaften verfügt jedes Document Information Item über eine geordnete Liste von Kindknoten. Darin ist genau ein Element Information Item enthalten, welches den Startknoten des XML-Dokuments verkörpert. Wegen seiner hervorgehobenen Bedeutung als Wurzel des Dokumentbaumes wird dieser Knoten auch als `Document Element` bezeichnet.
Zusätzlich kann die Liste Elemente vom Typ Processing Instruction Information Item enthalten. Sie dienen der Darstellung von Verarbeitungsanweisungen, die durch den XML-Prozessor interpretiert werden.
Im Kopfbereich vor `Document Element` plazierte XML-Kommentare werden durch Comment Information Items innerhalb der `children`-Liste dargestellt.

Zusammengefaßt enthält das Document Information Item folgende Informationen:

- **Kindknoten**: In der Reihenfolge des Auftretens im Dokument geordnete Liste. Sie enthält mindestens ein `Element Information Item`, welches in der Rolle des `Document Items` fungiert. Ferner je ein Element des Typs `Processing Instruction Information Item` für jede Processing Instruction die außerhalb des Wurzelements definiert ist und jeweils ein `Comment Information Item` zu jedem definierten Kommentar.
- **Document Element**: Ein Element des Typs `Element Information Item`, das auf den Wurzelknoten des Dokuments verweist.
- **Basis URI**: Lokation, falls bekannt, des XML-Dokuments in Form eines *Uniform Resource Identifiers* (URI) gemäß IETF RFC 2396.
- **Character Encoding Scheme**: Der Name der gewählten Codetabelle aus dem Unicode-Standard.
- **Standalone**: Legt -- als Boole'scher Wert (Zugelassene Belegungen: *yes*, *no*) -- fest, ob die im Dokument

gespeicherten Daten durch eine sie verarbeitende Applikation interpretiert und somit ergänzt oder verändert werden.
Häufigste Form dieses Interpretationsvorganges ist die Präsenz einer expliziten Grammatik in Form einer *Document Type Definition*, welche Gültigkeitsregeln für eine Familie von Dokumenten formuliert. Konsequenterweise bedeutet daher die Belegung mit *no*, daß die gespeicherten Daten entweder durch die Applikation interpretiert werden, dies ist beispielsweise bei der Auflösung von Entitäten der Fall, oder durch eine *DOCTYPE*-Deklaration ein Verweis auf die externe Dokumentgrammatik erfolgt. Die Angabe von *standalone* ist optional. Fehlt sie und ist gleichzeitig eine DOCTYPE-Deklaration im Dokument gegeben, so wird *standalone="no"* angenommen.
Im Beispiel ist die standalone-Deklaration auf *yes* gesetzt, d.h. es existiert explizit keine Dokumentgrammatik. (In XML-Spezifikation nachschlagen)

- **Version**: Die eingesetzte XML-Version. Dieser Wert wird aus dem Dokumentprolog übernommen.

Wie auch im Beispieldokument, bildet die erste Zeile den sog. *Prolog* eines jeden XML-Dokuments (In XML-Spezifikation nachschlagen) . Die Angabe der Version ist zwingend und derzeit auf die Konstante `1.0` fixiert. Die aktuelle XML-Spezifikation sieht als gültige Belegung der Versionsangabe ausschließlich die Zeichenkette `1.0` vor. Zukünftigen Weiterentwicklungen ist es jedoch freigestellt auch andere Revisionskennungen zu vergeben.
`encoding` leitet das zweite Namen-Wert-Paar ein. Die Deklaration ist innerhalb des Prologs optional, und kann daher auch unterbleiben. Die Zeichenkette der Encodingdeklaration benennt das Codierungsschema, welches für das so gekennzeichnete Dokument verwendet wurde. Es definiert den Satz der innerhalb des Dokumentes zugelassenen Zeichen fest.
Gemäß Produktion 22 der XML-Syntaxdefinition ist der gesamte Prolog optional.

Die Encoding-Deklaration hat folgendes Aussehen (In XML-Spezifikation nachschlagen) :

```
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )
[81] EncName      ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
[3]  S            ::= (#x20 | #x9 | #xD | #xA)+
[25] Eq           ::= S? '=' S?
```

Die Festlegung der Produktion 80, sowie die der Produktion 23, stellt heraus, daß sich die Encodingdeklaration nicht auf die Prologzeile selbst auswirkt. Hier sind die beiden Zeichenketten `xml` und `encoding` in der Codierung UTF-8 oder UTF-16 Vorschrift.

Als Belegungen des `Encoding Namens` (`EncName`) sind beliebige Zeichensätze zugelassen. Der XML-Standard empfiehlt jedoch lediglich auf die durch die *Internet Assigned Numbers Authority* verwalteten zurückzugreifen (Dokument: Official Names for Character Sets) (In XML-Spezifikation nachschlagen) .
Die häufigsten praktisch eingesetzten Deklarationen sind die der ISO-8859 (*extended ASCII*)-Familie, sowie die der Unicode- und ISO-10646-Standards.
Die verschiedenen Abschnitte der ISO-8859 Familie werden als `ISO-8851-`*n* ausgedrückt, wobei *n* die Nummer des Abschnittes des zugehörigen ISO-Dokuments referenziert. Ferner können die durch JIS X-0208-1997 normierten asiatischen Zeichensätze als `ISO-2022-JP`, `Shift_JIS` und `EUC-JP` dargestellt werden.

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes"?>
<kougi>
  <title begin="二千一年三月二十一日三時三十分＋九時">選挙義務講義ＸＭＬ</title>
  <organization>アウグスブルグ大学コンピュータ・サイエンス過分、工学と家政と形成の大学</organization>
</kougi>
```

Unicode stellt einen Industriestandard (entwickelt u.a. durch Apple, HP, IBM, Microsoft und SUN) zur Darstellung verschiedenster Alphabete und graphischer Zeichen dar. Sein zunächst durch 16-Bit codierter Zeichenvorrat bot Raum für 65536 unterschiedliche Symbole.
Die seit 1991 laufenden Unicodebemühungen münden in die ISO-Norm zur Erweiterung des klassischen ASCII-Codes (ISO 646) als ISO-10646 *Universal Multiple-Octet Coded Character Set (UCS)*. Seit 1996 sind beide Standards synchronisiert und werden abgestimmt vorangetrieben.
UCS definiert zwei aufeinander aufbauende Codierungen: UCS-2 (16 Bit Umfang) und UCS-4 (32 Bit). Der bisherige Unicode-Standard ist voll kompatibel zu UCS-2 und durch diesen darstellbar.

Tabelle 3: Verschiedene Codierungen des Zeichens "A"

Codierung	Bitbreite	Binärdarstellung	Größe der Beispieldatei in Byte (ohne Berücksichtigung des XML-Prologs)	Bemerkung zum Meßwert
UTF-7	>= 7	100 0001	263	(encoding="UTF-7")
Extended ASCII, Latin-1 (ISO-8859-1)	8	0100 0001	258	(encoding="ISO-8859-1")

UTF-8	>= 8	0100 0001	259	(encoding="UTF-8") keine Byte Order Mark
UCS-2, Unicode	16	0000 0000 0100 0001	516	(encoding="UCS-2") keine Byte Order Mark
UTF-16 (big endian)	>= 16	0000 0000 0100 0001	516	(encoding="UTF-16") keine Byte Order Mark
UCS-4	32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-8") keine Byte Order Mark
UTF-32	>= 32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-32") keine Byte Order Mark

Die Zeilenumbrüche wurden in allen Fällen durch die Kombination von Wagenrücklauf und Zeilenvorschub ausgedrückt.

Die Tabelle stellt einige Codierungen zur Darstellung des Zeichens *A* zusammen.
Auffallend ist der große Platzbedarf der UCS-2 und -4 Codierungen. Insbesondere bei den „klassischen" ASCII-Symbolen werden hier (u.U. sehr viele) führende Nullbits erzeugt, die in der Konsequenz zu einer deutlichen Vergrößerung der Beispieldatei führen.
Daher wurde mit dem *UCS Transformation Format* (UTF) eine kompaktere Darstellung zum jeweiligen UCS-Set eingeführt. UTF-8 verwendet standardmäßig die ersten acht Bit zur Darstellung der bekannten ASCII-Zeichen

Anmerkung: Inzwischen existiert auch eine „UTF-32" genannte 32-Bit Ausprägung, diese ist jedoch identisch zu UCS-4, mit Ausnahme daß durch UTF-32 „nur" 2^{21}-Zeichen dargestellt werden können.
Die Dateigröße ist daher für das betrachtete Beispiel in dieser Darstellungsweise unverändert zu der des UCS-4-Encodings.

Der Größenunterschied zwischen der UTF-7 codierten Datei und der Latin-1 encodierten erklärt sich aus der Darstellung des Umlautes sowie des +-Zeichens, die beide nicht nicht im klassischen 7-Bit ASCII-Code enthalten ist. So wird *Ü* im Wort *Übungsbetrieb* des Beispieldokumentes durch die die Bytefolge `2B 41 4E 77 2D` dargestellt, während alle übrigen Zeichen durch ein einzelnes Byte ausgedrückt werden können.
UTF-8 ist in der Lage sämtliche Standard-ASCII-Zeichen durch jeweils genau ein Byte auszudrücken, wiederum für den Umlaut muß auf die 16-Bit-Darstellung des UCS-2 zurückgegriffen werden. Daher erhöht sich hier die Dateigröße um ein Byte.
Erwartungsgemäß beträgt der Umfang des UCS-2 codierten Dokuments exakt das Doppelte des 8-Bit Äquivalents der Latin-1-Darstellung.
Dasselbe gilt für die UTF-16-Variante, die für das vorliegende Beispiel unterschiedslos zu UCS-4 verläuft, da keinerlei Zeichen aus UCS-4 im Dokument auftreten.

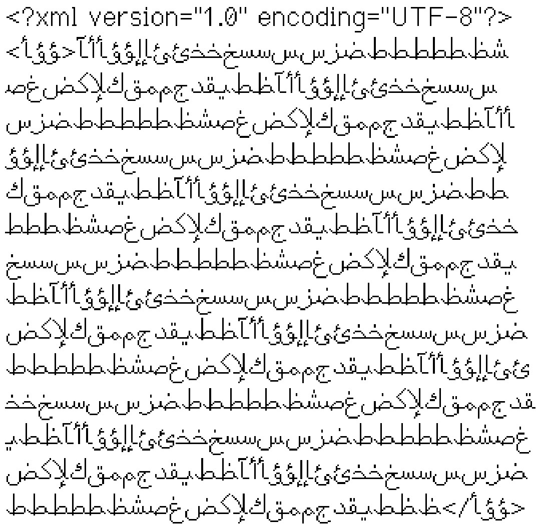
Die nachfolgende Tabelle stellt beispielhaft die Anwendung der UTF-8-Codierung zusammen:

Tabelle 4: UTF-8 Codierung

Unicode-Bereich	Bitbelegung
U-00000000 - U-0000007F:	0xxxxxxx
U-00000080 - U-000007FF:	110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF:	1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF:	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF:	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF:	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Diese Mimik zeigt den Nachteil des UTF-*n*-Encodings deutlich: Die Darstellung nicht *n*-Bit darstellbarer Zeichen benötigt u.U. mehr Bitstellen als im Standard UCS-Code.
So wird beispielsweise das Zeichen mit der größtmöglichen Position (`7FFFFFFF`) in UTF durch sechs Byte encodiert, während UCS dieselbe Information mit den verfügbaren 32-Bit ausdrücken kann. Andererseits „verschwendet" die UCS-Darstellung für die niederwertigen Zeichen Bitstellen durch die führenden Nullen.

In der Praxis gilt es daher für das zu wählende Encoding einen möglichst guten Kompromiß zu finden: Im allgemeinen stellt das UTF-8-Encoding einen solchen dar, soweit überwiegend ASCII-Zeichen, und nur vereinzelt Sonderzeichen (hierzu zählen auch die deutschen Umlaute) eingesetzt werden.
Bei überwiegender Verwendung nicht in acht-Bit ASCII darstellbarer Zeichen (z.B. arabischer, chinesischer, etc.) erhöht die dann aufwendigere UTF-8-Codierung die Datenmenge.
So umfaßt die UTF-16-Darstellung des unten abgebildeten Beispieldokuments, welche in diesem Anwendungsfall identisch zu UCS-2 ist, 966 Bytes, während UTF-8 1299 Byte benötigt.

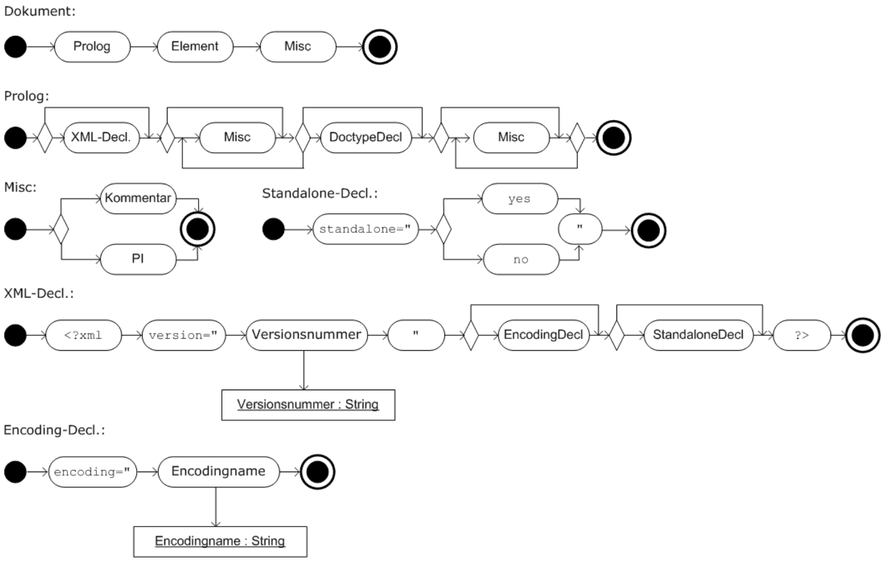


Achtung: Bereits durch die Unterstützung der beiden ISO-Zeichendarstellungen UTF-8 und UTF-16 ist die Konformität zum XML-Standard erfüllt! XML-Prozessorimplementierungen wird nicht abverlangt darüberhinausgehend weitere Darstellungen umzusetzen. (In XML-Spezifikation nachschlagen)

Wie bereits eingangs angemerkt, erklärt die XML-Spezifikation die Encodingdeklaration sowie den gesamten Prolog-Ausdruck als optionales Element (In XML-Spezifikation nachschlagen) .
Als Konsequenz geht dabei (auch) die Angabe des gewählten Encodings verloren.
Daher fordert der Anhang F der XML-Spezifikation *Autodetection of Character Encodings* bei einem von UTF-8 oder -16 abweichendem Codierungsschema die zwingende Angabe der XML-Deklaration (`<?xml ...`) (In XML-Spezifikation nachschlagen) .
Hintergrund dieser Maßnahme ist der Versuch anhand der damit bekannten fünf Zeichen das zugrundeliegende Encoding zu ermitteln.
Diese fünf Zeichen können als stabil angenommen werden, da Produktion 23 und 80 diese explizit von einem von UTF-8 oder -16 abweichenden Encoding ausnehmen.

Für Dokumente im deutschen Sprachraum, d.h. XML-Ströme die häuptsächlich aus den um die deutschen Umlaute ergänzten Standard-ASCII-Zeichen bestehen, hat es sich in der Vergangenheit eingebürgert den Zeichensatz latin-1 (`ISO-8859-1`) zu verwenden, um die Mehrbytedarstellung der Umlaute und weiterer Sonderzeichen in der UTF-Codierung zu umgehen.
Jedoch enthält der latin-1-Zeichensatz nicht das unter Unicode-Zeichennummer 20AC abgelegte Eurosymbol (_) welches zur Abkürzung des Währungsbegriffes der europäischen Gemeinschaftswährung verwendet wird.
Dieses Symbol wurde in die unter Nummer 15 veröffentlichte aktualisierte Fassung der Zeichensatzfamilie 8859 aufgenommen. Daher sollte bei der Erstellung von XML-Dokumenten generell darauf geachtet werden entweder `ISO-8859-15` als Codierung zu wählen oder auf die ohnehin ungleich flexiblere UTF-Codierung zurückzugreifen.

Die Darstellung der Abbildung 4 faßt die syntaktischen Elemente abgekürzt zusammen:



Web-Referenzen 3: Weiterführende Links

•Payer, M.: *UNICODE, ISO/IEC 10646, UCS, UTF*
•Kuhn, M.: *UTF-8 and Unicode FAQ*
•SC Unipad ein kostenfreier Unicode Editor

Element Information Item

Jedes XML-Dokument enthält mindestens ein *Element*, das `Document Element`.
Seine, wie auch die Grenzen aller anderen Elemente, werden durch die *Start*- und *Ende-Marke* (engl. *Tag*) markiert. Für den Sonderfall eines *leeren Elements* bildet die Start- auch zugleich die Ende-Marke. Als eine Konsequenz können diese Elemente keine weiteren Kindknoten besitzen.

Die XML-Spezifikation legt den Aufbau des Start-Tags wie folgt fest (In XML-Spezifikation nachschlagen) :

```
[40] STag      ::= '<' Name (S Attribute)* S? '>'
[41] Attribute ::= Name Eq AttValue
```

Mittels der Tag-Namen werden die Typen eines Dokumentes definiert. Sie werden später, in Verbindung mit einem Grammatikmechanismus wie XML-Schema, zur Gültigkeitsprüfung herangezogen.
Der Aufbau der Elementnamen ist ähnlich zu den aus den Programmiersprachen bekannten Regeln. Am Beginn muß ein Buchstabe, ein Unterstrich oder der Doppelpunkt stehen. Darauf können nahezu beliebige Zeichen folgen, die über ihre Unicoderepräsentation genau definiert sind.
Leerzeichen und sog. *white spaces* (vgl. Produktion 3 der XML-Spezifikation) wie Tabulatoren und Zeilenvorschübe sind nicht zugelassen. Desweiteren darf ein Elementname weder Auszeichnungssymbole, wie die öffnenden und schließenden Winkelklammern, enthalten, noch mit der Zeichenkette *XML* beginnen. Die Zeichenfolge *XML* ist -- in allen Schreibweisen -- für die Standardisierung reserviert und wird ausschließlich in W3C-Dokumenten verwendet.
Durch den Namespace Standard (siehe Abschnitt 1.3) wird dem Doppelpunkt, als Trennsymbol zwischen Namensraumkürzel und Elementnamen, eine besondere semantische Bedeutung zugeschrieben. Daher sollte -- obwohl er spezifikationsgemäß ein erlaubtes Zeichen darstellt -- von seiner Verwendung in Elementnamen abgesehen werden.

Oftmals wird -- insbesondere in der Praxis -- die existierende und notwendige Unterscheidung zwischen *Tag* und *Element* nicht getroffen.
Die Tags oder Marken drücken beschreibende Information über ein Element aus. Der durch den Tag ausgedrückte Elementname liefert somit lediglich deskriptive Information über die Natur des Elements. Hierzu können Worte einer natürlichen Sprache verwendet werden, jedoch auch beliebige andere identifizierende Zeichenketten. Üblicherweise sind jedoch sprechende Tags anzutreffen.

Über den Tag-Namen hinaus kann ein Startelement auch noch *Attribute* enthalten (Vgl. Produktion 41). Diese sind jedoch nicht vom Typ *Element* und werden daher im Abschnitt *Attribute Information Item* betrachtet.

Der Aufbau eines Elementnamens wird durch die Produktionen 4ff definiert (In XML-Spezifikation nachschlagen) :

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5] Name     ::= (Letter | '_' | ':') (NameChar)*
[6] Names    ::= Name (S Name)*
[7] Nmtoken  ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Im Beispiel sind `Vorlesung`, `Titel` und `Hochschule` („normale“) Elemente, während `Pflichtfach` ein leeres Element darstellt.
Die Abbildung zeigt, daß auf der semantischen Ebene des Information Sets die syntaktische Unterscheidung zwischen Elementknoten mit Kindelementen und leeren Elementen des XML-Dokuments keine Berücksichtigung findet.

Eine Sonderstellung unter den Elementen eines Dokuments nimmt der ausgezeichnete Wurzelknoten ein, er wird auch durch das *Document Information Item* referenziert. Unterhalb dieses Knotens spannt sich der Dokumentbaum auf. Hierfür enthält jedes Element Information Item eine geordnete Menge (*children*) weiterer Elementknoten.
Die durch den Elementnamen verwirklichte Typisierung spiegelt sich im Information Set durch das Attribut *local name* wieder.

Darüberhinaus enthält jedes *Element Information Item* durch die Eigenschaft *namespace name* die Identifikation des Namensraumes, in dem dieses Element plaziert ist.
Das Namensraumkürzel, welches zur Identifikation eines Elements herangezogen wird, findet sich in der Eigenschaft *prefix*.
Der *local name* entspricht dem -- um Namensraumkürzel und trennenden Doppelpunkt gekürzten -- wiedergegebenen Elementnamen des XML-Dokuments.
Zusätzlich wird jeder Namensraum, der syntaktisch an die Attributdefinition angelehnt ist, in ein Element der ungeordneten Menge *namespace attributes* abgebildet, welche (nochmals) die Namensräume eines Elements beinhaltet.

Beispiel 2: Element mit deklariertem Namensraum

```
(1)...
(2)     <myNS:aParent xmlns:myNS="example.com">
(3)             <myNS:aElement/>
(4)     </myNS:aParent>
(5)...
```

Das Beispiel zeigt das leere Element `aElement` innerhalb des Elements `aParent`. Durch das Elternelement wird der Namensraum `example.com` deklariert und dem Kürzel `myNS` zugewiesen.
Gemäß den Prinzipien der Namensräume steht der auf dem Elternknoten deklarierte Namensraum auch in allen Kindknoten zur Verfügung. Daher enthält die Eigenschaft *in-scope namespaces* des Elements `aElement` auch die Namensräume der übergeordneten Elemente.
Das resultierende *Element Information Item* des Knotens `aElement` ergibt sich daher als (der Ausschnitt enthält nur die für das Beispiel relevanten Elemente):

```
local name    = aElement
namespace URI = example.com
prefix        = myNS
```

Nähere Ausführungen zur Bedeutung von Namensräumen und ihrer Verwendung finden sich im Abschnitt *Namensräume*.

Verweise auf die im Dokumentbaum nachfolgenden Knoten eines Elements werden in einer geordneten Liste *children* gesammelt. Ihre Inhalte sind sind vom Typ *Element Information Item*, *Character Information Item* und *Comment Information Item*.
Anhand der beiden Informationstypen *Element Information Item* und *Character Information Item* zeigen sich bereits die beiden Strukturierungsformen eines XML-Dokuments. Einerseits die durch die starke Verwendung von Elementen- und Attributen gekennzeichnete strukturierte Darstellung, andererseits die durch „eingestreuten“ Freitext entstehende charakteristische semistrukturierte Variante.
In beiden Fällen werden die textartigen Inhalte durch *Character Information Items* repräsentiert.
Das Beispiel zeigt die verschiedenen Auftretensformen exemplarisch. Der Inhalt der Elemente `title` und `organization` ist rein Zeichenketten-artig; jedoch mischt `vorlesung` strukturierten Inhalt (in Form der genannten Elemente) und unstrukturierte Information -- repräsentiert durch den Text `2002/03`.
Die XML-Spezifikation prägt für Zeichenketten-artige Inhalte, die optional durch *eingestreute* Elemente angereichert werden, den Begriff *mixed Content*.

children enthält jedoch keine Verweise auf die Attribute eines Elements. Diese sind durch die separate ungeordnete Menge *attributes* repräsentiert. Die Diskussion der als *Attribute Information Item* bezeichneten Mengenelemente findet sich im folgenden.

Die in der Abbildung dargestellte Beziehung *parent* verbindet jedes Element mit seinem übergeordneten.
Als Ziele dieser Referenz sind ausschließlich Ausprägungen von Document Information Item oder Element Information Item zugelassen.
Diese Festlegung untermauert nochmals die strikte Baumstruktur eines XML-Dokuments. Andernfalls müßte *parent* als Menge definiert werden.

Attribute Information Item

Das betrachtete Beispiel enthält, neben den Elementen, auch ein XML-Attribut.
Syntaktisch werden Attribute innerhalb eines Start-Tags plaziert und durch Namen-Wert-Paare ausgedrückt (In XML-Spezifikation nachschlagen) .

Der Information Set enthält folgende Eigenschaften zu jedem Attribut:

- **namespace name**: Namensraum des Attributs, falls definiert.
- **Lokaler Name**: Der um das eventuell definierte Namensraumkürzel bereinigte Attributname.
- **Präfix**: Namensraumkürzel des Namensraumes, innerhalb dessen das Attribut plaziert ist.
- **Normalisierter Wert**: Normalisierter Attributinhalt. Der Normalisierungsvorgang ist in Abschnitt 3.3.3 der XML-Spezifikation beschrieben (In XML-Spezifikation nachschlagen) .
  Unter anderem eliminiert er Zeilenumbrüche innerhalb des Attributinhalts und löst Entitätsreferenzen auf.
- **specified**: Boole'scher Wert, der angibt, ob das Attribut im XML-Dokument auftrat oder aufgrund einer Vorgabebelegung durch die DTD erzeugt wurde.
  Zur Ermittlung dieser Eigenschaft des Attribute Information Items ist die Definition und Referenzierung einer expliziten Grammatik notwendig.
- **Attributtyp**: Typ des Attributs. Zugelassene Belegungen sind: `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, `NMTOKENS`, `NOTATION`, `CDATA`, und `ENUMERATION`.
  Zur Ermittlung dieser Eigenschaft ist der Zugriff auf die DTD des Dokumentes notwendig. Ist dies nicht möglich, so ist der *attribute type* mit keinem Wert belegt.
- **Referenzen**: Handelt es sich bei dem Attribut um ein Referenzattribut (d.h. es ist als `IDREF(S)`,

`ENTITY`, `ENTITIES` oder `NOTATION` typisiert), so enthält diese Eigenschaft eine Verweisliste auf alle Auftreten des Attributwertes.

- **Eigentümerelement**: Bildet die Entsprechung zur *parent*-Eigenschaft des *Element Information Item*. Als solches enthält die Eigenschaft einen Verweis auf das Element, welches das Attribut beherbergt.

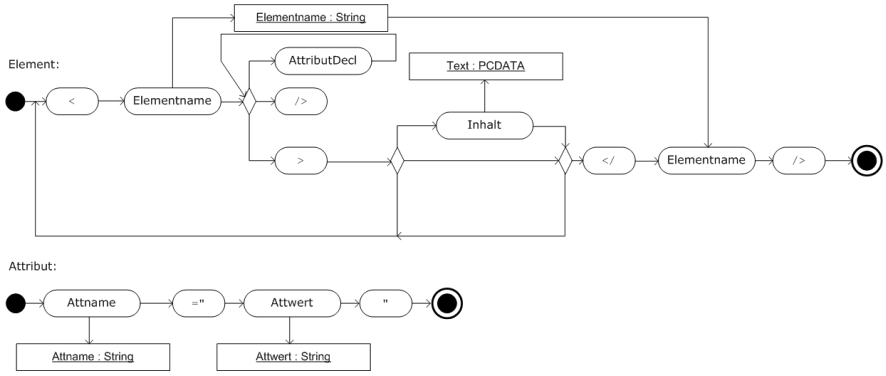
Im Vergleich zum *Element Information Item* erlaubt das Attribut keine weitere Unterstrukturierung (im XML-Sinne); insbesondere fehlen mengenwertige Eigenschaften zur Aufnahme der dann notwendigen Verweise. Stattdessen wird der gesamte Inhalt durch die Eigenschaft *normalized value* dargestellt.
Daher dürfen innerhalb von Attributen keine (Meta-)Symbole wie die öffnende Winkelklammer auftreten, die als Starttags (miß-)interpretiert werden könnten (In XML-Spezifikation nachschlagen) .

Auch die Form des Auftretens von Attributen innerhalb des definierenden Elements unterscheidet sich von der der Subelemente innerhalb eines Elements. Während Kindelemente durch die geordnete Liste *children* dargestellt werden, können Attribute (formalisiert in der ungeordneten Menge *attributes*) in beliebiger Reihenfolge angegeben werden, ohne die Dokumentsemantik zu verändern. Mehr noch, die Listenkonstruktion erlaubt das unterscheidbare mehrfache Auftreten desselben Elements. Diese Mimik ist für allgemeine Mengen, und damit für Attribute, nicht möglich.

Element vs. Attribut
Der Vergleich der Eigenschaften von Element und Attribut zeigt bereits, daß sich nicht weiter strukturierte Elemente auch durch Attribute darstellen ließen. Dies wirft innerhalb der Betrachtung der Syntax eines XML-Dokuments bereits die Frage nach der Organisation, und damit dem Entwurf, eines solchen auf.
Die bestehende XML-Spezifikation bleibt jedoch eine Anwendungs- oder Einsatzempfehlung zu dieser Fragestellung schuldig.
Aufgrund der inhärenten Einschränkungen der Attributprimitive bietet sich ihr Einsatz nur in einigen Sonderfällen an. Beispielsweise zur Darstellung deskriptiver Information über das enthaltende Element, die nicht Bestandteil der im XML-Dokument dargestellten Information ist. Hierbei kann es sich um Informationen höherer Ordnung, sog. Metainformation handeln.

Generell bieten sich Elemente immer dann an, wenn eine weitere Unterstrukturierung des Inhaltes gewünscht oder vielleicht zukünftig notwendig ist. Die Darstellungsform als Attribut würde in diesem Fall eine strukturelle Umorganisation des XML-Vokabulars erfordern, da die Spezifikation keine Unterstrukturierungsmöglichkeit für Attribute vorsieht.
Darüberhinaus gestatten Attribute keine Wiederverwendung in verschiedenen Bedeutungskontexten, da sie syntaktisch an das umgebende Element gebunden sind. Diese Einschränkung wird zwar durch die Einführung des Standards *XML Schema* weitgehend gemildert, jedoch nicht die zuvor genannte Mächtigkeitseinschränkung. Zusätzlich stellen Attribute die einzige Möglichkeit zur Typisierung des Inhaltes dar solange DTDs verwendet werden. Dieser Punkt dürfte jedoch durch den wachsenden Praxiseinsatz der XML Schemata immer mehr an Bedeutung verlieren.

Die Darstellung der Abbildung 5 faßt die syntaktischen Elemente abgekürzt zusammen:



Character Information Item

Die Betrachtung der Attribut- und Elementknotentypen im Information Set zeigt bereits die zwei grundlegenden Arten der Informationsdarstellung eines XML-Dokumentbaumes.
Die Eigenschaft *normalized value* des *Attribute Information Item*s kapselt den im XML-Dokument angegebenen Inhalt direkt im Informationsknoten. Der Datentyp der Eigenschaft ist für alle Dokumenttypen fixiert angebbar, da keine weitere Unterstukturierung von Attributen erfolgen kann.
Entgegensetzt hierzu verläuft die Argumentationslinie für Elemente. Ihr Inhaltsmodell kann eine freie Mischung aus Zeichenketten-Daten und weiteren Elementen aufweisen. Die Länge der Zeichenketten ist hierbei nicht näher festgelegt. Daher können diese im minimalen Falle nur aus einem einzelnen Zeichen bestehen. (In XML-Spezifikation nachschlagen) .
Innerhalb des Information Sets eines Dokuments werden alle Zeichen im Rumpf eines Elements als Ausprägungen des *Character Information Item*s dargestellt.

Jedes Character Information Item stellt das im Dokument gegebene Zeichen gemäß ISO 10646-Codierung in der Eigenschaft *character code* dar. Die Werte können hierbei jedoch nur in den durch die Spezifikation vorgegebenen Grenzen variieren (In XML-Spezifikation nachschlagen) . Darüberhinaus genügt bereits die

Unterstützung der UTF-8 und -16-Darstellung zur Erfüllung der Spezifikationsanforderungen an konforme Prozessoren.
Häufig werden white-spaces (Leerzeichen, Tabulator, Zeilenvorschub, Wagenrücklauf) zur besseren visuellen Strukturierung des XML-Dokumentes eingesetzt. So enthält das Beispieldokument jeweils nach der schließenden Marke einen Zeilenvorschub. Unter Datengesichtspunkten handelt es sich hierbei jedoch um keine verwertbare Information. Die Angabe der Berücksichtigung bzw. Vernachlässigung im XML-Dokument existierender white-spaces kann in der DTD gesetzt werden. Ist keine solche Deklaration gesetzt oder existiert keine explizite Grammatik, so hat die Eigenschaft *element content whitespace* keinen Inhaltswert.
Der als `parent`-Eigenschaft realisierte Verweis auf das beherbergende Elternelement bildet den Abschluß der Eigenschaften des *Character Information Items*.
Im betrachteten Beispiel sind unterhalb der Elemente `organization` und `title` *Character Information Element*-Ausprägungen plaziert. Die Darstellung zeigt diese als Objekte (Unterhalb des `organization`-Knotens wurde aus Übersichtlichkeitsgründen auf die Darstellung verzichtet).

Eine Sonderrolle kommt den Zeichen zu, die auch als Metasymbole der Auszeichnungssprache dienen. Sie dürfen daher nicht in XML-Dokumenten auftreten.
Bei diesen Zeichen handelt es sich um die beiden Winkelklammern, die einfachen und doppelten Anführungszeichen sowie das *Kaufmanns-Und*. Um eine Fehlinterpretation zu vermeiden existieren hierfür vordefinierte Textersetzungsmuster.
Jeder spezifikationskonforme XML-Prozessor berücksichtigt diese Symbole und gibt sie in der korrekten Darstellung an die Applikation weiter; damit sind diese Fluchtsymbole (engl. *escape characters*) aus Applikationssicht vollkommen transparent.

Tabelle 5: Vordefinierte Textersetzungsmuster

Entitätsreferenz	Ausgedrücktes Zeichen
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Web-Referenzen 4: Weiterführendes ... Die in XHTML v1.0 vordefinierten Entitäten
Latin-1 Entities
Special Entities
Symbole

Comment Information Item

Zur Dokumentation steht innerhalb jedes XML-Dokuments die von SGML ererbte Kommentierungssyntax zur Verfügung.
Die Spezifikation erlaubt die Anbringung von Kommentaren an zwei Stellen im XML-Dokument:

- Nach dem Prolog. (In XML-Spezifikation nachschlagen)
- An jeder beliebigen Stelle des Inhalts, außerhalb von Markup-Symbolen. (In XML-Spezifikation nachschlagen)

Nicht erlaubt sind demnach Kommentare in Tags, d.h. innerhalb geöffneter Winkelklammern. Dergleichen gilt für Kommentare selbst, was geschachtelte Kommentare verbietet.

Produktion 15 der XML-Spezifikation legt die Struktur wie folgt fest:

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Als Konsequenz sind innerhalb von Kommentaren alle Zeichen, auch Metasprachensymbole, zugelassen. Somit ist das beliebige „auskommentieren“ von Dokumentteilen möglich.
Als zentrale Einschränkung dürfen (aus SGML-Kompatibilitätsgründen) keine zwei aufeinanderfolgenden Trennstriche (*hyphen-minus*, ISO 10646 #x2D) innerhalb eines Kommentars auftreten, da diese fehlerhafterweise als Beginn des Kommentarendes interpretiert würden.

Der gesamte Inhalt eines Kommentars wird als uninterpretierte Zeichenkette in der Eigenschaft *content* des *Comment Information Item*s abgelegt.
Zusätzlich verweist jeder Kommentar über die bekannte *parent*-Eigenschaft auf seinen Elternknoten. Wie bereits durch die beiden Einsatzformen angedeutet, kann es sich hierbei ausschließlich um ein *Document Information Item* oder ein *Element Information Item* handeln.

Beispiel 3: Verschiedene Kommentarstrukturen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Root>
(3)     <!-- this is a comment -->
(4)     <ElementA>
(5)             <ElementB>
(6)                     <!--
(7)                     <ElementC/>
(8)                     <ElementD att1="..."/>
(9)                     -->
(10)            </ElementB>
(11)    </ElementA>
(12)</Root>
```

Das Beispiel zeigt verschiedene Einsätze von Kommentaren. Zunächst eine einzeilige Anmerkung, die nur verschiedene Zeichen versammelt. Im Anschluß einen mehrzeiligen Kommentar, der auch XML-Strukturen beinhaltet. Ein prozessierender Zugriff auf den Kommentarinhalt ist jedoch nicht vorgesehen, und wird durch gängige Parser und APIs zumeist nicht unterstützt.

Processing Instruction Information Item

Im Gegensatz zu den prinzipiell in beliebigem Freitext formulierbaren Kommentaren, die üblicherweise zur Kommunikation mit einem menschlichen Leser des XML-Dokuments dienen, zielt die *Processing Instruction* und das zugehörige Element des Information Sets auf Kommentare, welche einen maschinellen Verarbeiter des XML-Dokuments, den XML-Prozessor, betreffen.

Im Grunde genommen läuft die Anreicherung eines XML-Dokuments mit Verarbeitungsinformation der Idee einer deskriptiven Auszeichnungssprache entgegen ...
Jedoch wurde für die XML beschlossen, nicht zuletzt aus Kompatibilitätsgründen zu SGML, dieses Sprachmerkmal beizubehalten. Eine mögliche weitere Erklärung könnte das syntaktische Aussehen der XML-Deklaration innerhalb des des Dokumentprologs sein. Ihre in Produktion 23ff festgelegte Struktur stellt eine Anwendung der Processing Instruction dar, auch wenn dies innerhalb der Spezifikation nicht explizit formuliert wird.

Die Syntax einer *Processing Instruction* lautet:

```
[16] PI       ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Eine Processing Instruction wird demnach immer durch eine öffnende Winkelklammer und ein folgendes Fragezeichen eingeleitet. Daran schließt sich die Benennung der Applikation an, für die diese Instruktion eingefügt wurde. Optional können weitere Zeichen -- ausgenommen der Kombination aus Fragezeichen und schließender Winkelklammer -- folgen.
Das adressierte System kann beliebig identifiziert werden, jedoch ist die Zeichenkette *XML* in allen Variationen ausgeschlossen.
Unbedachterweise verbietet die Spezifikation jedoch nicht die Bildung von Namen, die *XML* als Präfix nutzen ... Jedoch sollte von der Nutzung solcher Konstruktionen abgesehen werden, da sie zur Verwirrung der (menschlichen) Leser beitragen.

Wie Kommentare auch können Processing Instructions an beliebiger Stelle innerhalb des XML-Dokuments auftreten: Vor Beginn des Wurzelelements sowie im Rumpf jedes Elements. Nicht gestattet ist ihre Angabe in Elementnamen und Attributen.
Ergänzend sei angemerkt, daß die Angabe von Processing Instructions auch innerhalb der *Document Type Definition* erfolgen kann. (siehe *Document Type Definition Information Item*).

Beispiel 4: Verschiedene Processing Instructions

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<?mySystem value="42"?>
(3)<root>
(4)   <?System2?>
(5)   <elementA>
(6)     <?System3 a="1" anotherValue?>
(7)  </elementA>
(8)</root>
```

Download des Beispiels

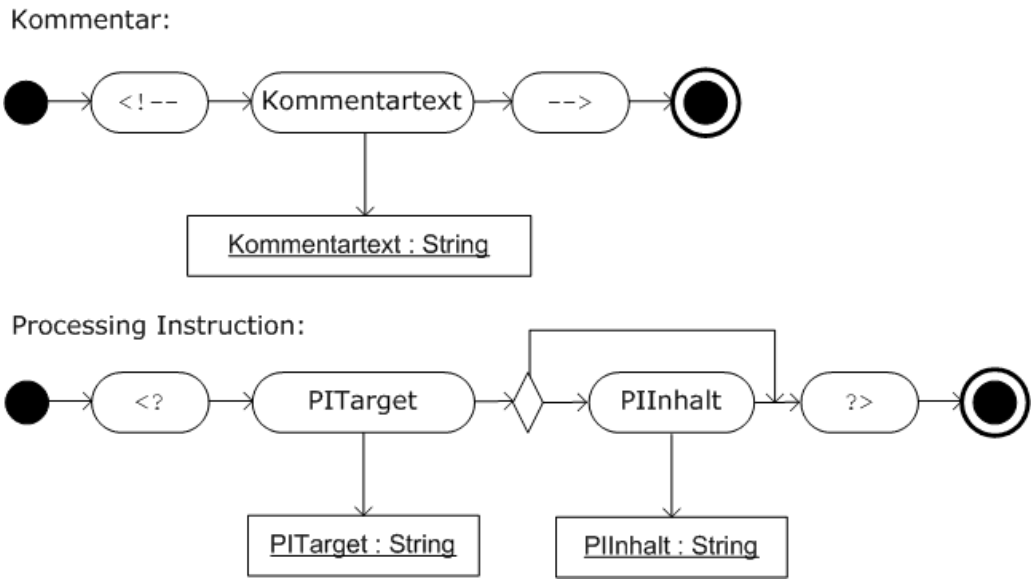
Übung 1: Processing Instructions

Begründen Sie mit Hilfe der XML-Spezifikation warum Processing Instructions nicht innerhalb von Elementen und Attributen zugelassen sind.
Hinweis: Es gibt mehr als eine Begründung!

Das *Processing Instruction Information Item* enthält die angesprochene Zielapplikation als Namen innerhalb der Eigenschaft *target*.
Der weitere Inhalt der Deklaration wird uninterpretiert als Zeichenkette in die Eigenschaft *content* übernommen.
Neben einem Verweis auf die Basis-URI der Processing Instruction wird durch *parent* das Elternelement -- entweder ein Knoten des Typs *Document Information Item* oder *Element Information Item* -- referenziert.

Zur Formalisierung der Identifikation der Zielapplikation empfiehlt die XML-Spezifikation die Verwendung des Sprachmittels *Notation*.

Die Darstellung der Abbildung 6 faßt die syntaktischen Elemente abgekürzt zusammen:



Namespace Deklaration Information Item

Jedem im XML-Dokument definierten Namensraum ist ein *Namespace Deklaration Information Item* zugeordnet. Es enthält die notwendigen syntaktischen Details zur Identifikation des Namensraumes:

- **prefix**: Das gewählte Präfix des Namensraumes, bzw. leer falls es sich um den Vorgabenamensraum handelt.
- **namespace name**: Der Name des Namensraumes, an den das Präfix gebunden ist.

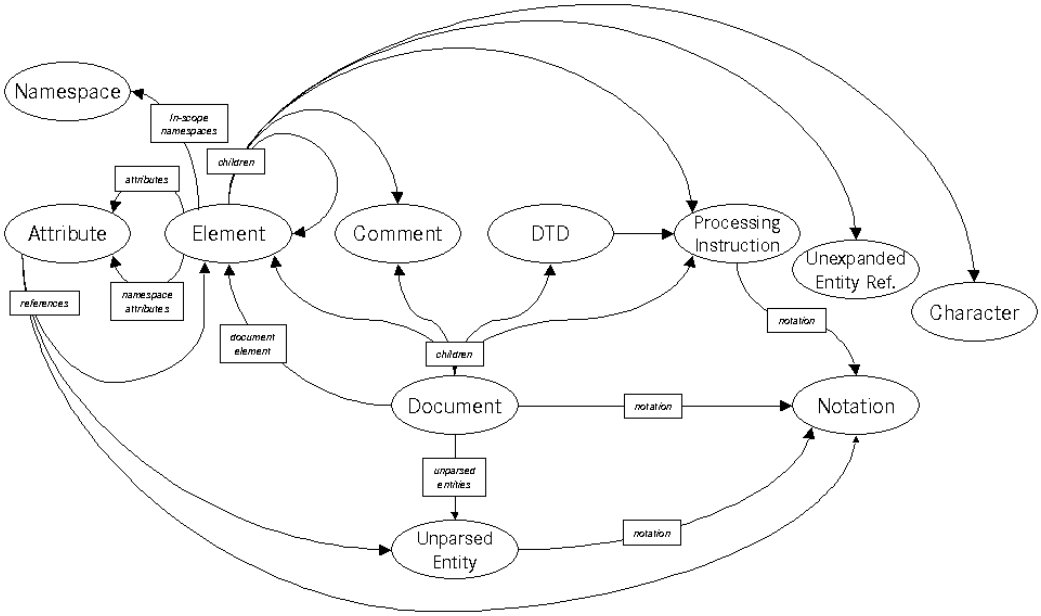
Beispiel 5: Beispiel eines Dokuments mit Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<root>
(3)     <elementA>...</elementA>
(4)     <elementB xmlns="http://www.fh-furtwangen.de">...</elementB>
(5)     <elementC xmlns:abc="http://www.xyz.com">
(6)             ...
(7)             <abc:elementD/>
(8)     </elementC>
(9)</root>
```

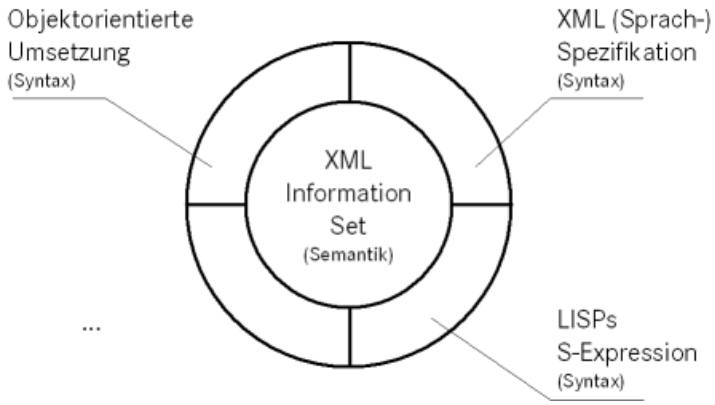
Für das Beispiel lauten die Namensräume wie folgt:

Elementname	Namensraum
root	(Das Element befindet sich im leeren Namensraum)
elementA	(Das Element befindet sich im leeren Namensraum)
elementB	http://www.fh-furtwangen.de
elementC	(Das Element befindet sich im leeren Namensraum)
elementD	http://www.xyz.com

Eine ausführliche Betrachtung zur Verwendung von Namensräumen findet sich im entsprechenden Abschnitt.



Die Graphik der Abbildung 7 stellt alle diskutierten Elemente des Information Sets in der Übersicht mit ihren Beziehungen dar. Zur Veranschaulichung wurde eine einfache Graphenstruktur gewählt, die alle Informationseinheiten als Knoten (darstellt als Ellipsen) und alle zugelassenen Beziehungen als gerichtete Kanten zwischen diesen enthält. Zusätzlich ist an die Kanten die Art der Beziehung angetragen.
Den Ausgangspunkt der baumartigen Struktur eines XML-Dokuments bildet die im Zentrum abgebildete Primitive `Document Information Item`, die alle weiteren Inhalte eines Dokuments über die `children`-Kante als Kindknoten enthält. Ferner fällt in dieser Darstellung besonders auf, daß lediglich `Element Information Items` über weitere Kindknoten verfügen und so die charakteristische XML-Struktur herausbilden. Alle übrigen Primitive dienen überwiegend als Blattknoten des Baumes.



Die Graphik der Abbildung 8 setzt die durch den Infoset-Standard definierte Semantik und die darauf aufsetzenden Syntaxen in Beziehung. Der XML-Basisstandard definiert hierbei nur eine von mehreren möglichen Syntaxen zur Darstellung von Infoset-Ausprägungen. Ebenso denkbar wäre der Einsatz anderer Darstellungen gleicher Mächtigkeit wie beispielsweise der S-Expression aus LISP oder objektorientierte Umsetzungen.

Auf Basis der Definitionen des Information Sets läßt sich ein beliebiges XML-Dokument, welches den Strukturierungsprinzipien des Infosets folgt, als *wohlgeformt* (*well-formed*) charakterisieren.

Definition 6: Wohlgeformtes XML-Dokument

Ein textartiges Objekt, dessen Inhalt folgenden Anforderungen genügt:

- Das XML-Dokument nutzt eine DTD, oder enthält die Deklaration `standalone="yes"`
- Zu jedem Start-Tag existiert genau ein Ende-Tag. Bei leeren Elementen können diese zu einem Tag zusammenfallen.
- Korrekte Elementschachtelung, d.h. Elemente überlappen einander nicht.
- Genau ein Wurzelelement.
- Alle Attributwerte sind in einfachen oder doppelten Anführungszeichen.
- Kein Start-Tag (oder Tag der ein leeres Element einleitet) enthält zwei oder mehr Attribute desselben Namens.
- Keine Kommentare oder Processing Instructions innerhalb

von Tags.
- Kommentare beginnen und enden mit genau zwei Bindestrichen.
- Die Sonderzeichen < und & treten nicht innerhalb von Elementinhalten oder Attributwerten auf.

siehe XML-Spezifikation

Der Textstrom des Beispiels 6 zeigt ein nicht-wohlgeformtes XML-Dokument, welches gegen eine Reihe der in Definition 6 verstößt:

Beispiel 6: Ein nicht wohl-geformtes XML-Dokument

```
(1)<?xml version="1.0"?>
(2)<root>
(3)     <elementA att=a oder b>
(4)             <elementB> iff a<b ==> ...
(5)     </elementA>
(6)     <elementC att1="42" att1="3.14">
(7)             <elementD <?do-something?> >
(8)     </elementC>
(9)             </elementD>
(10)    <!---- dies ist nicht erlaubt ---->
(11)</root>
```

Download des Beispiels

So findet sich in Zeile 3 ein nicht in die erforderlichen Anführungszeichen eingeschlossener Attributwert.
Der textuelle Elementinhalte des in Zeile 4 geöffneten Elements `elementB` enthält ein öffnendes Winkelklammersybol, welches um Fehler während des Einlesevorganges zu vermeiden durch die alternative Zeichensequenz `<` hätte ersetzt werden müssen. Darüberhinaus fehlt das korrekte schließende Tag zum Öffnenden.
Innerhalb des Elements `elementC` der Zeile 6 wird zweifach ein identisch benanntes Attribut definiert.
Im öffnenden Tag des in Zeile 7 definierten Elements `elementD` findet sich eine -- dort nicht zugelassene -- Processing Instruction.
Überdies überlappen sich die Elementgrenzen der Elemente `elementC` und `elementD` und zusätzlich wird der in Zeile 10 plazierte Kommentar nicht durch die erforderlichen genau zwei Bindestriche eingegrenzt.

2.2 XML-Namensräume

Namensräume

Die XML-Namensräume wurden schon verschiedentlich erwähnt. Sie bilden die wichtigste, und offensichtlichste Weiterentwicklung der XML-Urspezifikation seit ihrer Veröffentlichung.
Trotz ihrer engen Beziehung zum XML-Kernstandard bildet die Recommendation *Namespaces in XML* eine eigenständige Spezifikation. Aufgrund der engen syntaktischen Beziehung zum XML-Standard und der großen praktischen Bedeutung, sowie des Einflusses auf die weitere Entwicklung verschiedenster Sekundärstandards und XML-Sprachen, werden die Namensräume explizit in der Neuauflage des XML-Standards berücksichtigt. Einen Beleg hierfür bildet die Anmerkung zu Abschnitt 2.3 *Common Syntactic Constructs*. Dort wird von der -- laut Syntaxproduktion 5 erlaubten -- Verwendung des Doppelpunktes in Elementnamen abgeraten. Dies geschieht, um Mehrdeutigkeiten, oder schlichtweg der Verwirrung des Anwenders, vorzubeugen, da es sich beim Doppelpunkt um ein Symbol besonderer Bedeutung innerhalb der Namensraumdeklarationen handelt.

Warum Namensräume?

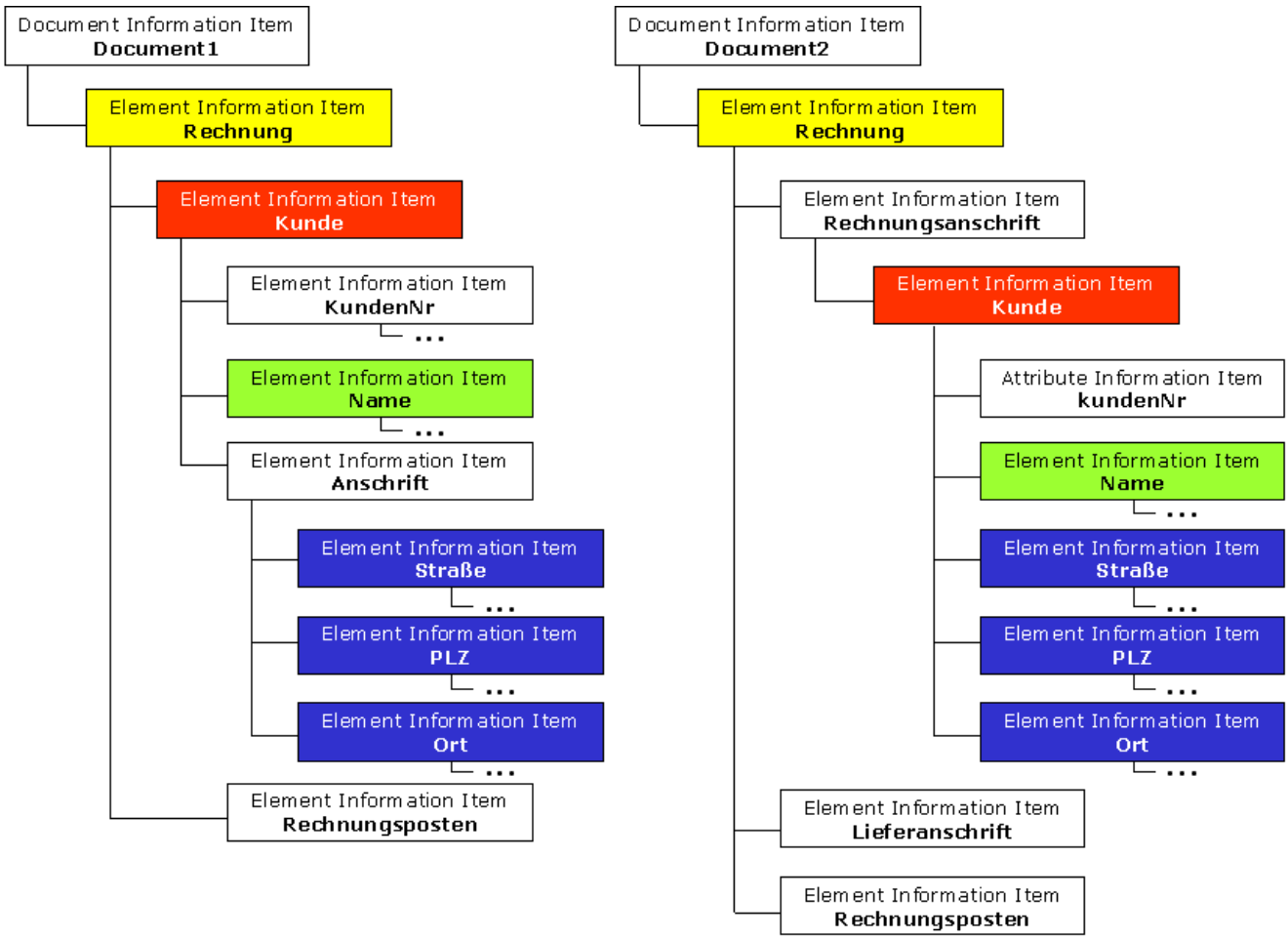
Die breite Entwicklung immer neuer XML-Sprachen führt zwangsläufig zu Mehrfachentwicklungen für ähnliche oder identische Problemstellungen. Technisch betrachtet äußerst sich dies -- bei natürlichsprachlicher Benennung der Elemente -- durch die Verwendung identischer Bezeichner in verschiedenen XML-Sprachen. Hierbei bilden die verschiedenen Sprachen Anwendungskontexte, innerhalb derer die Bezeichner, durch Einbezug der Anwendungssemantik, eindeutig sind; andernfalls kann unterstellt werden, daß bereits durch die Sprachentwicklung andere Benennungskonventionen gewählt worden wären.
In der Konsequenz der Verfügbarkeit verschiedenster XML-Sprachen für beliebige Anwendungsbereiche entsteht der (berechtigte) Wunsch existierende Sprachfragmente in eigene Sprachen zu integrieren, um so zeitraubenden und vielfach fehleranfälligen Mehrfachentwicklungen vorzubeugen. Jedoch tritt bei diesem Integrationsszenario die u. U. kontextabhängige Elementeindeutigkeit zu Tage.
Das Beispiel zeigt zwei Dokumente identischen Informationsumfanges, die lediglich strukturell differieren.

Beispiel 7: Ein Rechnungsdokument

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <Name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)      </Anschrift>
(11)   </Kunde>
(12)   <Rechnungsposten>
(13)    ...
(14)    </Rechnungsposten>
(15)</Rechnung>
```

Beispiel 8: Eine alternative Rechnungsstruktur

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)     <Rechnungsanschrift>
(4)             <Kunde kundenNr="4711">
(5)                     <Name>Max Mustermann</Name>
(6)                     <Straße>Musterplatz 1</Straße>
(7)                     <PLZ>12345</PLZ>
(8)                     <Ort>Musterstadt</Ort>
(9)             </Kunde>
(10)    </Rechnungsanschrift>
(11)    <Lieferanschrift>
(12)   ...
(13)   </Lieferanschrift>
(14)    <Rechnungsposten>
(15)    ...
(16)    </Rechnungsposten>
(17)</Rechnung>
```



Die beiden Bäume mit *Information Set*-Ausprägungen zeigen die Struktur der Beispieldokumente. Dabei sind Knoten die den selben Inhalt repräsentieren mit identischen Farben unterlegt, unabhängig davon um welchen Knotentyp es sich handelt. Die *Character Information Item* Knoten wurden aus Übersichtlichkeitsgründen weggelassen und durch Punkte angedeutet, sie sind jedoch für die vorliegende Betrachtung nicht von Interesse.

Einige der Elemente und Attribute werden in beiden Dokumenten mit gleichen Inhalten verwendet; z.B. `Name`, `Ort` oder `PLZ`. Dies äußert sich in identischen Teilbäumen unterhalb der Information Set-Knoten welche diese XML-Elemente repräsentieren. Hieraus läßt sich ableiten, daß die beiden vorgestellten Sprachen an den genannten Stellen keine strukturelle Differenz aufweisen.
Dagegen unterscheiden sich die Kindknoten der Elemente `Rechnung` und `Kunde` hinsichtlich ihrer Struktureigenschaften. So folgt im ersten Beispieldokument auf das `Rechnung`-Element direkt der `Kunde`, während im zweiten XML-Dokument zunächst ein Element mit dem Namen `Rechnungsanschrift` erwartet wird.
Dergleichen gilt für die Kindelemente des `Kunden`. Im zweiten Beispieldokument wird die diesem Element untergeordnete Kundennummer durch ein Attribut (`kundenNr`) dargestellt. Dagegen codiert das erste Beispiel diese Information direkt in den Elementinhalt.

Solange die beiden Dokumente in unterschiedlichen Anwendungswelten (Unternehmen o. ä.) verwendet werden, ist der gewählte Ansatz nicht problematisch. Bedenklich wird er jedoch in mindestens zweierlei Hinsicht:
Zunächst bei der „Mischung“ der beiden Dokumente. Dieser Wunsch tritt bei praktischen Problemstellungen häufig auf, wenn es um die Übernahme von XML-codierten Daten in ein anderes XML-Dokument geht. In der Konsequenz folgt das entstehende Zieldokument nicht mehr den Strukturierungsregeln eines der Ausgangsdokumente; mithin entsteht eine neue Dokumentstruktur, deren Regeln nicht explizit dokumentiert sind.
Eine weitaus größere Herausforderung stellt die Zusammenfassung und Veröffentlichung von XML-Strukturen in sog. Schemabibliotheken oder Datenbanken dar. Hier werden zwar die Dokumente nicht vereinigt, jedoch offenbart sich die gleiche Anwendungsdomäne (z.B. Rechnungsverwaltung, Stücklisten, Produktstrukturen) als problematisch, da sie die XML-Strukturen in direkte Konkurrenz treten läßt. In Zeiten immer stärker werdenden ökonomischen Flexibilisierungsdruckes erweist sich dies als äußerst kontraproduktiv, im Hinblick auf eine angestrebte Standardisierung. Die offene Konkurrenz verschiedener *Dialekte* innerhalb einer Domäne verzögert damit oft die Entscheidung zum Einsatz eines Sprachformates.

Einen anderen interessanten Anwendungsfall stellt der ausdrückliche Wunsch nach der Einbettung fremder Sprachelemente dar. Diese Form der Wiederverwendung knüpft an das durch öffentlich verfügbare XML-Formate eröffnete Anwendungsfeld an. Da nicht in jedem Fall ein alle Anforderungen erfüllendes existierendes XML-Format ermittelt werden kann, jedoch verschiedene vorhandene Formatteile des gewünschten Umfanges abdecken, entsteht der Wunsch nach einer selektiven Weiterverwendung. Ein bekanntes Beispiel bilden Freitexte in beliebigen XML-Sprachen, welche auf Teile des (X)HTML-Sprachumfanges zurückgreifen. Gleichzeitig ist damit die Semantik der Elemente durch den zugehörigen W3C-Standard festgelegt. XHTML selbst stellt ein interessantes Anwendungsbeispiel für die gemeinsame Verwendung verschiedener XML-Sprachen in einem Dokument dar. So können Web-Seiten neben den bekannten Textstrukturen (XHTML) auch mathematische Symbole und Formeln (in der XML-Sprache MathML) und Vektorgraphiken (in der XML-Sprache SVG) enthalten.
Als Nebeneffekt der Wiederverwendung existierender XML-Sprachen verringern sich mögliche Fehlerquellen, was in der Konsequenz zur Erhöhung der Qualität der entstehenden Sprachen führt.

Zusammenfassend lassen sich die (Hinter-)Gründe der Namensraumeinführung wie folgt darstellen:

- Wiederverwendung bestehender (fremder) XML-Strukturen in eigenen Dokumenten.
- Wunsch nach breiteren Standards.
- Verringerung des Designaufwandes.
- Nutzung bereits gesammelter Designerfahrung.
- Zusammenführung verschiedener XML-codierter Inhalte (*heterogeneous content syndication*).

Definition 7: Namensräume

XML-Namensräume stellen eine XML-basierte Syntax zur Verfügung um Element- und Attributnamen eines Vokabulars eindeutig zu identifizieren und so Bedeutungsüberschneidungen durch gleichbenannte Elemente- oder Attribute in zu unterscheidenden Vokabularen auszuschließen. XML-Namensräume bilden damit die notwendige Voraussetzung zur freien dezentralen Entwicklung eigener Vokabulare ohne die Möglichkeit einer späteren Syndikatisierung zu verlieren.

Konzept der Namensräume:
Die Recommendation *Namespaces in XML* definiert die Syntax und Semantik der Namensräume. Ihr Konzept wurde rund ein Jahr nach Verabschiedung der ersten XML-Version eingeführt. Daher wurde der Kompatibilität mit bereits existierenden XML-Dokumenten große Priorität eingeräumt.

Grundidee der Namensräume ist es, die Element- und Attributnamen dergestalt zu erweitern, daß (auch nach Vereinigung beliebiger Dokumente wieder) eineindeutige Bezeichner entstehen. Dies könnte durch anwenderdefinierte Erweiterungen geschehen, sie trügen jedoch wiederum die Gefahr in sich, daß sie unbeabsichtigt mehrfach benutzt würden.
Daher scheidet der unkoordinierte Einsatz solcher Namenserweiterungen aus. Jegliche Koordination

bedingt jedoch inhärent eine zentrale Vergabestelle zur Registrierung der vergebenen Namen, die über die Eindeutigkeit wacht und Mehrfachnutzungen unterbindet.
Die Einführung einer solchen Stelle hätte jedoch einen unüberschaubaren Verwaltungsaufwand bedeutet, den das W3C nicht zu leisten im Stande wäre. Man nehme nur als Vergleich das Vergabeverfahren von Einträgen des Internet Domain Name Systems (DNS), welches bereits dezentral durch die einzelnen nationalen Domain-Registrars gehandhabt wird. Der dort anzutreffende Aufwand hätte sich für XML-Namensräume potenziert, legt man pro Domainadresse mehrere Namensräume zugrunde.

Ziel des W3C war es, durch die Namensräume einen gleichermaßen mächtigen als auch leicht zu handhabenden und zu administrierenden Identifikationsmechanismus zu etablieren. Offenkundig wird diesem Anspruch nur ein (überwiegend) dezentraler, aber dennoch die Eineindeutigkeit garantierender, Ansatz gerecht.
Diesen Anforderungen genügt das aus IETF RFC 2396 bekannte Namensschema der *Uniform Resource Identification* (URI) (später aktualisiert in IETF RFC 2732). Es kombiniert zentrale und dezentrale Elemente in der Handhabung, und ermöglicht so -- trotz Existenz und Pflege einer zentralen Registratur -- größtmögliche Flexibilität in der Anwendung. Der bekannteste Einsatz von URI-Namen ist der im World-Wide-Web allgegenwärtige *Uniform Ressource Locator* (URL) (IETF RFC 1738); einer Untermenge der URI.
Die zentrale Komponente findet sich im Domainnamen verwirklicht. Er ist entweder durch die IP-Adresse (konkret: IPv4-Adresse; im Falle des RFC 2732: der IPv6-Adresse) oder deren literaler Repräsentation gegeben. Unterhalb der Domainebene kann durch deren Verwalter eine beliebige Strukturierung vorgenommen werden. Die verschiedenen Ebenen werden dabei durch ISO-10646/ASCII #x2F „/“ voneinander abgetrennt.
Wie auch bereits bei URLs notwendig, ist das Schema (*URI scheme*) (z.B. `http`) zwingend mitanzugeben.

Trotz der Möglichkeit XML-Namensräume durch URLs zu identifizieren handelt es sich dabei nicht die Bezeichnung einer Internetquelle. Die verwendete Zeichenkette dient ausschließlich Benennung der im Namensraum versammelten XML Element Information Items und Attribute Information Items.
Die Auflösung des Namensraumbezeichners durch einen XML-Prozessor ist nicht vorgesehen.

Nachfolgend ist die in definierte Syntax einer URI wiedergegeben. Sie wurde behutsam an die in der XML-Spezifikation verwendete BNF-Notation (In XML-Spezifikation nachschlagen) angepaßt, ohne jedoch die Produktionen in ihrer Struktur zu verändern.

```
[URI1]  URI-reference ::= (absoluteURI | relativeURI)? ("#" fragment)?
[URI2]  absoluteURI   ::= scheme ":" ( hier_part | opaque_part )
[URI3]  relativeURI   ::= ( net_path | abs_path | rel_path ) [ "?" query ]
[URI4]  hier_part     ::=  ( net_path | abs_path ) ("?" query)?
[URI5]  opaque_part   ::= uric_no_slash uric?
[URI6]  uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
[URI7]  net_path      ::= "//" authority abs_path?
[URI8]  abs_path      ::= "/" path_segments
[URI9]  rel_path      ::= rel_segment abs_path?
[URI10] rel_segment   ::= (unreserved | escaped |
                          ";" | "@" | "&" | "=" | "+" | "$" | "," )+
[URI11] scheme        ::= alpha (alpha | digit | "+" | "-" | "." )*
[URI12] authority     ::= server | reg_name
[URI13] reg_name      ::= ( unreserved | escaped | "$" | "," |
                          ";" | ":" | "@" | "&" | "=" | "+" )+
[URI14] server        ::= ((userinfo "@")? hostport)?
[URI15] userinfo      ::= ( unreserved | escaped |
                          ";" | ":" | "&" | "=" | "+" | "$" | "," )*
[URI16] hostport      ::= host (":" port)?
[URI17] host          ::= hostname | IPv4address
[URI18] hostname      ::= ( domainlabel "." )* toplabel (".")?
[URI19] domainlabel   ::= alphanum | alphanum *( alphanum | "-" ) alphanum
[URI20] toplabel      ::= alpha | alpha (alphanum | "-" )* alphanum
[URI21] IPv4address   ::= digit+ "." digit+ "." digit+ "." digit+
[URI22] port          ::= digit*
[URI23] path          ::= (abs_path | opaque_part)?
[URI24] path_segments ::= segment ("/" segment)*
[URI25] segment       ::= pchar* (";" param)*
[URI26] param         ::= pchar*
[URI27] pchar         ::= unreserved | escaped |
                          ":" | "@" | "&" | "=" | "+" | "$" | ","
```

```
[URI28] query         ::= uric*
[URI29] fragment      ::= uric*
[URI30] uric          ::= reserved | unreserved | escaped
[URI31] reserved      ::= ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                          "$" | ","
[URI32] unreserved    ::= alphanum | mark
[URI33] escaped       ::= "%" hex hex
[URI34] hex           ::= digit | "A" | "B" | "C" | "D" | "E" | "F" |
                          "a" | "b" | "c" | "d" | "e" | "f"
[URI35] digit         ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                          "8" | "9"
[URI36] uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
```

Die Produktionen `alphanum`, `lowalpha` sowie `upalpha` zur Konstruktion der alphanumerischen Namen wurden aus Übersichtlichkeitsgründen weggelassen.

Neben einigen anderen gängigen URI-Varianten stellt das nachfolgende Beispiel einige der möglichen syntaktisch korrekten URIs zusammen, die für die späteren Betrachtungen von Interesse sind.

Beispiel 9: Gültige URIs

```
(1)http://www.wi.fh-furtwangen.de
(2)http://meinrechner.wi.fh-augsburg.de
(3)mailto:mario@jeckle.de
(4)ftp://ftp.shareware.com
(5)http://www.jeckle.de/xml/vorlesung/script.htm#Namespaces
(6)#EinfuehrungUndUeberblick
(7)urn:oasis:names:specification:docbook:dtd:xml:4.1.2
(8)urn:oid:1.3.6.1.2.1.27
(9)org.omg/standards/UML
```

Exkurs: URIs, URLs, URNs ...

Vielfach wird in der Praxis die Abgrenzung der im Internet gebräuchlichen Adressierungs- und Identifikationsmechanismen nicht trennscharf vollzogen.
Darüberhinaus trat im Laufe der Entwicklung eine merkliche Bedeutungsverschiebung insbesondere zwischen der *Uniform Resource Identifikation* und den als WWW-Adressen genutzten *Uniform Resource Locators* ein.

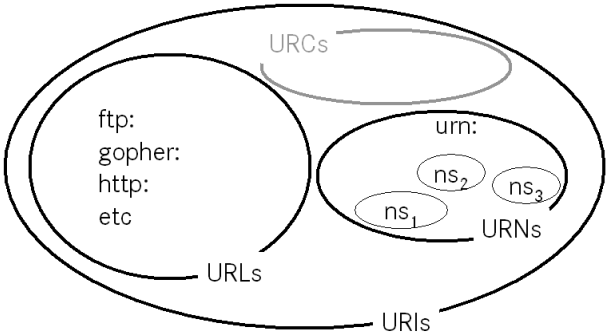
Gegenwärtig wird die Begriffsabgrenzung wie in Abbildung 10 schematisch dargestellt vollzogen:

- *Uniform Resource Identification* (URI) dient als abstrakter Oberbegriff eineindeutig identifizierbarer Web-Ressourcen.
  Konzeptionell sind URIs:
  - über Zeit und Raum eindeutig
  - für Menschen leicht zu merkend
  - mit keinerlei Registrierungskosten verbunden
  - unabhängig von der tatsächlichen Lokalisation der so identifizierten Ressource

  Der URI-Raum zerfällt in die disjunkten Bezeichnerschemata URL, URN und URC.
- *Uniform Resource Location* (URL) bezeichnet den physischen Aufenthaltsort einer Ressource, etwa den Ablageort einer HTML-Seite.
  Beispiele:
  - `http://www.jeckle.de/vorlesung/xml/script.html`
  - `http://www.wi.fh-furtwangen.de/`
  - `mailto:mario@jeckle.de`
  - `ftp://example.org/aDirectory/aFile`
  - `news:comp.infosystems.www`
  - `tel:+1-816-555-1212`
  - `ldap://ldap.example.org/c=GB?objectClass?one`
  - `urn:oasis:SAML:1.0`
- *Uniform Resource Name* (URN) bezeichnen den eineindeutigen Namen einer beliebigen Resource. Für die URN existiert kein Auflösungsmechanismus durch den die physische Lokation ermittelt werden könnte. URNs dienen daher ausschließlich der eindeutigen Benennung!
  Syntaktisch folgt auf das definierte Kürzel `urn` eine Zeichenkette welche eine Weiterklassifikation der Ressource gestattet. Hierdurch wird eine weitere Partitionierung des URN-Raumes erzielt. Diese *namespace ID* genannten Zeichenketten unterliegen einem globalen Registrierungszwang um ihre Eindeutigkeit zu gewährleisten. Diese Unterstrukturierungen sind in der Abbildung als ns_1 bis ns_3 benannt.

Beispiele:
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2
- urn:oid:1.3.6.1.2.1.27

- *Uniform Resource Citation* (URC) schlägt die Brücke zwischen Lokationsbezeichnung und reiner Benennungskonvention. Eine URC verweist in eine Metadatenstruktur welche die physischen Ressourcen-Aufenthaltsorte katalogisiert.
  Dieser URI-Typ ist jedoch gegenwärtig kaum verbreitet.



Web-Referenzen 5: Weiterführende Links
•URIs, URLs, and URNs: Clarifications and Recommendations
•The Anatomy of an URL

Verwendung von Namensräumen:
Am naheliegendsten wäre nach der Zielsetzung der Verwendung von URIs zur eindeutigen Benennung von XML-Element- und Attributnamen, die URI direkt vor dem XML-Bezeichner zu plazieren, evtl. separiert durch ein Trennsymbol wie den Doppelpunkt „:“.
Hieraus entstünden dann, auf jeden Fall eindeutige, Element- und Attributnamen wie beispielsweise für das erste Beispieldokument dieses Kapitels (die URI `http://www.example.com/sales` werde zur Identifizierung verwendet):

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)     <http://www.example.com/sales:Rechnung>
(3)             <http://www.example.com/sales:Kunde>
(4)                     <http://www.example.com/sales:KundenNr>4711</http://www.example.com/
sales:KundenNr>
(5)                     <http://www.example.com/sales:Name>Max Mustermann</http://www.
example.com/sales:Name>
(6)                     <http://www.example.com/sales:Anschrift>
(7)                             <http://www.example.com/sales:Straße>Musterplatz 1</http://
www.example.com/sales:Straße>
(8)                             <http://www.example.com/sales:PLZ>12345</http://www.example.
com/sales:PLZ>
(9)                             <http://www.example.com/sales:Ort>Musterstadt</http://www.
example.com/sales:Ort>
(10)                    </http://www.example.com/sales:Anschrift>
(11)            </http://www.example.com/sales:Kunde>
(12)    <http://www.example.com/sales:Rechnungsposten>
(13)            ...
(14)    </http://www.example.com/sales:Rechnungsposten>
(15)</http://www.example.com/sales:Rechnung>
```

Bei entsprechender Nachbearbeitung des zweiten Beispieldokumentes mit einem anderen URI-identifizierten Namensraum, entstehen eindeutige Element- und Attributnamen, die nicht mehr kollidieren.

Jedoch verstößt diese Lösung gegen die in Produktion 5 der XML-Spezifikation formulierte syntaktische Einschränkung. Sie erlaubt das in URIs elementare Pfadtrennersymbol („/“) (aus den URI-Produktionen 8, 24 und 31) nicht in XML-Namen (#x2F findet sich nicht in den in Produktion 85 aufgeführten Unicode-Blöcken).
Die Integration der Namensräume auf diesem Weg hätte daher eine Modifikation der XML-Spezifikation nach sich gezogen. Diese erweiternde Aufweichung der zugelassenen Namen für Elemente und Attribute hätte jedoch mit der Kompatibilität zu SGML gebrochen, und somit eine der Grundforderungen der XML-Entwicklung verletzt.
Darüberhinaus ist die Spezifikation vollständiger URIs für Menschen „unhandlich“ und reduziert die Lesbarkeit der entstehenden XML-Dokumente.

Als Ausweg und pragmatischer Kompromiß zwischen eineindeutigen Namenspräfixen und Lesbarkeit wurde daher ein zweistufiges Verfahren eingeführt. Es erlaubt die Zuordnung von URIs zu Präfixen. Dieser Vorgang wird als „Bindung“ bezeichnet.
Diese Präfixes können Attributen oder Elementen vorangestellt werden, um sie in bestimmte

Namensräume zu übernehmen.
Für die Präfixe gelten dieselben Bildungsgesetze wie für die Element- und Attributnamen. Im Einzelnen legt die *Namespace Recommendation* fest: (im XML-Namespace-Dokument nachschlagen)

```
[NS7] Präfix     ::= NCName
[NS4] NCName     ::= (Letter | '_') (NCNameChar)*
[NS5] NCNameChar ::= Letter | Digit | '.' | '-' | '_'
                   | CombiningChar
                   | Extender
```

Anmerkung: Die rechten Seiten der Produktionen beziehen sich entweder auf die dargestellten Definitionen des Namespace-Standards oder auf Syntaxregeln der XML-Recommendation.

Die Bindung einer URI an ein -- gemäß Produktion NS7 frei wählbares -- Präfix geschieht durch das reservierte Attribut `xmlns`.
Die Syntax hierfür wird mit

```
[NS2] PräfixedAttName ::= 'xmlns:' NCName
```

angegeben.

Nach der Bindung der URI an das Präfix kann dieses jedem Element oder Attribut vorangestellt werden, um es in den Namensraum zu übernehmen.
Hierdurch verändert sich die Produktion `Name` aus der XML-Spezifikation zum *qualifizierten Namen*, der durch die Voranstellung des Präfixes entsteht. Der rechts vom trennenden Doppelpunkt folgende Elementname stellt den lokalen Namen (innerhalb des Namensraumes dar). Dieser lokale Name darf *keinen* Doppelpunkt mehr enthalten; insofern schränkt Produktion NS8 in Verbindung mit NS4 die Festlegung der Produktion 5 der XML-Spezifikation ein.

```
[NS6] QName     ::= (Präfix ':')? LocalPart
[NS8] LocalPart ::= NCName
```

Während der Verarbeitung eines XML-Dokuments, das Namensräume nutzt, ersetzt ein XML-Prozessor jedes Auftreten eines deklarierten Präfixes transparent durch die gebundene URI.
Prozessoren, welche die Namensraum-Spezifikation unterstützen, werden als *namespace aware* bezeichnet. Alle anderen Prozessoren treffen die durch NS6 eingeführte Unterscheidung zwischen *Präfix* und *LocalPart* eines qualifizierten Namens nicht und betrachten die Kombination aus Präfix und Element- bzw. Attributnamen als Bezeichner. Die Präfix-URI-Bindung durch das `xmlns:...`-Attribut wird hierbei als gewöhnliches XML-Attribut betrachtet und führt daher zu keinen Validierungsfehlern. (Die Einschränkung der Produktion 5, ein Name dürfe nicht mit der Zeichenfolge `(('X'|'x') ('M'|'m') ('L'|'l'))` beginnen, stellt in der XML-Spezifikation lediglich einen Hinweis dar.)

Semantisch bildet die durch `xmlns` eingeleitete Deklaration ein *Pseudoattribut*, da es für die maschinelle Verarbeitung vorbehalten und mit festelegter Bedeutung ausgestattet ist, welche durch den XML-Dokumentautor nicht verändert werden kann.
Zusätzlich werden Namensraumdeklarationen durch Programmiersprachenschnittstellen nicht den gewöhnlichen Attributen gleichgestellt betrachtet, sondern nehmen, wie auch im Information Set, dort eine Sonderstellung ein.

Anmerkung: Auf Webseiten und in Mailinglisten finden sich manchmal Formulierungen der Struktur `{`*`namespaceName`*`}`*`elementName`* (z.B. `{http://www.w3.org/2001/XMLSchema}element` oder `{http://www.w3.org/1999/XSL/Transform}template`).
Hierbei handelt es sich um eine zwar geläufige, aber *nicht spezifikationskonforme Schreibweise*!
Sie dient lediglich dazu, das prinzipiell beliebig wählbare Präfix einzusparen und den gewählten Namensraum hervorzuheben.
Strukturen dieses Stils sind jedoch keine gültigen XML-Dokumente!

Angewendet auf das betrachtete Beispiel läßt sich die URI `http://www.example.com/sales` an das Präfix `myNS1` binden. Diese Bindung steht im definierenden Element (local name: `rechnung`) und allen untergeordneten zur Verfügung.

Beispiel 10: Dokument mit W3C-konformen Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<myNS1:Rechnung xmlns:myNS1="http://www.xyz.com/sales">
(3)   <myNS1:Kunde>
(4)      <myNS1:KundenNr>4711</myNS1:KundenNr>
(5)      <myNS1:Name>Max Mustermann</myNS1:Name>
(6)      <myNS1:Anschrift>
(7)         <myNS1:Straße>Musterplatz 1</myNS1:Straße>
(8)         <myNS1:PLZ>12345</myNS1:PLZ>
(9)         <myNS1:Ort>Musterstadt</myNS1:Ort>
(10)      </myNS1:Anschrift>
(11)   </myNS1:Kunde>
(12)   <myNS1:Rechnungsposten>
(13)    <!--...-->
(14)    </myNS1:Rechnungsposten>
(15)</myNS1:Rechnung>
```

Download des Beispiels

Hinweis: Für das Attribut `xmlns` kann keine Namensraumdeklaration angegeben werden; es ist spezifikationsgemäß an keinen Namensraum gebunden.

Die Deklaration des Namensraumes mit der Präfixbindung kann auf beliebige hierarchisch höhergeordnete Elemente ausgelagert werden. In der Praxis hat es sich aus Übersichtlichkeitsgründen durchgesetzt, alle in einem XML-Dokument benutzten Namensräume mit ihren Präfixen zu Beginn des Dokuments im Wurzelelement zu definieren.
Das nachfolgende Beispiel zeigt dies anhand eines XHTML-Dokuments, das neben Elementen der Hypertextsprache auch mathematische Formeln und Vektorgraphiken enthält.

Beispiel 11: Ein XHTML-Dokument mit MathML- und SVG-Inhalten

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml"
(3)            xmlns:mml="http://www.w3.org/TR/REC-MathML"
(4)            xmlns:svg="http://www.w3.org/2000/svg">
(5)     <xhtml:head>
(6)             <xhtml:title>XHTML Dokument, mit MathML- und SVG-Inhalten</xhtml:title>
(7)     </xhtml:head>
(8)     <xhtml:body>
(9)             <xhtml:h1>Eine Û¢erschrift</xhtml:h1>
(10)            <mml:math>
(11)            <mml:mrow>
(12)                    <mml:mi>x</mml:mi>
(13)                    <mml:mo>=</mml:mo>
(14)                    <mml:mfrac>
(15)                            <mml:mrow>
(16)                                    <mml:mrow>
(17)                                            <mml:mo>-</mml:mo>
(18)                                            <mml:mi>b</mml:mi>
(19)                                    </mml:mrow>
(20)                                    <mml:mo>&PlusMinus;</mml:mo>
(21)                                    <mml:msqrt>
(22)                                            <mml:mrow>
(23)                                                    <mml:msup>
(24)                                                            <mml:mi>b</mml:mi>
(25)                                                            <mml:mn>2</mml:mn>
(26)                                                    </mml:msup>
(27)                                                    <mml:mo>-</mml:mo>
(28)                                                    <mml:mrow>
(29)                                                            <mml:mn>4</mml:mn>
(30)                                                            <mml:mo>&InvisibleTimes;</
mml:mo>
(31)                                                            <mml:mi>a</mml:mi>
(32)                                                            <mml:mo>&InvisibleTimes;</
mml:mo>
(33)                                                            <mml:mi>c</mml:mi>
(34)                                                    </mml:mrow>
(35)                                            </mml:mrow>
(36)                                    </mml:msqrt>
(37)                            </mml:mrow>
(38)                            <mml:mrow>
(39)                                    <mml:mn>2</mml:mn>
(40)                                    <mml:mo>&InvisibleTimes;</mml:mo>
(41)                                    <mml:mi>a</mml:mi>
(42)                            </mml:mrow>
(43)                    </mml:mfrac>
```

```
(44)            </mml:mrow>
(45)            </mml:math>
(46)            <svg:svg width="4cm" height="8cm">
(47)                    <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(48)            </svg:svg>
(49)    </xhtml:body>
(50)</xhtml:html>
```

Download des Beispiels

Definition 8: Namensraumidentifikation
Jeder XML-Namensraum wird durch eine gültige URI identifziert. Diese URI dient ausschließlich der Benennung, daher muß sie nicht auf eine gültige Ressource verweisen.

Überschreiben des Vorgabe-Namensraums:
Aus den Beispielen ist leicht ersichtlich, daß die explizite Angabe des definierten Präfixes für jedes Element eines Namensraumes platzraubend und für die Zuordnung aller Elemente eines Teilbaumes zum selben Namensraum redundant und -- wegen des zusätzlichen Spezifikationsaufwandes -- unpraktikabel ist. Die mehrmalige explizite redundante (identische) Angabe des identifizierenden Präfixes bildet zusätzlich noch eine potentielle Fehlerquelle hinsichtlich Übertragungsfehlern und reiner Tippfehler bei manuell erstellten XML-Dokumenten.

Eine einfache Kompaktifizierungsvariante greift auf die aus den Programmiersprachen geläufigen Regeln für Namensräume zurück. Dort beinhaltet ein explizit geöffneter Block alle enthaltenen Elemente bis zum Blockendesymbol und faßt sie so zu einem Gültigkeitsbereich zusammen.
Dieses Prinzip läßt sich leicht auch auf XML-Dokumente, die immer eine streng hierarchische Baumstruktur aufweisen, anwenden.

Hierzu wird das `xmlns`-Attribut leicht modifiziert eingesetzt. Wird es *ohne nachfolgendes Präfix* und unter Weglassung des separierenden Doppelpunktes verwendet, so definiert es einen *Vorgabenamensraum* (*default namespace*). Dieser umfaßt neben dem Element, welches das Attribut beinhaltet, auch alle Kindelemente. Eine Ausnahme hiervon bilden untergeordnete Elemente, die explizit durch Präfix oder Redefinition des Vorgabenamensraumes einem anderen Namespace zugeordnet werden.

Das nachfolgende Beispiel zeigt dies für das bereits mit Namenräumen versehene Rechnungsdokument

Die syntaktische Definitionsform der Namensraumüberschreibung als XML-(Pseudo-)Attribut stellt hierbei sicher, daß für ein Element keine mehrmalige Überschreibung des Vorgabenamensraumes vorgenommen werden kann, da in diesem Falle das Attribut `xmlns` mehrfach im selben Elementkontext auftreten müßte, was der XML-Basisspezifikation widerspräche.

Beispiel 12: Rechnungsdokument mit überschriebenem Vorgabenamensraum

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung xmlns="http://www.xyz.com/sales">
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)     </Anschrift>
(11)  </Kunde>
(12)  <Rechnungsposten>
(13)           <!--...-->
(14)   </Rechnungsposten>
(15)</Rechnung>
```

Download des Beispiels

Durch die Definition des Vorgabenamensraumes für das Element `rechnung` und all dessen Kindelemente wird derselbe Effekt erreicht wie durch die Präfixangabe im vorangegangenen Beispiel.
Diese Schreibweise stellt lediglich eine Abkürzung der expliziten Qualifizierung jedes einzelnen XML-Namens dar. Insbesondere führt die mehrmalige Redefinition des Vorgabenamensraumes *nicht* zu kaskadierten Namensräumen. Jeder Namensraum ist von allen umgebenden unabhängig definiert.
So kann das Dokument des XHTML-Beispiels auch dahingehend verändert werden, daß die Namensräume erst an der Stelle im Dokument deklariert werden, an der sie auch benötigt werden.

Beispiel 13: Ein XHTML-Dokument mit MathML- und SVG-Inhalten, unter Verwendung überschriebener Vorgabenamensräume

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<html xmlns="http://www.w3.org/1999/xhtml">
(3)     <head>
(4)             <title>XHTML Dokument, mit MathML- und SVG-Inhalten</title>
(5)     </head>
(6)     <body>
(7)             <h1>Eine Überschrift</h1>
(8)             <math xmlns="http://www.w3.org/1998/Math/MathML">
(9)             <mrow>
(10)                    <mi>x</mi>
(11)                    <mo>=</mo>
(12)                    <mfrac>
(13)                            <mrow>
(14)                                    <mrow>
(15)                                            <mo>-</mo>
(16)                                            <mi>b</mi>
(17)                                    </mrow>
(18)                                    <mo>+-</mo>
(19)                                    <msqrt>
(20)                                            <mrow>
(21)                                                    <msup>
(22)                                                            <mi>b</mi>
(23)                                                            <mn>2</mn>
(24)                                                    </msup>
(25)                                                    <mo>-</mo>
(26)                                                    <mrow>
(27)                                                            <mn>4</mn>
(28)                                                            <mo> </mo>
(29)                                                            <mi>a</mi>
(30)                                                            <mo> </mo>
(31)                                                            <mi>c</mi>
(32)                                                    </mrow>
(33)                                            </mrow>
(34)                                    </msqrt>
(35)                            </mrow>
(36)                            <mrow>
(37)                                    <mn>2</mn>
(38)                                    <mo> </mo>
(39)                                    <mi>a</mi>
(40)                            </mrow>
(41)                    </mfrac>
(42)            </mrow>
(43)            </math>
(44)            <svg xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/
svg" svg:width="4cm" svg:height="8cm">
(45)                    <ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(46)            </svg>
(47)    </body>
(48)</html>
```

Download des Beispiels

Die Namensraumpräfixe können durch den Anwender frei vergeben werden. Sie dienen lediglich der abkürzenden Schreibweise und sind für die Namensraumauflösung unerheblich.
Daher werden zwei Elemente oder Attribute als gleich betrachtet, wenn sie lexikalisch in Namen und Namensraumidentifier übereinstimmen. Hierbei ist es unerheblich, ob der Namensraum explizit durch Präfixangabe oder durch Überschreiben des Vorgabenamensraumes definiert wurde.
Die Elemente der XML-Dokumente aus den Beispielen 14 und 15 befinden sich alle ausnahmslos im Namensraum `http://www.example.com`.

Beispiel 14: Namensraumpräfixe 1

```
(1)<abc:ElementA xmlns:abc="http://www.example.com"
(2)                             xmlns:xyz="http://www.example.com">
(3)     <ElementB xmlns="http://www.example.com">
(4)             <ElementC/>
(5)     </ElementB>
(6)     <xyz:ElementB>
(7)             <abc:ElementC/>
(8)     </xyz:ElementB>
(9)</abc:ElementA>
```

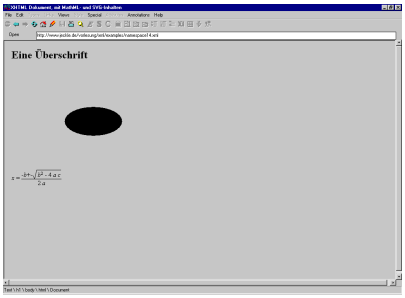
Download des Beispiels

Beispiel 15: Namensraumpräfixe 2

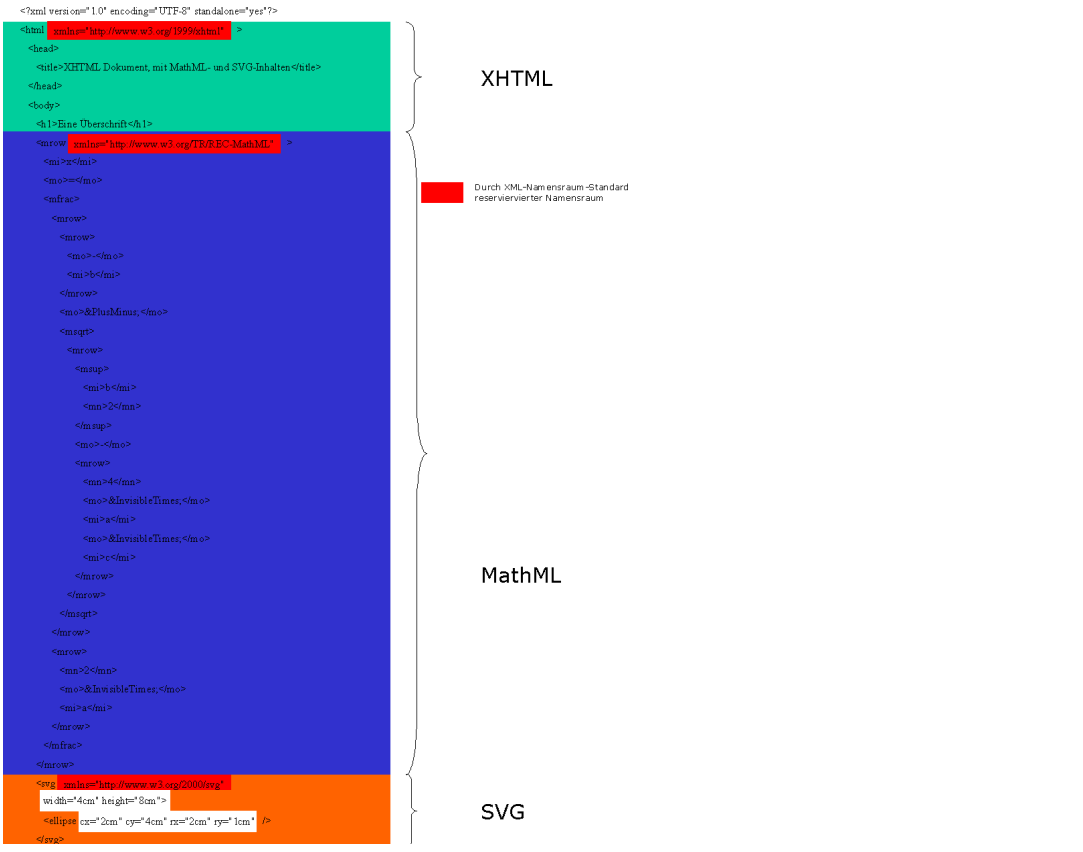
```
(1)<ElementA xmlns="http://www.example.com"
(2)             xmlns:myNamespace="http://www.example.com">
(3)   <foo:ElementB xmlns:foo="http://www.example.com">
(4)      <myNamespace:ElementC/>
(5)   </foo:ElementB>
(6)   <ElementB xmlns="http://www.example.com">
(7)      <myNamespace:ElementC/>
(8)   </ElementB>
(9)</ElementA>
```

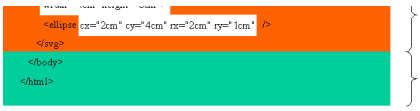
Download des Beispiels

Die Abbildung zeigt das Beispieldokument in der Darstellung des W3C-Browsers Amaya.



Im Beispieldokument wird der Vorgabenamensraum dreimal, entsprechend der verschiedenen verwendeten XML-Sprachen, neu gesetzt. So wird auf `html` und alle direkt untergeordneten Elemente der URI-identifizierte Namensraum `http://www.w3.org/1999/xhtml` angewendet. head, `title` und body sowie dessen Kindelemente finden sich demnach, da sie keinen eigenen Namensraum definieren, ebenfalls im so definierten Vorgabenamensraum.
`mrow` als hierarchisch tieferstehendes Element redefiniert den Namensraum zu `http://www.w3.org/TR/REC-MathML`. Daher werden das Element `mrow` sowie all dessen Kindelemente (im Beispiel: `ellipse`) auch diesem zugeordnet.
Die Attribute `width`, `height`, `cx` , ... verfügen über kein explizites Namensraumpräfix und sind daher dem leeren Namensraum zugeordnet.
Auf den MathML-Namensraum folgend wird der Vorgabenamensraum zu `http://www.w3.org/2000/svg` redefiniert. Auch hier gelten dieselben Regeln, d.h. der überschriebene Vorgabenamensraum erstreckt sich auf alle Kindelemente.
Mit dem schließenden Tag svg endet auch dessen Namensraum. Alle folgenden Elemente befinden sich wieder im umgebenden Namensraum, der zu Beginn des Dokuments mit `http://www.w3.org/1999/xhtml` festgelegt wurde.
Die nachfolgende Graphik stellt die Namensräume nochmals farblich hervorgehoben dar.
Ein weiteres Beispiel findet sich in der Namespace-Recommendation.





Der XML-Namensraumstandard des W3C sieht die beiden im Vorhergehenden diskutierten Varianten exklusiv zueinander vor. D.h. für ein Element, welchem bereits durch Präfixangabe eine Namensraumzuordnung gegeben wurde, kann nicht zusätzlich der Vorgabenamensraum überschrieben werden. Deklarationen der Form `<xyz:abc xmlns="..." ...>` sind widersprüchlich; und daher illegal. (in der XML-Namespace Recommendation nachschlagen)

Das abschließende Beispiel 16 zeigt die Verwendung zweier Vokabulare (SVG und MathML), die beide ein mit `set` benanntes Element definieren.
Durch die Deklaration der jeweiligen Namensräume unterscheiden sich die qualifizierten Namen, die dem (gleichnamigen) Elementnamen die Namensraum-URI voranstellen.

Beispiel 16: Namensräume im realen Einsatz

```
(1)<?xml version="1.0"?>
(2)<document>
(3)     <svg xmlns="http://www.w3.org/2000/svg">
(4)             <g transform="translate(100,100)">
(5)                     <text id="TextElement" x="0" y="0" style="font-family:Verdana;
font-size:35.27; visibility:hidden">
(6)                     It's alive!
(7)                             <set attributeName="visibility" attributeType="CSS"
to="visible" begin="3s" dur="6s" fill="freeze"/>
(8)                     </text>
(9)             </g>
(10)    </svg>
(11)
(12)    <math xmlns="http://www.w3.org/1998/Math/MathML">
(13)            <set>
(14)                    <ci> b </ci>
(15)                    <ci> a </ci>
(16)                    <ci> c </ci>
(17)            </set>
(18)    </math>
(19)</document>
```

Download des Beispiels

Präzedenz des explizit zugeordneten Namensraumes:
Eine explizit durch Präfixzuordnung vorgenommene Namensraumfestlegung besitzt Präzedenz gegenüber dem evtl. überschriebenen Vorgabenamensraum.
Findet daher für ein Element sowohl die Überschreibung des Vorgabenamensraumes, als auch gleichzeitig die Namensraumfestlegung durch explizite Präfixzuordnung statt, so wird das Element demjenigen Namensraum zugeordnet, der durch die URI identifiziert wird, an den das Präfix gebunden ist.
Dies gilt insbesondere auch dann, wenn ein und dasselbe Element sowohl über ein Präfix, als auch eine Überschreibung des Vorgabenamensraumes verfügen.
Das XML-Dokument aus 17 illustriert dies beispielhaft. So wird `ElementA` -- durch Überschreibung des Vorgabenamensraumes -- dem Namensraum `urn:namspaces:Namespace1` zugeordnet und diese Festlegung auch an das Kindelement `ElementB` weitergegeben.
Das Kindelement `ElementC` hingegen überschreibt die Vorgabe des Elternelements durch explizite Präfixangabe und ist daher dem durch `urn:namespace:Namespace2` identifizierten Namensraum zugeordnet.
Für `ElementD` findet sich sowohl eine Namensraumdefinition, welche durch Überschreiben des Vorgabenamensraumes zu `urn:namespace:Namespace3` stattfindet, als auch eine Präfix-gebundene Definition an den Namensraum `urn:namespace:Namespace2`. Gemäß der Präzedenz der expliziten Festlegung durch Präfix wird `ElementD` jedoch ausschließlich dem Namensraum zugeordnet, an den das angegebene Präfix `ns1` gebunden ist. Im Beispiel ist dies die URI `urn:namespace:Namespace2`.

Beispiel 17: Präzedenzregel

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ElementA xmlns="urn:namspaces:Namespace1"
(3)                     xmlns:ns1="urn:namespace:Namespace2"
(4)                     xmlns:ns2="urn:namespace:Namespace3">
(5)     <ElementB/>
(6)     <ns1:ElementC/>
(7)     <ns1:ElementD xmlns="urn:namespace:Namespace3"/>
(8)</ElementA>
```

Download des Beispiels

Aufheben der Namensraumzuweisung:
Durch Überschreibung des Vorgabenamensraumes mit der Zeichenkette leeren Inhalts -- formal der Zuweisung der leeren URI als Namensraumidentifikator -- kann eine bestehende Namensraumdefinition aufgehoben werden. Als Resultat entsteht eine Situation identisch zu einem Dokument ohne festgelegte Namensräume, d.h. die Elemente finden sich im leeren Namensraum.

Beispiel 18: Aufheben von Namensraumdeklarationen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Adressen>
(3)     <table xmlns="http://www.w3.org/TR/REC-html40">
(4)             <tr>
(5)                     <td>Name</td>
(6)                     <td>Adresse</td>
(7)             </tr>
(8)             <tr>
(9)                     <td>
(10)                            <Vorname xmlns="">Max</Vorname>
(11)                       <Nachname xmlns="">Mustermann</Vorname>
(12)                    </td>
(13)                    <td>
(14)                            <Straße xmlns="">Musterstr. 1</Straße>
(15)                            <PLZ xmlns="">12345</PLZ>
(16)                            <Ort xmlns="">Musterstadt</Ort>
(17)                    </td>
(18)            </tr>
(19)    </table>
(20)</Adressen>
```

Das Beispiel 18 zeigt die notwendigen Deklarationen zur Aufhebung der Vorgabenamensraumdefinition. So wird zwar für das Element `table` und alle seine Kindelemente der Vorgabenamensraum auf `http://www.w3.org/TR/REC-html40` gesetzt, dies jedoch für die Kindelemente `Vorname`, `Nachname`, `Straße`, `PLZ` und `Ort` durch die Festlegung `xmlns=""` explizit für das jeweilige Element aufgehoben.

Die Aufhebung von definierten Namensräumen kann ausschließlich durch die Überschreibung des Vorgabenamensraum erfolgen. Eine Bindung der leeren URI an ein Präfix zur späteren Verwendung ist nicht zugelassen.

Namensräume für Attribute:
Abweichend von der Mimik für Elemente, dort wirkt sich ein überschriebener Vorgabenamensraum auch immer auf die Kindelemente aus, wird eine Namensraumdeklaration auf Elementebene nicht auf Attribute propagiert.
Diese Festlegung der Spezifikation mag insbesondere unter Kenntnis der Baumstruktur der Infosets, welche Attribute und Elemente gleichermaßen als Kindknoten der beherbergenden Elementinformationseinheit darstellt, verwundern. Eine mögliche Begründung dieser Asymmetrie mag in der besonderen Rolle der Attribute zur Informationsdarstellung liegen. So wird teilweise damit argumentiert, daß Attribute üblicherweise unabhängig vom aktuell umgebenden Element sein sollten und daher nur zur Darstellung von Daten herangezogen werden sollten, die nicht über einen direkten Bezug zum sie umgebenden Element verfügen.
In der Konsequenz müssen Attribute immer explizit mit einem Namensraumpräfix versehen werden, um sie einem Namensraum zuzuordnen.
Beispiel 19 zeigt die Anwendung der Namensräume auf Attribute. So befinden sich weder das Attribute `att1` des Elements `ElementB`, noch dasjenige von `ElementD` in einem Namensraum. Das mit dem Wert `XYZ` versehene Attribut `att2` des Elements `ElementC` wird hingegen -- aufgrund des explizit angegebenen Präfixes -- dem Namensraum `http://www.example.com/NS2` zugeordnet.
Ferner illustriert `ElementC` die Rolle der Namensräume als Bestandteil des identifzierenden Namens von Elementen und Attributen. Aufgrund der Interpretation des Namensraumes als Benennungsbestandteil darf das `att2` benannte Attribut mehrfach auftreten, da die Zuhilfenahme des Namensraumes die eindeutige Identifikation gestattet.

Beispiel 19: Namensräume für Attribute

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Wurzelelement>
(3)     <ElementA xmlns:NS1="http://www.example.com/NS1" xmlns:NS2="http://www.example.com/
NS2">
(4)             <ns2:ElementB att1="...">
(5)                     <ElementD att1="..." xmlns="http://www.example.com/NS3">
(6)                             <ElementC att2="ABC" NS2:att2="XYZ"/>
(7)                     </ElementD>
(8)             </ns2:ElementB>
(9)     </ElementA>
(10)</Wurzelelement>
```

Definition 9: Namensraumvererbung

Namensräume, die durch Überschreiben des Vorgabenamensraumes zugewiesen werden wirken sich ausschließlich auf Elemente und deren direkte oder transitive Kindelemente aus, sofern diese den Namensraum nicht wieder verändern.
Namensräume, die durch explizite Präfixangabe zugewiesen werden, wirken sich ausschließlich auf dasjenige Element aus vor dessen Name das Präfix plaziert ist.
Namensräume für Attribute werden ausnahmslos durch explizite Präfixangabe festgelegt und gelten ausschließlich für das Attribut selbst.

Ausgehend von der Vererbungsregel für Namensräume, sowie der Präzedenz expliziter Präfixangaben lassen sich daher folgende Auswertungsregeln definieren:

Ein Element befindet sich in demjenigem Namensraum ...

1. ... an den das vorangestellte Präfix gebunden ist.
   Verfügt das Element über kein Namensraumpräfix, so befindet es sich in demjenigen Namensraum ...
2. ... der auf diesem Element durch Überschreibung des Vorgabenamensraumes definiert wurde.
   Findet für dieses Element keine Überschreibung des Vorgabenamensraumes statt, so befindet es sich in demjenigen Namensraum ...
3. ... der für das Elternelement gilt, sofern er dort Vorgabenamensraum ist.
   Man beachte: Das *gilt* im vorangehenden Satz umschließt sich nicht nur die Überschreibung des Vorgabenamensraumes im direkten Elternelement, sondern auch eine dort geltende Namensraumüberschreibung die in dessen Elternelement oder dessen Elternelement ... stattfand.
   Findet in keinem der Elternelemente eine Überschreibung des Vorgabenamensraumes statt, so befindet sich das Element in demjenigen Namensraum ...
4. ... der leer ist (d.h. im leeren Namensraum).

Ein Attribut befindet sich in demjenigem Namensraum, der durch explizite Präfixangabe festelegt wurde.

Internationale URIs und Namensraumidentifikatoren:
Die Berücksichtigung von Zeichen, die in XML v1.1 zugelassenen, deren Nutzung in den klassischen URIs nach RFC 2396 bzw. RFC 2732 jedoch untersagt ist, führt zur Einführung des neuen Begriffes des *Internationalized Resource Identifier*s (IRI). Diese Neuschöpfung stellt im Kern eine URI-Fassung dar innerhalb der Leerzeichen sowie diverse Sonderzeichen zulassen sind. Diese internationalisierten Identifikatoren werden durch einen im Spezifikationsentwurf festgelegten Algorithmus in syntaktisch korrekte URIs umgewandelt.
Beispiel 20 zeigt gültige IRIs und jeweils dahinter in Klammern angegeben die daraus resultierende URI-Darstellung.

Beispiel 20:

```
(1)http://www.{iri-}example.com (http://www.%7Biri-%7Dexample.com)
(2)mailto:marc léon@example.org (mailto:marc%20l%E9on@example.org)
```

Kompatibilität zu älteren Dokumenten:
Elemente, für die weder ein expliziter Namensraum durch Präfix definiert ist, noch ein Namensraum von einem Elternelement übernommen werden kann, sind einem leeren Namensraum zugeordnet; konzeptionell entspricht dies einem NULL-Präfix.
Somit befinden sich alle Elemente, die keinem Namensraum angehören, automatisch in einem gemeinsamen Namensraum, der an keine URI gebunden ist.

Zusammenfassend gelten somit folgende Prinzipien:

- Jede beliebige URI kann an eigendefinierte Präfixe gebunden werden.
  Insbesondere kann dieselbe URI an verschiedene Präfixe gebunden werden, und dasselbe Präfix in verschiedenen Kontexten für verschiedene URIs stehen.
- Jedes Element übernimmt den überschriebenen Vorgabenamensraum seines Elternelements, sofern es keinen eigenen definiert.
- Elemente oder Attribute ohne Namensraumzuordnung (die auch keine von ihrem hierarchisch höherstehenden Element übernehmen) befinden sich im Standardnamensraum.
- Attribute ohne Namensraumprefix befinden sich im leeren Namensraum.

Web-Referenzen 6: Weiterführende Links

- XML-Namespace Recommendation
- Namespace Recommendation in deutscher Übersetzung
- Namespace Tutorial @ Zvon.org
- Tim Bray: Namespaces by Example
- Hintergrundartikel: Namespaces in XML Adopted by W3C
- (Tutorial) Simon St. Laurent: Namespaces in XML
- Roland Bourret: XML Namespaces FAQ

2.3 XML-Schema

XML Schema

Neben den in der Vergangenheit zur Sprachdefinition verwendeten Document Type Definitions ist in jüngerer Zeit ein alternativer Ansatz in den Blickpunkt des Interesses gerückt: die XML-Schemasprachen. Sie setzen die Emanzipation der Metasprache XML von ihrer Vorgängersprache SGML fort. Bereits in engem zeitlichem Bezug zur Veröffentlichung der XML-Recommendation wurde mit *XML Data* ein erster Ansatz vorgestellt. In der Zwischenzeit fanden verschiedene konkurrierende Vorschläge ein breites Interesse. Übereinstimmende Zielsetzung aller verschiedenen vorgeschlagenen Schemasprachen ist die Schaffung eines Sprachdefinitionsmechanismus, der die Dokumenten-orientierten Strukturen und Inhaltsmodelle der DTD überwindet.
An die Spitze der Bemühungen setzte sich eine Arbeitsgruppe des W3C zur Definition einer XML-Schemasprache, unter Berücksichtigung der bekanntesten und verbreitetsten Vorschläge. Durch sie wurde im Mai 2001 der *XML Schema*-Standard des W3C veröffentlicht.

Der Begriff *Schema* ist der im Datenbankumfeld gebräuchlichen Terminologie entlehnt. Dort bezeichnet er Informations- oder Datenmodelle als Konstruktionsvorlage oder Dokumentation eines Datenbankdesigns. Hierzu muß ein Schema nicht unbedingt in einer graphischen Datenmodellierungssprache vorliegen, sondern kann beispielsweise auch die Tabellenstruktur einer relationalen Datenbank bezeichnen.

Zur Notwendigkeit einer Schemasprache:
Zum Zeitpunkt der Konzeption der Metasprache SGML war das Anwendungsfeld klar umrissen und im wesentlichen auf die Digitalisierung vormals papiergestützter Dokumentation festgelegt. Daraus erklärt sich auch die Mächtigkeit der Document Type Definition, der angebotenen Grammatiksprache zur Darstellung der Dokumentstrukturen.
Insbesondere war weder die Daten-orientierte Verwendung von SGML, noch die rund 30 Jahre später einsetzende Weiterentwicklung (eigentlich: Reduktion) zur eXtensible Markup Language abzusehen.
Die inzwischen eingesetzte breite Anwendung von XML-Sprachen zur Darstellung beliebiger Inhalte läßt jedoch die Beschränkungen und Unzulänglichkeiten des DTD-Mechanismus für diesen Anwendungen offenkundig werden.

Nachfolgend sind einige der durch Nutzung des DTD-Mechanismus zur Beschreibung Daten-intensiver Strukturen induzierten Einschränkungen zusammengestellt:

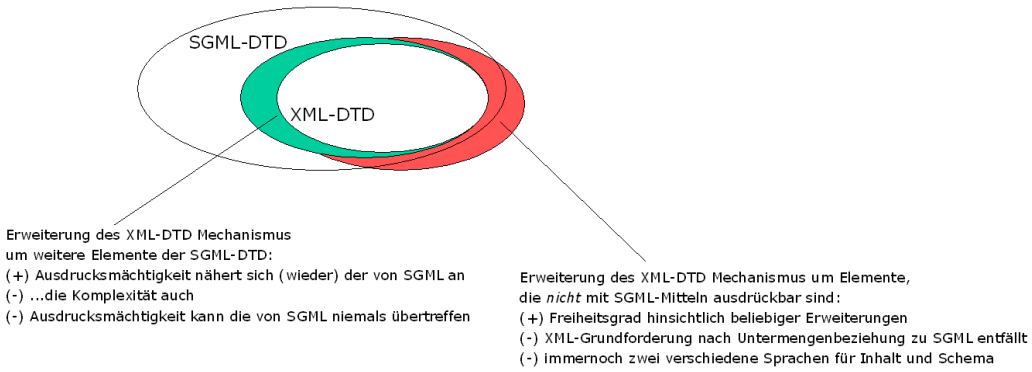
- Unzureichende Datentypunterstützung.
  DTDs erlauben für Elemente nur die vier Inhaltsmodelle: `child elements`, `PCDATA`, *mixed content* sowie das leere Inhaltsmodell `EMPTY`.
- Unzureichende Strukturierungsunterstützung.
  Zwar erlauben die Operatoren zur Steuerung der Auftrittshäufigkeit („?“, „+“ und „*“) einzelner Kindelemente die Codierung beliebiger Kardinalitäten. Allerdings sind die hierfür anzuwendenden Prinzipien teilweise umständlich in der Schreibung, und daher fehlerträchtig. Die Lesbarkeit der entstehenden DTD wird entscheidend gesenkt.
- Keine Unterstützung von Wiederverwendbarkeit.
  Während Elementstrukturen (zumindest) innerhalb der definierenden DTD beliebig wiederverwendet werden können, sind Attribute immer an das umgebende Element gebunden .
  Eine Nutzung in anderen Dokument-Typ-Definitionen als der definierenden ist nicht vorgesehen.
- Starres Typsystem.
  Das angebotenen Typsystem kann durch den Anwender nicht erweitert werden.
- Keine Unterstützung von Namensräumen.
  Namensräume können in der DTD nicht angegeben werden.
- Nur rudimentärer Referenzierungsmechanismus.
  Die offerierten Verknüpfungsmechanismen sind ausschließlich Dokument-lokal möglich und gestatten keine Differenzierung hinsichtlich der Semantik des eindeutig identifizierten oder referenzierten Elements.
- DTD-Syntax ist nicht XML.
  Die von SGML übernommene Syntax der DTD bildet eine eigene Sprache, die von Werkzeugen gesondert zu implementieren ist.

Technische Ansätze:
Prinzipiell lassen sich die in der Vergangenheit vorgeschlagenen Ansätze zur Definition einer Schemasprache in vier Kategorien unterscheiden:

1. Orientierung am bestehenden DTD-Mechanismus.
   Erweiterungen des bestehenden Mechanismus um zusätzliche Sprachelemente.
2. Orientierung an der programmiersprachlichen Interpretation.
   Versuch XML und ein Ausführungsmodell möglichst eng zu koppeln.
3. Orientierung an Wissensdarstellungen
   Interpretation des Schemas einer XML-Sprache als Wissen über die Sprache.
4. XML-Sprachen zur Inhaltsbeschreibung.
   Da XML i.A. zur Beschreibung beliebigster Informationen herangezogen werden kann, ist die Verwendung auch für die Beschreibung von XML-Strukturen denkbar.

Die naheliegendste Option dürfte die Erweiterung des bestehenden DTD-Sprachumfanges bilden. Durch geeignete Modifikationen und Ergänzungen ließen sich alle, mit Ausnahme der letzten, identifizierten Unzulänglichkeiten beheben.
Konzeptionell lassen sich zwei Erweiterungsvarianten aufzeigen. Zunächst die Möglichkeit, die XML-DTDs um Elemente der ursprünglichen SGML-DTD zu erweitern. In der Konsequenz nähert sich XML, positiv formuliert, wieder der Ausdrucksmächtigkeit der Ursprache SGML an. Negativ formuliert, kann jedoch XML auf diesem Wege niemals Inhaltsstrukturen ausdrücken, die nicht durch SGML ausdrückbar sind, da die Mächtigkeit des SGML-DTD-Mechanismus eine natürliche Obergrenze der Erweiterbarkeit darstellt.
Zusätzlich ist anzumerken, daß ein solcher Ansatz der ursprünglichen Intention der XML-Entwicklung -- ein leichter einsetzbares SGML zu schaffen -- entgegenläuft.
Eine der bekannten Ideen zur Erweiterung des DTD-Mechanismus stellt *Datatypes for DTDs* (DT4DTD) dar.
Alternativ zur Erweiterung hin zur SGML-Mächtigkeit ließe sich der bestehende XML-DTD-Mechanismus um neue zusätzliche Konstrukte anreichern, die nicht Bestandteil der SGML-DTD-Syntax sind. Dieser Ansatz böte den Vorteil, den Vorgängerstandard nicht berücksichtigen zu müssen und beliebige Erweiterungen in Syntax und Semantik einbringen zu können. Allerdings würde damit eine zentrale Forderung der XML-Entwicklung, die sich bereits im Abstract der XML-Recommendation findet, nicht berücksichtigt: die Untermengenbeziehung zu SGML. Durch eine Erweiterung, welche über die SGML-Mächtigkeit hinausreicht, würden legale (*well formed* und sogar *valid*) XML-Dokumente entstehen, die keine gültigen SGML-Dokumentinstanzen wären.

Die nachfolgende Graphik veranschaulicht die beiden Erweiterungsoptionen und die Argumente der geführten Diskussion.



Die im zweiten Punkt angedeutete Umsetzung ist durch eine programmiersprachliche Verarbeitung der XML-Dokumente motiviert. Aus Sicht dieser Anwendungsfacette ist ein Schemamechanismus idealerweise so ausgelegt, daß er die transparente Umsetzung in Applikationsdatenstrukturen ermöglicht. Dahinter steht der Wunsch, den *impedance mismatch*, mithin den zu leistenden Abbildungsaufwand zwischen XML-Konstrukten und Datenstrukturen, möglichst gering zu halten.
Beispielsweise greift der -- durch den Einsatz im e-Commerce-System der Firma CommerceOne bekannt gewordene -- Vorschlag *Schema for Object-Oriented XML* (SOX) zur Definition der notwendigen Semantik der angebotenen Schemaprimitiven auf die bekannte plattformunabhängige Programmiersprache Java zurück.
Die aktuelle Version der Schemasprache SOX, die zur Definition der XML-Sprache xCBL eingesetzt wird, findet sich unter xCBL.org.

Der dritte technische Ansatz weist auf eine alternative Interpretation der XML-Grammatikstruktur hin. So spiegelt ein Schema auch immer *Wissen* über Struktur und Inhalt eines betrachteten Problembereichs wieder.
Der bekannteste Vorschlag -- die *Document Content Description* (DCD) -- nutzt zur Definition der Wissensstrukturen eines XML-Dokuments das *Resource Description Framework* (RDF) des World Wide Web Consortiums.
Der Ansatz hat sich durch Referenzimplementierungen durchaus als tragfähig und, wegen der RDF-basiertheit, als allgemein verwendbar erwiesen. Jedoch liegt hierin auch die offensichtlichste Limitierung. RDF als Metasprache der Schemasprache legt bereits eine gewisse Strukturierung aller Schemata zugrunde, da jedes gültige DCD-Schema definitionsgemäß ein RDF-Dokument darstellt. Ebenso ist die Semantik der eingesetzten RDF-Elemente bereits durch diese Spezifikation vorgegeben. Beide Punkte zusammengenommen offenbaren eine ausgeprägte Abhängigkeit von den weiteren RDF-Aktivitäten des World Wide Web Consortiums, die bisher nicht auf die Interdependenz von Schemasprache und Wissensbeschreibungsformat ausgerichtet ist.
Positiv fällt an DCD die Verwendung von XML zur Beschreibung von XML-Sprachen auf, womit auch die letzte der erhobenen Anforderungen zu erfüllen wäre.
Die Verknüpfung von RDF mit DCD als Schemasprache birgt allerdings ein potentielles Problem hinsichtlich der Validierbarkeit der entstehenden Strukturen. Durch den Rückgriff von DCD auf RDF entsteht bei der Angabe eines Schemas für RDF ein transitiver Zirkelschluß. In der Konsequenz wird zur Validierung eines XML-Dokuments, welches einer mittels DCD-formulierten Grammatik folgt, neben dem eigentlichen DCD-Schema des Dokuments auch das DCD-Metaschema und dessen Semantik-liefernde RDF-Beschreibung benötigt.

Diese Beschränkung mildert die vierte Familie von XML-Schemasprachen ab. Sie umfaßt die meisten

Vorschläge, die alle als eigenständige XML-Sprachen ausgelegt sind; daher definieren sie ein eigenständiges XML-Vokabular zur Darstellung der benötigten XML-Strukturen, sowie die zugehörige Semantik.
In der Folge sind sie für die Meta-Schemaebene selbstbeschreibend. Das bedeutet das Schema eines Schemas kann durch sich selbst validiert werden. Da dieser Validierungsschritt statisch nur einmal erfolgen muß, kann er durch Schemawerkzeuge vorweggenommen werden.
In dieser Kategorie sind die meisten der bisher vorgeschlagenen Schemadialekte einzuordnen.

Die größte Bedeutung haben kontextfreie reguläre Sprachen zur Spezifikation von XML-Sprachstrukturen erlangt.
Eine Sprache dieses Typs entwickelt auch die W3C-Arbeitsgruppe zur Definition eines XML-Schemasprachstandards. Insbesondere berücksichtigt diese Aktivität explizit die Vorgängersprachen XML Data, DCD, SOX sowie Document Definition Markup Language. Die erwähnten konkurrierenden Vorschläge unterscheiden sich semantisch lediglich in Nuancen, bieten dem Anwender jedoch teilweise (optisch) stark unterschiedliche Konstrukte zur Syntaxspezifikation an.

Einen strukturell unterschiedlichen Ansatz verfolgt die durch Rick Jelliffe vorgeschlagene Sprache *Schematron*. Sie interpretiert ein Schema als Sammlung von Regeln, denen ein gegebenes Dokument genügen muß, um als gültig akzeptiert zu werden. Dies erlaubt die Formulierung mächtiger konktextsensitiver Einschränkungen, die während des Validierungsvorganges geprüft werden.
Die Umsetzung dieser Schemasprache setzt auf den XML-Standards XPath und XSLT auf.

W3Cs XML-Schema:
Jenseits aller existierenden verschiedenen Sprachvorschläge kommt dem W3C-Standard der *XML Schema Description Language* (XSD) die größte praktische Bedeutung zu.
Tim Berners-Lee verkündete in der Eröffnungsrede der WWW-Konferenz in Hong Kong am 2. Mai 2001 die Verabschiedung als Recommendation. Gleichzeitig deutete er bereits weitere Schema-Aktivitäten des World Wide Web Consortiums an.
XML-Schema bildet zusammen mit XML v1.0 2nd edition und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards.

Aus formalen Gründen ist nicht mit dem Ersatz der DTD durch Schema zu rechnen. Jedoch werden mittelfristig neu entwickelte XML-Sprachen keine Grammatiken mehr in der Syntax der DTD entwickeln, sondern direkt Schemata definieren.

XSD bildet eine vollständig in XML-Syntax formulierte kontextfreie reguläre Grammatik zur Formulierung beliebiger XML-Strukturen ab. Hierbei handelt es sich um die bekannten Grundprimitive *Element* und *Attribut*
Gleichzeitig wurde, neben zahlreichen anderen Neuerungen, die Kommentarsyntax für Schemata neu definiert.

Inhaltlich gliedert sich der XSD-Sprachvorschlag in zwei große Teilbereiche: Part 1: Structures zur Definition von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbaren Strukturen und Part 2: Datatypes zur Festlegung diverser inhaltlicher Charakteristika wie Datentypen und konsistenzgarantierende Einschränkungen.
In beiden Teilen werden XML-Namensräume explizit berücksichtigt. Konzeptionell rekonstruiert XSD-Part1 zunächst die bekannte Mächtigkeit der DTD um so die evolutionäre Weiterentwicklung bestehender XML-Sprachen zu ermöglichen.
Der zweite Teil der XSD-Spezifikation definiert ein eigenständiges Typsystem, das neben der naheliegenden Verwendung im ersten Teil der Schemasprache XSD auch in anderen W3C-Arbeitsgruppen Verwendung findet. Inhaltlich baut auch Part2 auf den in der DTD definierten Typen auf und erlaubt zunächst direkt ihre Angabe in Schemata. Darauf aufbauend wird eine Fülle verschiedenster Typen angeboten, die an die verschiedenen verfügbaren Typsysteme aus den Programmiersprachen, Datenbanken und internationalen Standards angelehnt sind.
Alle durch XSD definierten Elemente, d.h. alle Primitive zur Definition eines eigenen Schemas, befinden sich im Namensraum `http://www.w3.org/2001/XMLSchema`, der üblicherweise an das Präfix `xsd` gebunden wird. Elemente und Attribute aus XML-Schema, die in Instanzdokumenten verwendet werden könne sind im Namensraum `http://www.w3.org/2001/XMLSchema-instance` (übliches Präfix `xsi`) organisiert.
Wegen des Umfanges der offiziellen Schemadokumente wird zusätzlich durch das W3C ein Part 0: Primer herausgegeben. Er stellt die beiden XSD-Teile in der Zusammenschau an Beispielen dar.

Schemareferenz:
Jedes XML-Schema bildet als XML-Dokument eine eigenständige Speichereinheit, üblicherweise eine Datei. Die Verbindung zwischen Schema und beschriebenem Dokument wird durch das in der XSD-Spezifikation vordefinierte Attribut `schemaLocation` bzw. `noNamespaceSchemaLocation` definiert. Eines dieser Attribute muß zwingend im Wurzelelement des XML-Dokuments angegeben werden.
Legt das Schema keinen Namensraum für die enthaltenen Deklarationen fest, d.h. alle darin deklarierten Elemente befinden sich im Vorgabenamensraum, so findet sich die Schemareferenz in `noNamespaceSchemaLocation`; andernfalls in `schemaLocation`.
Das nachfolgende Beispiel zeigt die Deklaration:

Beispiel 21: Definition einer Schemareferenz

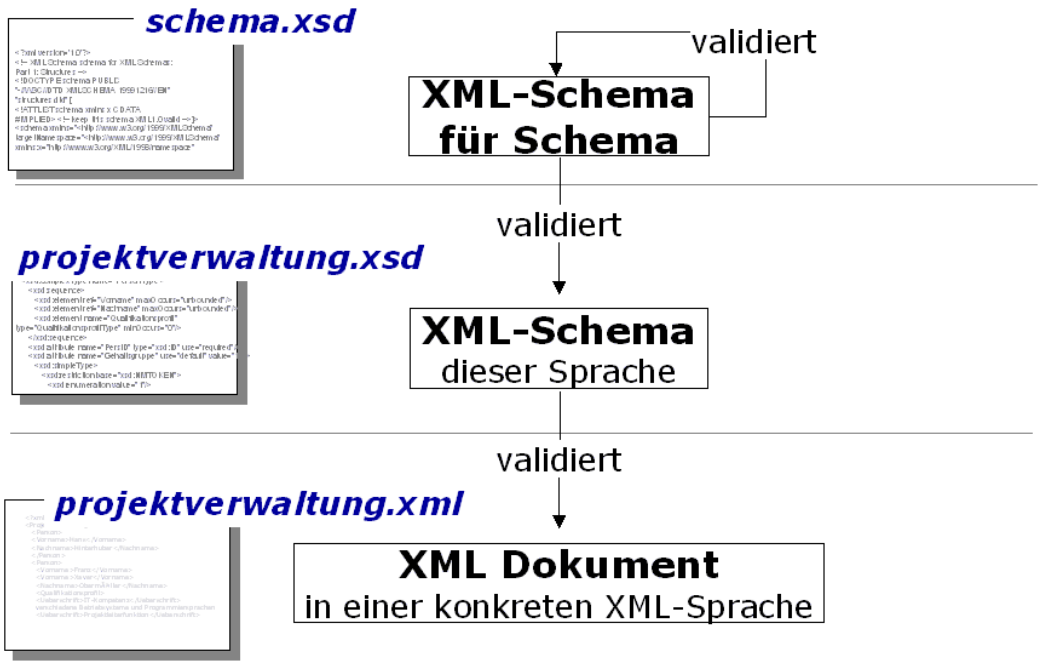
```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:schemaLocation="http://www.jeckle.de/vorlesung/xml/examples/projektverwaltung.
xsd">
(5)     ...
```

Im Beispiel wird zunächst der XML-Schema-Instanzen-Namensraum an das Präfix `xsi` gebunden. Dies ermöglicht die Einbindung von Elementen und Attributen aus der Schemaspezifikation in das eigene Dokument.
Als erste Nutzung eines solchen Elements aus XSD wird das Attribut `schemaLocation` im Wurzelelement mit der URI des Schemas als Wert belegt. Die Deklaration des XSI-Namensraumes ist daher zwingend.
Die angegebene URI kann zur Ermittlung des Schemas für Validierungszwecke durch einen XML-Prozessor genutzt werden.

Aufbauend auf dem Begriff der Wohlgeformtheit definiert XML-Schema den der *Schemagültigkeit* als höhere Qualitätsstufe eines XML-Vokabulars:

Definition 10: Gültigkeit hinsichtlich eines Schemas
Ein XML-Dokument heißt *gültig hinsichtlich eines Schemas* (*schema valid*), wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Daher eröffnet sich die Möglichkeit, das Schema selbst durch ein Schema zu beschreiben. Dieses *Schema für Schema* -- auch *Metaschema* genannte -- XML-Dokument erlaubt die Validierung (im Sinne der *schema validness*) jedes Schemas. Damit erfüllt sich eine der Anforderungen an den Schemamechanismus: die Validierbarkeit der erstellten Schemata selbst, was für DTDs nicht gegeben war. In der praktischen Anwendung zeigt sich dies in der Möglichkeit, erstellte Schemata mit denselben Werkzeugen zu analysieren, verarbeiten und zu prüfen, die auch für Instanzdokumente verwendet werden.
Da das Metaschema selbst wiederum ein XML-Dokument ist, folgt, daß hierfür auch ein Schema angegeben werden kann. Die XML-Standardisierung hat hier -- nicht zuletzt um eine unendliche Reihung zur Validierung notwendiger Schemata zu vermeiden -- den Ansatz gewählt, das Schema für Schema durch sich selbst zu beschreiben.
Die Abbildung stellt die getroffenen Aussagen und Validierungsbeziehungen nochmals graphisch zusammen.



Die Schema-Definition:
Wuzelknoten jedes XSD-Dokuments ist das Element Information Item `schema`. Alle Definitionen eines Schemas sind direkte Kindknoten dieses Elements oder dessen Kindknoten.
Durch die Attribute des `schema`-Elements werden verschiedene Eigenschaften festgelegt, die für alle im Schema definierten Elemente und Attribute gelten.

Zunächst wird durch eine Reihe von Attributen das Verhalten des Schemas in Bezug auf Namensräume festgelegt. Als Besonderheit eines XML-Schemas fällt hier die ständige Berücksichtigung von mindestens zwei Namensräumen ins Auge. Während ein Schema mit Elementen des Schemanamensraumes aufgebaut wird, trifft es zeitgleich Aussagen über einen zweiten Namensraum -- den Namensraum des Vokabulars

für das das Schema erstellt wird. Dieser Namensraum wird *Zielnamensraum* (*target namespace*) genannt. Daher findet sich im Attribut `targetNamespace` die URI des Zielnamensraumes. In diesen Namensraum werden automatisch alle durch das Schema deklarierten Elemente und Attribute übernommen. Als Konsequenz müssen diese in jedem Schema-gültigen XML-Dokument im entsprechenden Namensraum auftreten. Hierbei wird nicht zwischen expliziter Namensraumdeklaration durch ein gebundenes Präfix und impliziter Deklaration durch Überschreiben des Vorgabenamensraumes unterschieden.
Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` kann der durch `targetNamespace` implizierte Namensraumzwang für das XML-Instanzdokument gelockert werden. Wird der Wert der beiden Attribute auf `unqualified` gesetzt, so können die Attribute auch außerhalb des Zielnamensraumes auftreten. Dies entspricht auch dem Vorgabeverhalten.

Definition von Elementen:
Als Obermenge der Ausdrucksmächtigkeit der DTD unterstützt auch XSD die Inhaltsmodelle

- unstrukturierter Inhalt
- beliebig (jedes Element kann als Kindelement angegeben werden)
- leer (keine Kindelemente)
- explizit angegebene Kindelemente
- gemischter Inhalt (gleichzeitiges Auftreten von Elementen und unstrukturiertem Text)

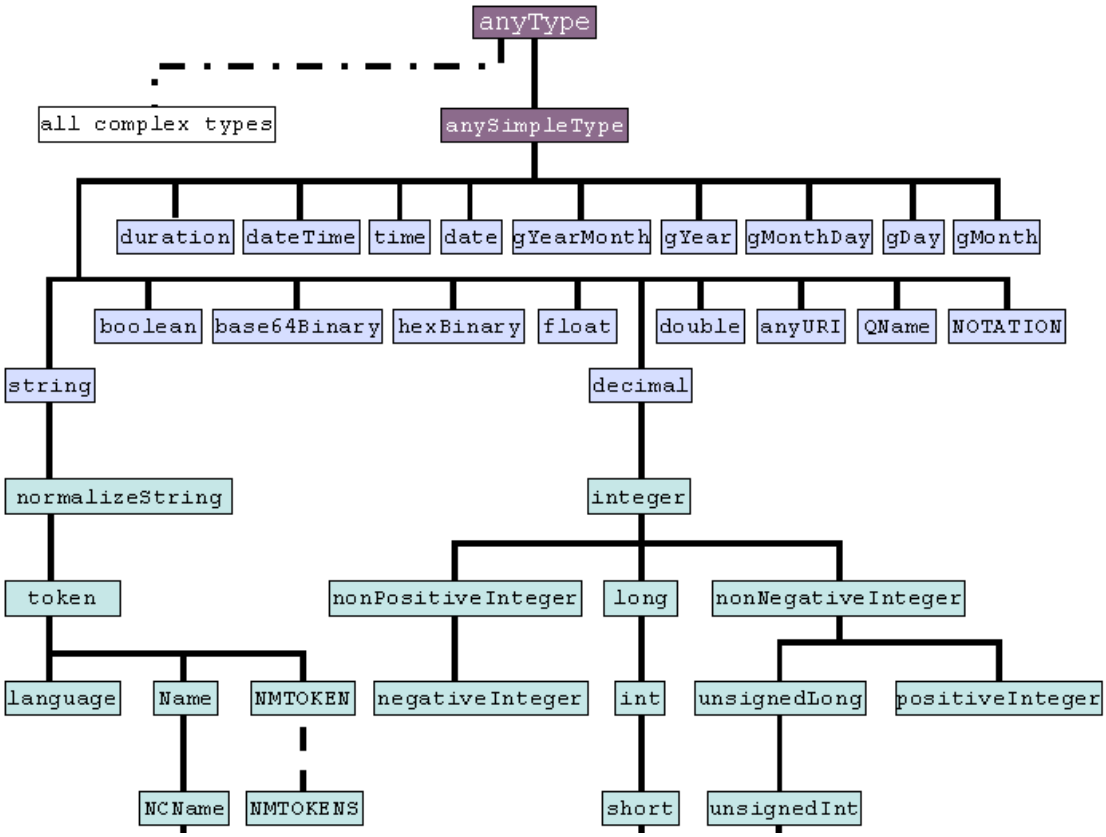
Generell wird jedes Element durch das XSD-Element `element` ausgedrückt.

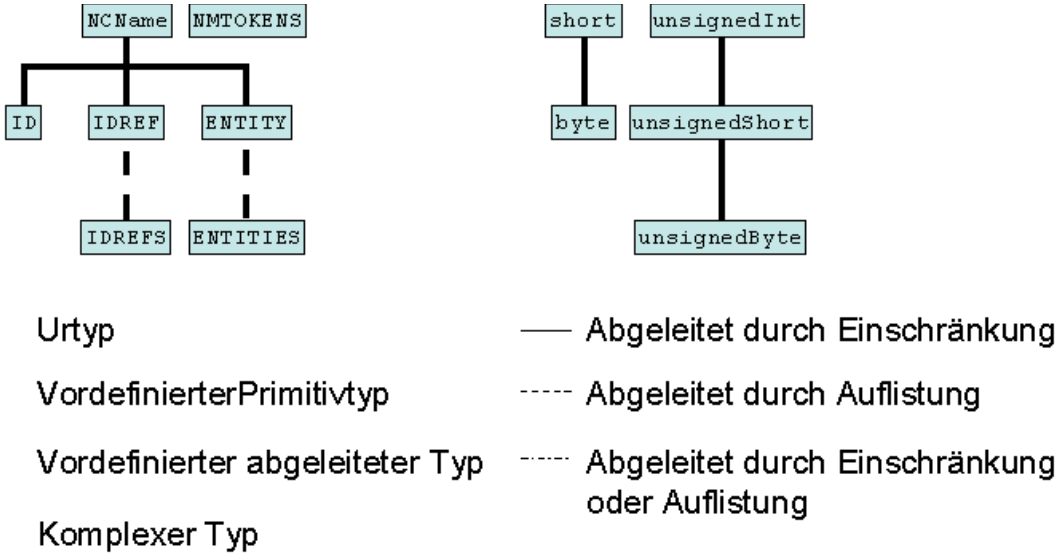
Während die DTD für unstrukturierten Inhalt ausschließliche uninterpretierte Zeichenketten unterstützt, wird die Ausdrucksmächtigkeit durch XML-Schema deutlich gesteigert.
XML-Schema Part 2 definiert insgesamt 44 Primitivtypen. Darunter finden sich die bereits in der DTD angebbaren Element- und Attributtypen, sowie eine Fülle Neuer.

Im Kern zerfallen die XSD-Typen in drei Typklassen:

- Atomare und aggregierte Typen
  Atomare Typen bestehen aus unteilbaren Werten. Der Begriff der *Unteilbarkeit* soll dabei auf die nicht erkennbare Unterstrukturierung (`string` besteht zwar aus einzelnen Zeichen) bzw. falls erkennbar, dem nicht explizit unterstützten Zugriff auf die Komponenten (`date` bietet keine Zugriffsmöglichkeit auf die Komponenten Tag, Monat und Jahr), verweisen.
  Die aggregierten Typen teilen sich in Listen-artige und Vereinigungstypen (*Union*). Listen bestehen dabei aus einer geordneten Menge atomarer Typen.
  Vereinigungstypen erlauben hingegen die Verknüpfung Typ-verschiedener Datentypen zu einem neuen.
- Primitive und abgeleitete Typen
  Primitive Datentypen existieren unabhängig von anderen Datentypen, während abgeleitete Datentypen von der Definition ihres Elterntyps abhängig sind.
- Vorgegebene und anwenderdefinierte Typen
  Die vorgegebenen Typen sind diejenigen, die in XML-Schema Part2 beschrieben sind. Alle weiteren Typen eines Schemas sind durch den Anwender definierte abgeleitete Typen.

Durch Erweiterungs- und Aggregationsmechanismen ergibt sich das in der nachfolgenden Abbildung dargestellte Typsystem.





Die Tabelle stellt die angebotenen Typen mit einigen Beispielen dar:

Tabelle 6: Typen in XSD-Schema Part 2

Typname	Beispiel	Bemerkung
string	Hello
World	Jedes beliebige Unicode Symbol gemäß XML-Syntaxproduktion 2
normalizedString	&#20Hello&#20;World	Jedes beliebige Unicode Symbol außer Zeilenvorschub, Wagenrücklauf und Tabulatoren normalizedString ist eine einschränkende Spezialisierung des Typs string
token	Hello World	Jeder normalizedString, unter Weglassung führender, abschließender und mehrfacher Leerzeichen (#x20), sowie Zeilenvorschüben (#xA) und Tabulatoren (#x9). token ist eine einschränkende Spezialisierung des Type normalizedString
Name	aName, _helloWorld, :notAGoodIdea	XML Name gemäß Syntaxproduktion 5. Name ist eine einschränkende Spezialisierung des Typs token
QName	xsd:element, element	Durch Namensraumpräfix qualifizierter Name gemäß Produktion 6 der XML Namespace Recommendation
NCName	aName, _anotherName, X	Name, der keinen Doppelpunkt enthält (*non colonized name*), gemäß Produktion 4 der XML Namespace Recommendation
decimal	-1.23, 12678967.543233, +100000.00, 210	Wertebereich: $i*10^{-n}$, mit i, n aus integer, n>=0 Ein Prozessor muß mindestens 18 Dezimalstellen unterstützen
long	-9223372036854775808, ... -1, 0, 1, ... 9223372036854775807	Wertebereich: 2^{63} <= long <= 2^{63}-1 long ist eine einschränkende Spezialisierung des Typs integer
int	-2147483648, ... -1, 0, 1, ... 2147483647	Wertebereich: -2^{31} <= int <= 2^{31}-1 int ist eine einschränkende Spezialisierung des Typs long
short	-32768, ... -1, 0, 1, ... 32767	Wertebereich: -2^{15} <= short <= 2^{15}-1 short ist eine einschränkende Spezialisierung des Typs int
byte	-128, ...-1, 0, 1, ... 127	Wertebereich: -2^{7} <= byte <= 2^{7}-1 byte ist eine einschränkende Spezialisierung des Typs short

integer	...-1, 0, 1, ...	Wertebereich: entspricht der mathematischen Menge der ganzen Zahlen (Z) integer ist eine einschränkende Spezialisierung des Typs decimal
positiveInteger	1, 2, ...	Wertebereich: entspricht der mathematischen Menge der natürlichen Zahlen (N) positiveInteger ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
negativeInteger	... -2, -1	Wertebereich: {..., -2, -1}, die unendliche Menge der negativen Zahlen negativeInteger ist eine einschränkende Spezialisierung des Typs nonPositiveInteger
nonNegativeInteger	0, 1, 2, ...	Wertebereich: 0 <= nonNegativeInteger nonNegativeInteger ist eine einschränkende Spezialisierung des Typs integer
nonPositiveInteger	... -2, -1, 0	Wertebereich: {..., -2, -1, 0} die unendliche Menge der negativen Zahlen, und die Null nonPositiveInteger ist eine einschränkende Spezialisierung des Typs integer
unsignedLong	0, 1, ... 18446744073709551615	Wertebereich: 0 <= unsignedLong <= 2^{64}-1 unsignedLong ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
unsignedInt	0, 1, ...4294967295	Wertebereich: 0 <= unsignedInt <= 2^{32}-1 unsignedInt ist eine einschränkende Spezialisierung des Typs unsignedLong
unsignedShort	0, 1, ... 65535	Wertebereich: 0 <= unsignedShort <= 2^{16}-1 unsignedShort ist eine einschränkende Spezialisierung des Typs unsignedInt
unsignedByte	0, 1, ... 255	Wertebereich: 0 <= unsignedByte <= 2^{8}-1 unsignedByte ist eine einschränkende Spezialisierung des Typs unsignedShort
float	-1E4, 1267.43233E12, 12.78e-2, 12, INF	32-Bit-Zahl mit einfacher Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit $m <= 2^{24}$, und $-149 <= e < 104$ sind.
double	-1E4, 1267.43233E12, 12.78e-2, 12, INF	64-Bit-Zahl mit doppelter Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit $m <= 2^{53}$, und $-1075 <= e < 970$ sind.
boolean	true, false, 1, 0	Unterstützung der klassischen zweiwertigen Logik
time	13:20:00-05:00, 13:20:00.000	Uhrzeit, die täglich wiederkehrt, ausgedrückt im Format gemäß ISO 8601
date	2004-06-08	Datumsformat: CCYY-MM-DD, gemäß ISO 8601
gYear	1999, 2001, 2004	Darstellung von Jahren des gregorianischen Kalenders gemäß ISO 8601
gYearMonth	2004-06	Darstellung eines Monats eines bestimmten Jahres des gregorianischen Kalenders gemäß ISO 8601

gDay	----05, ----31	Darstellung eines wiederkehrenden Tages eines Monats gemäß ISO 8601
gMonthDay	--31-12, --01-01	Darstellung eines wiederkehrenden gregorianischen Datums, gebildet aus Tag Monat und Monat im Format `--MM-DD`, gemäß ISO 8601
gMonth	--03, --12	Monatsformat: --MM-- gemäß ISO 8601
dateTime	2004-06-08T12:51:02.000+02:00	Zeitpunkt, ausgedrückt durch Datum und Uhrzeit; beide gemäß ISO 8601 codiert.
duration	P1Y2M3DT10H30M12.3S Zeitraum von einem Jahr, zwei Monaten, drei Tagen, zehn Stunden, 30 Minuten und 12,3 Sekunden	Nach Größe (Signifikanz) geordnete Koordinate im sechs-dimensionalen Raum aus Jahr, Monat, Tag, Stunde, Minute und Sekunde. Formatdefinition laut ISO 8601
base64Binary	SGVsbG8gd29ybGQhCg==	Base64-Darstellung eines beliebigen Binär-interpretierten Inhaltes gemäß IETF RFC 2045
hexBinary	0FB7	Hexadezimale Darstellung beliebiger Binär-interpretierter Inhalte
anyURI	http://www.jeckle.de	Jede gemäß IETF RFC 2396 bzw. IETF RFC 2732 gültige URI
language	en-GB, en, de-de	Sprachcodierung gemäß IETF RFC 1766 und XML Recommendation language identification. Die Identifikationsnamen werden durch ISO 639 sowie ISO 3166 definiert. `language` ist eine einschränkende Spezialisierung des Typs `token`
ID	test, XYZ	XSD-Darstellung des DTD-Typen `ID`. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). `ID` ist eine einschränkende Spezialisierung des Typs `NCName`
IDREF	test, XYZ	XSD-Darstellung des DTD-Typen `IDREF`. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). `IDREF` ist eine einschränkende Spezialisierung des Typs `NCName`
IDREFS	test1 test2 test4, test3 test5	XSD-Darstellung des DTD-Typen `IDREFS`. Zugelassen sind Listen aus white space separierten Ausprägungen der Namespaceproduktion 4 (NCName). `IDREFS` ist eine nichtleere Aufzählung von `IDREF`-Ausprägungen
ENTITY		XSD-Darstellung des DTD-Typen `ENTITY`. Zugelassen sind alle Satzformen, die der Produktion `NCName` der XML-Namensräume entsprechen und als ungeparste Entität definiert sind. `ENTITY` ist eine einschränkende Spezialisierung des Typs `NCName`

ENTITIES		XSD-Darstellung des DTD-Typen ENTITIES. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs ENTITY. ENTITIES ist eine nichtleere Aufzählung von ENTITY-Ausprägungen
NOTATION		XSD-Darstellung des DTD-Typen NOTATION. Zur Verwendung dieses Typs in einem Schema muß eine Ableitung von NOTATION durch den Anwender definiert werden.
NMTOKEN	US, Deutschland	XSD-Darstellung des DTD-Typen NMTOKEN. Ausprägungen dieses Typs müssen konform zur Produktion 7 der XML-Spezifikation sein. NMTOKEN ist eine einschränkende Spezialisierung des Typs token
NMTOKENS	US UK Aus, Ger	XSD-Darstellung des DTD-Typen NMTOKENS. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs NMTOKEN. NMTOKENS ist eine nichtleere Aufzählung von NMTOKEN-Ausprägungen
anyType	1, 2.3, aGVsb, 06b8f45, test&#20; für&#20;anyType&#0A; <sentence>the quick brown <animal>fox</animal>...</sentence>	Allgemeinster Datentyp. Konzeptionell bildet er die Vereinigung aller angebotenen XSD-Typen.

Die einfachste Form zur **Definition eines Elements mit unstrukturiertem typisierten Inhalt** lautet:

```
<xsd:element
                    name="elementName"
                    type="typeName"/>
```

XSD definiert ferner folgende Charakteristika für Elemente, die durch Attribute der Elementdeklaration ausgedrückt werden:

- **abstract**: falls auf `true` gesetzt, darf ein solches Element nicht in einem XML-Dokument auftreten. Es kann ausschließlich zur Strukturierung des Schemaentwurfs eingesetzt werden und als Basis von Spezialisierungen dienen.
  Vorgabewert ist `false`
- **block**: erlaubt die Kontrolle der Verwendung abgeleiteter Typen. Zugelassene Belegungen beliebige Kombinationen aus `extension`, `restriction` und `substitution` oder der Einzelwert `#all`.
  Ist das Attribut auf `restriction` gesetzt, so dürfen keine (einschränkend) abgeleiteten Typen Ausprägungen des Originaltyps in Instanzdokumenten ersetzen. Dasselbe gilt für `extension` oder als Substitutionsgruppen deklarierte Typen (`substitution`-Belegung).
  Der Wert `#all` versammelt alle möglichen Varianten und verbietet generell die Ersetzung eines Elements durch andere.
- **default**: Vorgabebelegung des Inhalts durch eine beliebige Zeichenkette, die konform zum gewählten Typ ist.
  Dieser Wert wird durch den XML-Prozessor an die Applikation gemeldet, wenn kein Wert im Dokument angegeben wird.
- **final**: verhindert die Ableitung von Typen. Die zulässigen Belegungen sind mit denen für `block` identisch, nur daß durch dieses Attribut die Vererbungsmechanismen bereits auf Schemaebene verboten werden, während `block` ihre Nutzung im Instanzdokument einschränkt.
- **fixed**: erlaubt die konstante Wertbelegung.
- **form**: legt fest, ob das Element im Instanzdokument mit Namensraumpräfix erscheint.
  Zulässige Belegungen: `qualified` (Namensraumpräfix muß angegeben werden) und `unqualified`.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Elements durch eine Schema-weit eindeutige Zeichenkette.
- **minOccurs**: Minimalkardinalität, d.h. Mindestzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist ein Element aus nonNegativeInteger.
  Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert `1` belegt.
- **maxOccurs**: Maximalkardinalität, d.h. Höchstzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist entweder ein Element aus nonNegativeInteger oder die Zeichenkette unbounded zur Kennzeichnung beliebig vieler Auftreten.

Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert 1 belegt.

- **name**: Unqualifizierter Name des Elements, konform zur NCName-Produktion der Namensraum-Spezifikation.
- **nillable**: Erlaubt Null-Werte im Instanzdokument, die Semantik ist dabei an die in relationalen Datenbanksystemen verwirklichte angelehnt.
  Die Belegung ist entweder `true` oder `false`, was auch als Vorgabe bei Fehlen dieses Attributs angenommen wird.
- **ref**: Referenz auf eine andere Elementdeklaration zur Übernahme der dort spezifizierten Definitionen.
- **substitutionGroup**: Name einer Gruppe von Elementen, die anstatt des aktuellen Elements im Instanzdokument auftreten dürfen.
- **type**: Ein durch Schema Part 2 vordefinierter Typ, oder jeder beliebige anwenderdefinierte.

Nachfolgend sind einige Elementdeklarationen für unstrukturierten Inhalt versammelt

Beispiel 22:

```
(1)<element name="geburtsdatum" type="xsd:date"/
>
(2)<element name="pi"
(3)     type="xsd:double"
(4)     fixed="3.141592653"
(5)     block="#all"
(6)     final="#all"/>
(7)<element name="vorname"
(8)     type="xsd:token"
(9)     minOccurs="1"
(10)    maxOccurs="unbounded"/>
(11)<element name="artikelNummer"
(12)    type="xsd:NCName"
(13)    form="qualified"/>
```

Die Deklaration `geburtsdatum` definiert ein XML-Element des Typs date zur Darstellung eines Datums. Weitere Festlegungen sind nicht getroffen, daher wird das Element mit `minOccurs` und `maxOccurs` 1 belegt, wodurch es als zwingend anzugebend (*mandatory*) und skalar (d.h. nicht mengenwertig) ausgewiesen wird.
`pi` legt die gleichnamige mathematische Konstante fest. Als Datentyp wurde double, eine Gleitkommazahl mit doppelter Genauigkeit gewählt. Als konstante Belegung wird durch das `fixed` Attribut der entsprechende Zahlenwert festgelegt. Daher muß eine Vorgabebelegung durch das Attribut `default` nicht erfolgen; gemäß Schema-Spezifikation darf sie sogar nicht erfolgen, `fixed` und `default` schließen sich gegenseitig aus. Um eine weitere Spezialisierung des Elements durch Vererbung oder Aggregation zu verhindern wird der Wert von `block` auf `#all` gesetzt, wodurch die Teilnahme an allen Typbildungsmechanismen unterbunden wird.
Die Definition für `vorname` nutzt als Datentyp den token, der automatisch mehrfache, führende und abschließende Leerzeichen sowie sonstige Formatierungssymbole entfernt. Ferner kann dieses Element beliebig häufig auftreten -- `maxOccurs` ist daher auf unbounded gesetzt. Die Fixierung der minimalen Auftrittshäufigkeit auf 1 (`minOccurs`) entspricht der Vorgabebelegung.
Für das Element `artikelNummer` ist als Typ `NCName` ausgewählt, was beliebigen Zeichenketten -- die keinen Doppelpunkt enthalten -- entspricht. Darüberhinaus ist das Attribut `form` mit dem Wert `qualified` versehen. Dies führt dazu, daß das Namensraumkürzel für dieses Element zwingend im Instanzdokument anzugeben ist.

Zur Umsetzung des freien Inhaltsmodells, das beliebige Inhalte aus den definierten Elementen und freien Texten zuläßt, wird ebenfalls auf das Typsystem zurückgegriffen.
Wird das `type` Attribut nicht belegt, so wird gemäß Vorgabe der Typ anyType angenommen. Elemente dieses Typs können beliebige wohlgeformte Inhalte beherbergen.
Die beiden nachfolgenden Angaben sind daher äquivalent.

```
<element
   name="elementName"
   type="xsd:anyType"/>
<element name="elementName"/>
```

XSD prägt den bereits im Kontext der DTD genutzten Typbegriff (dort beschränkt er sich lediglich auf verschiedene Darstellungsformen uninterpretierter Zeichenketten) strenger. Dies zeigt sich deutlich in der Existenz des XSD-Elements `complexType`. Es führt die Möglichkeit einer expliziten, d.h. von der Verwendung losgelösten Typbildung, ein. Syntaktisch kann die complexType-Definition sowohl innerhalb einer Elementdefinition, als auch separat erfolgen.
Den einfachsten Anwendungsfall bildet die eingebettete leere complexType-Definition zur Darstellung des **leeren Inhaltsmodells**.
Die Syntax hierfür lautet (der XSD-Namensraum sei an das Präfix `xsd` gebunden):

```
<xsd:element
```

```
    name="elementName">
    <xsd:complexType/>
</xsd:element>
```

Ein XML-Schema-validierender Parser verhält sich in diesem Falle identisch zu einem (DTD-)validierenden Parser. Daher werden für die obige Festlegung ausschließlich die beiden Darstellungsformen zur Angabe eines leeren Elements (`<elementName/>` bzw. `<elementName></elementName>`) akzeptiert.

Die Befüllung des `complexType`-Elements leitet direkt zum wichtigsten Inhaltsmodell über, dem explizit angegebener Kindelemente.
Zur Festlegung der Elementreihenfolge definiert XML-Schema das Element `sequence`, welches die Angabe der Kindelemente in genau der im Schema angegebenen Reihenfolge erzwingt.
Das Auswahlinhaltsmodell (auch: Selektionsmodell) --- welches alternativ das Auftreten beliebiger Elemente definiert --- wird entsprechend durch das XSD-Element `choice` ausgedrückt.
Eine besondere Variante des Selektionsmodells stellt die `all`-Gruppe dar. Es erlaubt die Angabe der Kindelemente in beliebiger Reihenfolge.
Die drei Ausgangsvarianten können im Rahmen einer Elementdefinition beliebig geschachtelt und auf diesem Wege kombiniert werden.
Am Beispiel der Elementdefinitionen der Projektverwaltung:

Beispiel 23: Einige Elementdefinitionen

```
(1)<?xml version = "1.0" encoding = "UTF-8"?>
(2)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name = "ProjektVerwaltung">
(4)             <xsd:complexType>
(5)                     <xsd:sequence>
(6)                             <xsd:element ref = "Person" maxOccurs = "unbounded"/>
(7)                             <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(8)                     </xsd:sequence>
(9)             </xsd:complexType>
(10)    </xsd:element>
(11)    <xsd:element name = "Person">
(12)            <xsd:complexType>
(13)                    <xsd:sequence>
(14)                            <xsd:element name = "Vorname" type = "xsd:token" maxOccurs
= "unbounded"/>
(15)                            <xsd:element name = "Nachname" type = "xsd:token"/>
(16)                            <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(17)                    </xsd:sequence>
(18)            </xsd:complexType>
(19)    </xsd:element>
(20)    <xsd:element name = "Projekt">
(21)            <xsd:complexType/>
(22)    </xsd:element>
(23)    <xsd:element name = "Qualifikationsprofil">
(24)            <xsd:complexType mixed = "true">
(25)                    <xsd:sequence>
(26)                            <xsd:element name = "Qualifikation" type = "xsd:string"
minOccurs = "0" maxOccurs = "unbounded"/>
(27)                            <xsd:element name = "Leistungsstufe" type = "xsd:string"
minOccurs = "0" maxOccurs = "unbounded"/>
(28)                    </xsd:sequence>
(29)            </xsd:complexType>
(30)    </xsd:element>
(31)</xsd:schema>
```

Download des Beispiels

Das Schema enthält alle Elementdefinitionen für die Projektverwaltung. Innerhalb jedes `element`-Elements sind die entsprechenden Kindelemente in `sequence`-Strukturen eingebettet. Die Elemente müssen daher in der Reihenfolge ihres Auftretens im Schema auch im Instanzdokument wiedergegeben werden.
Von besonderem Interesse ist die Definition des *Qualifikationsprofil*s. Es handelt sich dabei um ein **mixed content model**, ausgedrückt durch das Boole'sche Attribut `mixed` (in Spezifikation nachschlagen).
Darüberhinaus enthält das Beispiel neben *lokalen Elementdeklarationen*, die sich vollständig im Elternelement finden (wie `Vorname`, `Nachname` und `Qualifikation`), auch *globale Elementdeklarationen*, die zunächst deklariert und in einem zweiten Schritt durch Referenzierung als Kindelemente verwendet werden (wie `Person` und `Projekt` innerhalb `Projektverwaltung`, oder `Qualifikationsprofil` innerhalb des Elements `Person`). Hierdurch können vollständige Elemente an verschiedenen Stellen im Schema referenziert und so verwendet werden. Die Definition ist der lokalen ebenbürtig und wird im Instanzdokument identisch behandelt. Zusammenfassend läßt sich festhalten: Mit dem Referenzierungsmechanismus für Elemente kann eine einfache Form der Wiederverwendung umgesetzt werden.
Den Zeichenketten-artigen Elementtypen wurde durchgehend der XSD-Typ `string` zugewiesen.

Durch die Referenzierungsmöglichkeit existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elemente. Jedoch werden Elemente hierbei zwingend in ihrer vollständigen Definition, d.h. Name, Typ und Inhaltsmodell, eingebunden.
XML-Schema bietet die Möglichkeit, strukturierte Typen, die ausschließlich durch ihr Inhaltsmodell definiert werden, festzulegen. In der Konsequenz verändert sich der durch die DTD formulierte Typbegriff hin zu einer eher an den Programmiersprachen orientierten Sichtweise, da die Benennung des Typs von der Namensgebung der typisierten Instanz separiert wird.
Syntaktisch erfolgt die Typbildung durch die Benennung des `complexType`-Elements durch ein Attribut `name`. Um die mehrfache Verwendung eines solchen Typen zu ermöglichen, muß seine Definition zwingend auf einer Baumstufe erfolgen, die für alle nutzenden Elemente erreichbar ist. Üblicherweise werden daher diese Definitionen auf der ersten Stufe, direkt unterhalb des Wurzelknotens, plaziert.
Zur Unterscheidung dieser *benannten komplexen Typen* werden die bisher genutzten -- namenlosen Typen -- als *anonyme komplexe Typen* bezeichnet.
Das nachfolgende Beispiel zeigt die Definition eines benannten komplexen Typen am Beispiel des Elements `Person`:

Beispiel 24: Nutzung benannter komplexer Typen

```
(1)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(2)   <xsd:complexType name="PersonType">
(3)        <xsd:sequence>
(4)        <xsd:element name = "Vorname" type = "xsd:string"
(5)                        maxOccurs = "unbounded"/>
(6)        <xsd:element name = "Nachname" type = "xsd:string"/>
(7)        <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(8)   </xsd:sequence>
(9)   </xsd:complexType>
(10)
(11)   <xsd:element name = "ProjektVerwaltung">
(12)      <xsd:complexType>
(13)         <xsd:sequence>
(14)            <xsd:element name="Person" type="PersonType" maxOccurs = "unbounded"/>
(15)            <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(16)         </xsd:sequence>
(17)      </xsd:complexType>
(18)   </xsd:element>
(19)
(20)   <xsd:element name = "Projekt">
(21)      <xsd:complexType/>
(22)   </xsd:element>
(23)
(24)   <xsd:element name = "Qualifikationsprofil">
(25)      <xsd:complexType mixed = "true">
(26)         <xsd:sequence>
(27)            <xsd:element name = "Qualifikation" type = "xsd:string"
(28)                           minOccurs = "0" maxOccurs = "unbounded"/>
(29)            <xsd:element name = "Leistungsstufe" type = "xsd:string"
(30)                           minOccurs = "0" maxOccurs = "unbounded"/>
(31)         </xsd:sequence>
(32)      </xsd:complexType>
(33)   </xsd:element>
(34)</xsd:schema>
```

Download des Beispiels

Das Schema zeigt die Definition des komplexen Typen `PersonType`. Dieser Typ wird zur Festlegung des Inhaltsmodells des Elements `Person` verwendet.

Definition eigener Datentypen durch Vererbung:
Zur Unterstützung von Wiederverwendung und Erhöhung der Strukturierung des Entwurfs definiert XSD ein Vererbungskonstrukt zur Bildung neuer komplexer Typen auf der Basis bereits bestehender.
Zwei verschiedene Ableitungssemantiken werden angeboten:

- Ableitung durch Einschränkung (*derivation by restriction*)
  Der erbende Subtyp gibt eine engere Definition des Supertypen
- Ableitung durch Erweiterung (*derivation by extension*)
  Der erbende Subtyp erweitert die Definition des Supertypen

Das nachfolgende Beispiel zeigt die Anwendung der einschränkenden Ableitung.
Hierbei erbt der benannte komplexe Typ `childType` von `parentType`. Innerhalb des -- aus syntaktischen Gründen notwendigen -- Elements `complexContent` findet sich die Definition der Vererbung im Element `restriction`, das `base`-Attribut verweist auf den benannten Elterntypen.
Der Inhalt des `restriction`-Elements gleicht der Inhaltsmodelldefinition des komplexen Typen: Auch hier werden Elemente und ihre Auftrittsstruktur (im betrachteten Beispiel `sequence`) angegeben. Die

Elementdefinition des Elements `elementA` in `childType` schränkt die gleichnamige Elementdefinition innerhalb des Elterntypen ein. Nachvollziehbar wird diese Einschränkungsbeziehung zwischen `short` und `int` bei Betrachtung der Datentyphierarchie und der Typdefinition der verwendeten Primitivtypen. So bildet `short` per definitionem eine eingeschränkte Untermenge von `int` an. (Die entsprechende XSD-Definition findet sich im *Schema für Schema*).
Die beiden Elementdefinitionen `usage1` und `usage2` zeigen die Verwendung der anwenderdefinierten Typen.

Beispiel 25: Einschränkende Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)<xsd:complexType name="parentType">
(4)     <xsd:sequence>
(5)             <xsd:element name="elementA" type="xsd:int"/>
(6)     </xsd:sequence>
(7)</xsd:complexType>
(8)
(9)<xsd:complexType name="childType">
(10)<xsd:complexContent>
(11)    <xsd:restriction base="parentType">
(12)            <xsd:sequence>
(13)                    <xsd:element name="elementA" type="xsd:short"/>
(14)            </xsd:sequence>
(15)    </xsd:restriction>
(16)</xsd:complexContent>
(17)</xsd:complexType>
(18)
(19)<xsd:element name="usage1" type="parentType"/>
(20)<xsd:element name="usage2" type="childType"/>
(21)
(22)</xsd:schema>
```

Download des Beispiels

Durch das strukturierte Inhaltsmodell ergeben sich über die reine Typisierung hinausgehende Möglichkeiten zur Einschränkung der Inhalte. Die nachfolgende Tabelle stellt einige Varianten zusammen.

Tabelle 7: Beispiele für zulässige Restriktionen

Basistyp	Restriktion	Bemerkung
	`default`	Zusätzliche Belegung eines Elements mit einem Vorgabewert
	`fixed`	Beschränkung eines zunächst frei wählbaren Elements auf konstanten Inhalt
	`type`	Definition eines Typen für ein zunächst untypisiertes Element. (Auch hierbei handelt es sich um eine einschränkende Redefinition, da allen Elementen ohne Typdefinition standardmäßig der Typ `anyType` zugeordnet wird.)
`minOccurs`=n_1, `maxOccurs`=m_1	`minOccurs`=n_2, `maxOccurs`=m_2	Restriktion der Auftrittshäufigkeit auf eine geringere Anzahl. Daher gilt: $n_1 <= n_2$ und $m_1 >= m_2$

Die direkte Umkehrung der einschränkenden Spezialisierung bildet die *erweiternde Spezialisierung*. Sie greift nicht verändernd auf die Elemente des Supertyps zu, sondern definiert zusätzliche neue. Untenstehendes XSD-Schema zeigt dies am Beispiel des Supertyps `parentElement`, der durch das abgeleitete Kindelement `childElement` erweitert wird. Hierzu definiert `childElement` ein zusätzliches `elementB`.

Beispiel 26: Erweiternde Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:complexType name="parentElement">
(4)             <xsd:sequence>
(5)                     <xsd:element name="elementA"/>
(6)             </xsd:sequence>
(7)     </xsd:complexType>
(8)
(9)     <xsd:complexType name="childElement">
(10)            <xsd:complexContent>
(11)                    <xsd:extension base="parentElement">
(12)                            <xsd:sequence>
(13)                                    <xsd:element name="elementB"/>
(14)                            </xsd:sequence>
(15)                    </xsd:extension>
(16)            </xsd:complexContent>
(17)    </xsd:complexType>
(18)</xsd:schema>
```

Download des Beispiels

Zusätzlich sieht XML Schema die Möglichkeit vor, komplexe Typen von simplen abzuleiten. Dies mag auf den ersten Blick ungewöhnlich erscheinen, eröffnet es doch scheinbar einen Weg, unstrukturierte Typen in strukturierte zu überführen.
Bei näherer Betrachtung offenbart sich jedoch, daß hier lediglich der Ableitungsbegriff überladen wurde, um einen einfachen Weg zur Verknüpfung der beiden Inhaltsmodelle strukturierter „XML-artiger“ Inhalt -- wie er durch `complexTypes` repräsentiert wird -- auf der einen, und unstrukturierter Inhalt -- wie er durch die einfachen Datentypen repräsentiert wird -- auf der anderen Seite, zu erhalten.

Beispiel 27: Ableitung eines komplexen Typen von einem Simplen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xs:schema
(3)     xmlns:xs="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xs:element name="Vorname">
(7)             <xs:complexType>
(8)                     <xs:simpleContent>
(9)                             <xs:extension base="xs:string">
(10)                                    <xs:attribute
(11)                                            name="rufname"
(12)                                            type="xs:boolean"/>
(13)                            </xs:extension>
(14)                    </xs:simpleContent>
(15)            </xs:complexType>
(16)    </xs:element>
(17)</xs:schema>
```

Download des Beispiels

Durch die im Beispiel dargestellte Syntax wird es ermöglicht unstrukturiert-getypten Elementen Attribute zuzuordnen, obwohl diese eigentlich Bestandteil der Definition komplex-getyper Elemente sind.

So wird im Beispiel dem Element `Vorname` sowohl der simple Typ `string`, als auch durch den Ableitungsmechanismus das Attribut `rufname` -- im Rahmen eines `complexType`, zugeordnet.
Die Typisierung des Elements erfolgt hierbei nicht durch das `type`-Attribut innerhalb der Elementdeklaration, sondern innerhalb der `simpleContent`-Festlegung.

Neben der anwenderdefinierten Bildung komplexer Typen steht es dem XSD-Modellierer auch offen, eigene (primitive) Datentypen festzulegen oder eigene Typen von bestehenden abzuleiten.
Hierfür definiert XML-Schema Part1 das Element simpleType. Für einfache Typen ist jedoch nur die einschränkende Vererbung (*restriction*) zugelassen. Dies liegt in der praktischen Beherrschbarkeit des Typsystems begründet. Durch die strikte Restriktionssemantik ergibt sich die Möglichkeit kontravarianter Substitution, wie sie bei objektorientierten Typsystemen und Vererbungsstrukturen anzutreffen ist. Dies bedeutet, daß an jeder Stelle, an der eine Ausprägung eines Supertyps erwartet wird, auch -- unter Erhalt der Typrestriktion -- eine Ausprägung eines Subtypen auftreten darf. Beispielhaft: Wird an einer Stelle des Instanzdokumentes durch das Schema das Auftreten einer Ausprägung von `integer` verlangt, so kann der Anwender auch Ausprägungen der Subtypen `int`, `short` oder `byte` angeben ohne die Gültigkeit des XML-Dokuments zu beeinträchtigen.

Vereinigungstypen werden aus einer nichtleeren Menge von Ausgangstypen gebildet.

Das Beispiel zeigt die Definition eines Typen `termin`, der den vorgegebenen Primitivtypen `date` und eine Liste `NamenDerWochentage` (deren Definition nicht dargestellt ist) vereinigt. Insbesondere zeigt der Ausschnitt die Möglichkeit der Vereinigungsbildung auch über aggregierte Typen.

```
(1)<xs:simpleType name="termin">
(2)     <xs:union memberTypes="xs:date NamenDerWochentage"/>
(3)</xs:simpleType>
```

Das XSD-Beispiel zeigt, als Fragment der XML-Schemaspezifikation, die Definition des vorgegebenen Typs `short`, einer einschränkenden Spezialisierung des Typs `int`.
Am Beispiel gut nachvollziehbar sind die beiden Schritte zur Bildung eines eigenen Typen:

1. Auswahl eines Ausgangstypen (später Elementtyp (bei aggregierten Typen) oder Basistyp (bei abgeleiteten Typen))
2. Typdefinition durch Anwendung der entsprechenden Typkonstruktion und evtl. Einschränkung verschiedener Charakteristika

Im Beispiel wird der kleinste und größte gültige Wert (`minInclusive` bzw. `maxInclusive`) des neuen Typen `short` gegenüber dem Basistypen beschränkt.

Beispiel 28: Einschränkende Spezialisierung eines simplen Typen

```
(1)<xsd:simpleType name="short" id="short">
(2)     <xsd:restriction base="xsd:int">
(3)             <xsd:minInclusive value="-32768"
(4)                     id="short.minInclusive"/
>
(5)             <xsd:maxInclusive value="32767"
(6)                     id="short.maxInclusive"/>
(7)     </xsd:restriction>
(8)</xsd:simpleType>
```

Die Bildung aggregierter Typen folgt demselben Muster. Jedoch tritt an die Stelle der Ableitung die Spezifikation des Aggregationstyps (im Beispiel *Liste*) und Angabe des Inhaltstyps (im Beispiel `string`).

Beispiel 29: Bildung eines Aggregationstypen

```
(1)<xsd:simpleType name="WarenkorbElemente">
(2)     <xsd:list itemType="xsd:string"/>
(3)</xsd:simpleType>
```

Nachfolgend sind die verschiedenen Beschränkungsmöglichkeiten zusammengefaßt:

- `length`
  Längenbeschränkung bei atomaren Typen bzw. Beschränkung der Elementanzahl bei aggregierten Typen.
  Längenbeschränkung eines simplen Typs:

```
(1)<xs:simpleType name="Postleitzahl">
(2)     <xs:restriction base="xs:string">
(3)             <xs:length value="5"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Beschränkung der Elementanzahl einer Liste:

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:length value="5"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

- `minLength`
  Minimale Länge (bei atomaren Typen bzw. minimale Elementanzahl bei aggregierten Typen.
  Beispiel (der aggregierte Typ `VornamenRestrictedList` muß mindestens einen Eintrag enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
```

```
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- maxLength
  Maximale Länge bei atomaren Typen bzw. maximale Elementzahl bei aggregierten Typen.
  Beispiel (der aggregierte Type VornamenRestrictedList muß mindestens einen, jedoch höchstens drei, Einträge enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)             <xs:maxLength value="3"/>
(8)     </xs:restriction>
(9)</xs:simpleType>
```

Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf, darf jedoch aus höchstens 80 Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)             <xs:maxLength value="80"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

- pattern
  Erlaubt die Inhaltsdefinition durch Muster (reguläre Ausdrücke).
  XML-Strukturen sind nicht durch reguläre Ausdrücke darstellbar, hierfür können lediglich die vorgestellten Strukturierungsprimitive eingesetzt werden; daher sind Muster auf atomare Typen beschränkt.
  Die XML-Schemaspezifikation definiert einzelne Symbole und Symbolsequenzen, sowie einige abkürzende Schreibweisen zum Aufbau beliebiger Ausdrücke. Alle anderen Zeichen können direkt in die Musterspezifikation übernommen werden.
  Die aus der Verarbeitung regulärer Ausdrücke mit anderen Programmiersprachen oder Werkzeugen bekannten Quantifier stehen in der bekannten Semantik zur Verfügung.
  Sei S eine Zeichenkette aus einer beliebigen Anzahl von Zeichen (d.h. sie kann auch leer sein!), dann gilt:

Tabelle 8: Übersicht der Quantifier

Quantifiziertes Mustersymbol	Bedeutung
S	Alle Zeichenketten, die S genau entsprechen.
S?	Alle Zeichenketten, die S genau entsprechen oder die leere Zeichenkette.
S*	Alle Reihungen von S; insbesondere entspricht S* auch der leere Zeichenkette.
S+	Alle Reihungen von S, die S mindestens einmal enthalten. Entspricht der Formulierung: S S*
S{n,m}	Alle Zeichenketten, die aus mindestens *n*, jedoch höchstens *m* Auftreten von S bestehen. Durch *n=0* lassen sich somit optionale, nach oben begrenzte, Auftrittsanzahlen realisieren, sowie durch *n=m=0* die leere Zeichenkette ausdrücken
S{n}	Alle Zeichenketten, die aus genau *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n,n}

S{n,}	Alle Zeichenketten, die aus mindestens *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n} S*

Zur Darstellung nicht-druckbarer Zeichen oder von Metasymbolen werden folgende Fluchtsymbole angeboten:

Tabelle 9: Übersicht der Escape-Symbole

Mustersymbol (*escape character*)	**ausgedrücktes Zeichen**
\n	Zeilenumbruch (#xA)
\r	Zeilenvorschub (#xD)
\t	Tabulator (#x9)
\\	\
\\|	|
\.	.
\-	-
\^	^
\?	?
*	*
\+	+
\{	{
\}	}
\(	(
\)	)
\[	[
\]	]

Ferner sind noch eine Reihe häufig benötigter Zeichenfamilien definiert und in den Kategorien Buchstaben (Letter), Marken (Marks), Zahlen (Numbers), Interpunktion (Punctuation), Trennsymbole (Separators) sowie sonstige (Others) zusammengefaßt. Die Zeichenfamilien innerhalb dieser Kategorien entsprechen den in Unicode v3.1 definierten general categories in Namen und Aufbau.

Tabelle 10: Buchstaben

symbolische Darstellung	**Bedeutung**
L	(All Letters) Alle Buchstaben
Lu	(Uppercase) Alle Großbuchstaben
Ll	(Lowercase) Alle Kleinbuchstaben
Lt	(Titlecase) Sprachabhängige (potentielle) Großschreibung des ersten Buchstabens eines Wortes. (vgl. UNICODE technical report #21 sowie Derived General Category Lt).
Lm	(Modifiers) Zusammenfassung der verschiedensten Ton- und Betonungszeichen
Lo	(Others) Zusammenfassung von Zeichen, die in keine der sonstigen L-Zeichenfamilien fallen

Tabelle 11: Markierungssymbole

symbolische Darstellung	**Bedeutung**
M	(All Marks) Alle Markierungssymbole
Mn	(Non-Spacing) Markierungssymbole, ausschließlich Leerzeichen
Mc	(Space combining) Markierungssymbole mit Leerzeichen kombiniert
Me	(Enclosing) Markierungssymbole, die andere Zeichen umschließen

Tabelle 12: Zahlen

symbolische Darstellung	Bedeutung
N	(All Numbers) Beliebige Ziffer; daher nicht notwendigerweise arabisch. Unicode stellt eingekreiste (#x2460-#x2473), geklammerte (#x2474-#x2487) sowie Ordinalzahlen (mit abschließendem Punkt) (#x2488-#x249b) zur Verfügung.
Nd	(Decimal Digit) eine arabische Ziffer
Nl	(Letter) auf Buchstaben beruhende Zifferndarstellungen, wie römische Ziffernsymbole (#x20dd-#x217f)
No	(Other) Alle sonstigen Ziffernsymbole

Tabelle 13: Interpunktionszeichen

symbolische Darstellung	Bedeutung
P	(Punctuation) Alle Interpunktionssymbole
Pc	(Connector) Verbindende Interpunktionssymbole (z.B. Unterstrich (#x5f)
Pd	(Dash) Verschiedene Verbindungsstriche
Ps	(Open) Öffnende Bereichssymbole wie die verschiedenen Klammertypen
Pe	(Close) Schließende Bereichssymbole wie die verschiedenen Klammertypen
Pi	(Initial Quote) Öffnende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Pf	(Final Quote) Schließende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Po	(Other) Alle anderen Interpunktionssymbole

Tabelle 14: Separatoren

symbolische Darstellung	Bedeutung
Z	Alle Separatoren
Zs	(Space) Trennende Leerzeichen
Zl	(Line) Zeilentrenner (#x2028)
Zp	(Paragraph) Absatztrenner (#x2029)

Tabelle 15: Symbole

symbolische Darstellung	Bedeutung
S	Alle Symbole
Sm	(Math) Verschiedene mathematische Symbole (Operatoren, Pfeile, etc.)
Sc	(Currency) Währungssymbole (Dollarzeichen: #x24, Eurozeichen: #x20ac)
Sk	(Modifier) Ton- und Betonungssymbole (ähnlich zu Lm)
So	(Other) Alle anderen Symbole

Tabelle 16: Sonstige Zeichen

symbolische Darstellung	Bedeutung
C	Alle sonstigen
Cc	(Control) Nicht-druckbare Kontrollzeichen
Cf	(Format) Formatierungszeichen (z.B. syrisches Abkürzungssymbol)
Co	(Private Use) ungefähr 137500 Zeichen zur freien anwenderdefinierten Belegung
Cn	(Not Assigned) Zeichen, denen innerhalb Unicode explizit keine Belegung zugewiesen wurde

Reguläre Ausdrücke können innerhalb des pattern-Elements direkt angegeben werden. Die Zeichenkettenfamilien werden durch ihre symbolische Darstellung, eingeleitet durch \p und durch geschweifte Klammern umschlossen, dargestellt.
Zusätzlich ist durch \P das Komplement zu jeder der aufgeführten Zeichenkettenfamilien definiert.
Ergänzend kann die Definition der zulässigen Zeichengruppen vollständig wahlfrei erfolgen. Hierzu wird das Negationssymbol ^ angeboten, welches eine Aufzählung von Zeichen von der Verwendung ausschließt.
Darüberhinaus können auf der Basis simpler Mengendifferenzoperationen eigene Zeichenklassen komfortabel definiert werden.
Für die am häufigsten benötigten Zeichenklassen sind durch XML-Schema bereits vorgegebene abkürzende Schreibweisen definiert. Hierzu werden bereits die bisher vorgestellten Syntaxmechanismen angewendet.

Tabelle 17: Zeichensequenzen

abkürzende Schreibweise	entsprechende Langform
.	[^\n\r]
\s	[#x20\t\n\r]
\S	[^\s]
\i	Die initialen Zeichen eines gültigen XML-Namens; entspricht dem ersten Teil der Syntaxproduktion 5
\I	[^\i]
\c	Diejenigen Zeichen, die der XML-Syntaxproduktion 4 entsprechen
\C	[^\c]
\d	\p{Nd}
\D	[^\d]
\w	[#x0000-#x10FFFF]-[\p{P}\p{S}\p{C}] Alle Zeichen, außer der Klassen für Interpunktions- und Separatorenzeichen, sowie der Klasse der sonstigen Zeichen.
\W	[^\w]

Beispiel (eine vereinfachte, d.h. unter Nicht-Berücksichtigung von Behörden- und Diplomatennummern) deutsche Autonummer im bekannten Format: Einführende Bezeichnung des Landkreises durch mindestens einen, jedoch höchstens drei Großbuchstaben, gefolgt von einem trennenden Bindestrich, an den sich ein oder zwei weitere Großbuchstaben anschließen. Auf ein Leerzeichen folgen eine, jedoch höchstens vier Ziffern:

```
(1)<xs:simpleType name="gerAutoNummer">
(2)     <xs:restriction base="xs:string">
(3)             <xs:pattern value="\p{Lu}{1,3}-\p{Lu}{1,2} \p
{Nd}{1,4}"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

Weitere Informationen: Anhang F -- Reguläre Ausdrücke -- der XML-Spezifikation

- enumeration
  Festlegung von zulässigen Werten eines Typs durch vollständige Aufzählung.
  Beispiel (die Ampelfarben: rot, gelb, grün):

```
(1)<xs:simpleType name="ampelfarben">
(2)     <xs:restriction base="xs:string">
(3)             <xs:enumeration value="rot"/>
```

```
(4)             <xs:enumeration value="gelb"/>
(5)             <xs:enumeration value="grün"/>
(6)     </xs:restriction>
(7)</xs:simpleType>
```

- whitespace
  Behandlung von white spaces innerhalb eines Elements des definierten Typs. Erlaubte Belegungen und ihre Bedeutung:
  preserve: Keine Veränderung des Inhaltes durch Normalisierung; evtl. enthaltene white spaces bleiben erhalten
  replace: Alle Vorkommen von Tabulatoren, Zeilenvorschub und Wagenrücklauf werden durch Leerzeichen (#x20) ersetzt
  collapse: Nach der Substitution gemäß *replace* werden mehrfach auftretende Leerzeichen zu einem einzigen kompaktifiziert, sowie führende und abschließende Leerzeichen entfernt.
  (in Spezifikation nachschlagen)
- maxInclusive
  Höchster zulässiger numerischer Wert. Im mathematischen Sinne sind daher alle Werte kleiner oder gleich dem im value-Attribut angegebenen zugelassen.
  Beispiel (Zahlen <= 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxInclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Anmerkung: In vielen Fällen kann eine maxInclusive-Festlegung ohne Informationsverlust in eine äquivalente maxExclusive-Definition überführt werden.
- maxExclusive
  Wert „unterhalb“ (im mathematischen Sinne „kleiner“) dem alle numerischen Belegungen zugelassen sind.
  Beispiel (Zahlen < 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxExclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Als Belegungen des Typs uHu sind alle Zahlen kleiner als 100 zugelassen. Durch die Verwendung des XSD-Typen decimal als Basistyp kann auch keine verlustfreie Überführung in eine maxInclusive-Festlegung überführt werden, da hierfür die größte zugelassene Zahl fixiert werden müßte, was jedoch die Typdefinition von decimal explizit offen läßt.
- minInclusive
  Kleinster zugelassener Wert.
  Beispiel (Zahlen >= 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minInclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- minExclusive
  Kleinster Wert, der nicht mehr zugelassen ist. Im mathematischen Sinne sind alle Zahlen, die größer sind, gültige Belegungen.
  Beispiel (Zahlen > 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minExclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- totalDigits
  Gesamtstellen einer Zahl, gebildet aus der Summe der Vorkomma- und der Nachkommastellen.
  Beispiel (Zahlen mit höchstens sieben Stellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:totalDigits value="7"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- fractionDigits
  Anzahl der Nachkommastellen eines Dezimalbruches.
  Beispiel (Zahl mit höchstens neuen Stellen, davon zwei Nachkommastellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="decimal">
(3)             <xs:totalDigits value="9"/>
(4)             <xs:fractionDigits value="2"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

Definition von Attributen:
Die Attributdeklaration erfolgt durch das XSD-Element attribute. Die Mächtigkeit entspricht auch hier, wie bereits für die Elemente verwirklicht, einer Obermenge der DTD. So können neben optionalen, zwingenden und konstanten Attributen auch Aufzählungsattribute und Mengen realisiert werden. Hierbei wurde auf die Orthogonalität zum durch simpleType geschaffenen Typmechanismus geachtet.
Die Charakteristika (ausgedrückt in Attributen des XSD-Elements attribute) einer Attributdeklaration umfassen:

- **name**: Ein Doppelpunkt-freier Namen (NCName) gemäß Namensraumproduktion 7.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Attributs durch eine Schema-weit eindeutige Zeichenkette.
- **default**: Belegung mit Vorgabewert.
- **fixed**: Konstante Belegung.
- **type**: Typ des Attributes, definiert durch einen simpleType.
- **form**: Legt fest, ob der Attributname im XML-Instanzdokument durch ein Namensraumpräfix eingeleitet wird (Belegung: qualified, andernfalls unqualified).
- **ref**: Verweis auf eine globale Attributdefinition.
- **use**: Verwendung des Attributes, Wert entspricht optional, required oder prohibited. Vorgabegemäß wird optional angenommen und das Attribut damit nicht zwingend im XML-Dokument erwartet. Den Gegensatz hierzu bildet required, wodurch das Attribut als zwingend anzugeben definiert wird. prohibited verbietet die Nutzung des Attributes im XML-Dokument.

Anmerkung: Einen Anwendungsfall der Belegung prohibited für use bilden Attribute, die innerhalb des Schemas bereits definiert sind, jedoch noch nicht zur allgemeinen Nutzung freigegeben wurden.

Beispiel 30: Einige Attributdefinitionen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
(3)     <xsd:attribute name="myAtt1"/>
(4)
(5)     <xsd:attribute name="myAtt2" type="xsd:decimal"/>
(6)
(7)     <xsd:attribute name="myAtt3">
(8)             <xsd:simpleType>
(9)                     <xsd:restriction base="xsd:int">
(10)                            <xsd:minInclusive value="10"/>
(11)                            <xsd:maxInclusive value="20"/>
(12)                    </xsd:restriction>
(13)            </xsd:simpleType>
(14)    </xsd:attribute>
(15)
(16)    <xsd:simpleType name="myType1">
(17)            <xsd:restriction base="xsd:string">
(18)                    <xsd:maxLength value="5"/>
(19)            </xsd:restriction>
(20)    </xsd:simpleType>
(21)    <xsd:attribute name="myAtt4" type="myType1"/>
(22)
(23)
(24)
(25)    <xsd:element name="foo">
(26)            <xsd:complexType>
(27)                    <xsd:attribute ref="myAtt1" use="optional"/>
(28)                    <xsd:attribute ref="myAtt2" use="required"/>
(29)                    <xsd:attribute ref="myAtt3" use="prohibited"/>
(30)                    <xsd:attribute ref="myAtt4"/>
(31)                    <xsd:attribute name="myAtt5" type="xsd:date" id="myDate"/>
(32)                    <xsd:attribute name="myAtt6">
(33)                            <xsd:simpleType>
(34)                                    <xsd:restriction base="xsd:float">
(35)                                            <xsd:totalDigits value="5"/>
(36)                                    </xsd:restriction>
```

```
(37)                            </xsd:simpleType>
(38)                    </xsd:attribute>
(39)            </xsd:complexType>
(40)    </xsd:element>
(41)
(42)</xsd:schema>
```

Download des Beispiels

Das Beispiel zeigt einige Varianten der Attributdeklaration. So definieren `myAtt1` mit `myAtt4` globale Attribute, die innerhalb verschiedener Elemente verwendet werden können. Hierdurch wird die bereits für Elemente verwirklichte Mimik der einmaligen Deklaration und anschließenden beliebigen Verwendung auch auf Attribute ausgedehnt. Die Nutzung der so deklarierten Attribute geschieht durch das `ref`-Attribut innerhalb des `Attribute`-Elements des beherbergenden Elements.
`myAtt1` definiert ein typenloses Attribut, dem vorgabegemäß der allgemeinste Typ `anyType` zugeordnet wird. Die Angabe dieses Attributes ist optional (`use="optional"`), was der Vorgabe entspricht.
Der XSD-Standardtyp `decimal` findet zur Definition des Attributs `myAtt2` Verwendung. Die zwingend anzugebenden (`use="required"`) Inhalte dieses Attributs werden durch einen XML-Schema-Parser auf Typkonformität geprüft.
`myAtt3` veranschaulicht die Bildung eines anonymen (inneren) atomaren Typen zur Definition eines Attributs. Der durch Restriktion gebildete neue Datentyp steht ausschließlich innerhalb des Attributs `myAtt3` zur Verfügung. Die Syntax der Datentypspezialisierung entspricht der im vorhergehenden Abschnitt diskutierten. Zudem ist die Verwendung des Attributes innerhalb eines XML-Dokumentes untersagt; ausgedrückt durch die Belegung `use="prohibited"`
Analog der Typisierung eines Elementinhaltes durch einen anwenderdefinierten Typen gestaltet sich das Vorgehen für Attribute. Veranschaulicht wird dies durch die Definition von `myAtt4`. Sie greift auf den eigen-definierten Typen `myType1` zurück.
Dem Attribut `myAtt5` ist zusätzlich zur Benennung, die innerhalb des verwendenden Elementes eindeutig sein sollte, ein Dokument-weiter Schlüssel (`id`) zugeordnet.
Innerhalb des Elements `foo` werden die fünf zuvor definierten Attribute verwendet. Trotz der Reihenfolge der Definitionen im `complexType`-Element verfügen die Attribute im XML-Instanzdokument -- auch bei der Verwendung von XML-Schema -- über keinerlei Reihenfolge (vgl. XML-Spezifikation).
Zusätzlich enthält die Elementdefintion für `foo` mit `myAtt6` ein „lokales“ Attribut. Diese Definitionsvariante entspricht am ehesten der der Document Type Definition, da sie eine Wiederverwendung außerhalb des definierenden Elements ausschließt.

Beispiel 31: Vollständiges XML-Schema der Projektverwaltung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name="Nachname" type="xsd:string"/>
(4)     <xsd:complexType name="PersonType">
(5)             <xsd:sequence>
(6)                     <xsd:element ref="Vorname" maxOccurs="unbounded"/>
(7)                     <xsd:element ref="Nachname" maxOccurs="unbounded"/>
(8)                     <xsd:element name="Qualifikationsprofil"
type="QualifikationsprofilType" minOccurs="0"/>
(9)             </xsd:sequence>
(10)            <xsd:attribute name="PersID" type="xsd:ID" use="required"/>
(11)            <xsd:attribute name="Gehaltsgruppe" default="1a">
(12)                    <xsd:simpleType>
(13)                            <xsd:restriction base="xsd:NMTOKEN">
(14)                                    <xsd:enumeration value="1"/>
(15)                                    <xsd:enumeration value="1a"/>
(16)                                    <xsd:enumeration value="2"/>
(17)                            </xsd:restriction>
(18)                    </xsd:simpleType>
(19)            </xsd:attribute>
(20)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:IDREFS" use="required"/>
(21)    </xsd:complexType>
(22)    <xsd:complexType name="ProjektType">
(23)            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
(24)            <xsd:attribute name="date" type="xsd:date"/>
(25)            <xsd:attribute name="budget" default="10000.00">
(26)                    <xsd:simpleType>
(27)                            <xsd:restriction base="xsd:double">
(28)                                    <xsd:fractionDigits value="2"/>
(29)                            </xsd:restriction>
(30)                    </xsd:simpleType>
(31)            </xsd:attribute>
(32)            <xsd:attribute name="Projektleiter" type="xsd:IDREF" use="required"/>
(33)            <xsd:attribute name="Mitarbeiter" type="xsd:IDREFS" use="required"/>
(34)    </xsd:complexType>
(35)    <xsd:element name="ProjektVerwaltung">
```

```
(36)            <xsd:complexType>
(37)                    <xsd:sequence>
(38)                            <xsd:element name="Person" type="PersonType"
maxOccurs="unbounded"/>
(39)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(40)                    </xsd:sequence>
(41)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(42)            </xsd:complexType>
(43)    </xsd:element>
(44)    <xsd:complexType name="QualifikationsprofilType" mixed="true">
(45)            <xsd:choice minOccurs="0" maxOccurs="unbounded">
(46)                    <xsd:element ref="Qualifikation"/>
(47)                    <xsd:element ref="Leistungsstufe"/>
(48)                    <xsd:any namespace="http://www.w3.org/1999/xhtml"/>
(49)            </xsd:choice>
(50)    </xsd:complexType>
(51)    <xsd:element name="Qualifikation" type="xsd:string"/>
(52)    <xsd:element name="Leistungsstufe" type="xsd:string"/>
(53)    <xsd:element name="Vorname" type="xsd:string"/>
(54)</xsd:schema>
```

Download des Beispiels

Abschließend eine gültige (sowohl *valid* als auch *schema valid*) Dokumentinstanz der Projektverwaltungsstruktur.

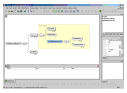
Beispiel 32: Gültiges Projektverwaltungsdokument

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/
projektverwaltung.xsd">
(5)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(6)             <Vorname>Hans</Vorname>
(7)             <Nachname>Hinterhuber</Nachname>
(8)     </Person>
(9)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(10)            <Vorname>Franz</Vorname>
(11)            <Vorname>Xaver</Vorname>
(12)            <Nachname>Obermüller</Nachname>
(13)            <Qualifikationsprofil>
(14)     IT-Kompetenz verschiedene Betriebssysteme und <Leistungsstufe>professionelle</
Leistungsstufe>
(15)                    <Qualifikation>Programmierung</Qualifikation> verschiedener
Programmiersprachen
(16)     <Qualifikation>Entwickler</Qualifikation> von 1988-1990
(17)     <Qualifikation>Projektleiterfunktion</Qualifikation> von 1990-93 im X42-Projekt
in Abteilung AB&amp;amp;C
(18)   </Qualifikationsprofil>
(19)    </Person>
(20)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(21)            <Vorname>Fritz</Vorname>
(22)            <Nachname>Meier</Nachname>
(23)    </Person>
(24)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(25)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(26)</ProjektVerwaltung>
```

Download des Beispiels

Werkzeuge:
Zwar existiert -- wie für alle XML-Dokumente -- die Möglichkeit, Dokument Typ Definitionen und XML-Schemata „per Hand“ mit einem Texteditor zu erstellen, jedoch ist dieses Vorgehen, insbesondere für umfangreiche XML-Vokabulare, zeitaufwendig und fehlerträchtig. Zusätzlich läßt die rein textuelle Formulierung die entstehenden Schemadokumente schnell unübersichtlich werden.
Inzwischen existieren einige gute DTD- und Schemaeditoren, die zumeist neben visueller Syntaxhervorhebung auch die kontextsensitive Editierung erlauben und so eine wesentliche Erleichterung der Schemaerzeugung bilden. Gleichzeitig bieten die meisten verfügbaren Werkzeuge dieser Klasse auch Möglichkeiten zur Validierung des erzeugten Schemas an.
Ergänzend wird vielfach auch eine graphische Repräsentation der DTD- oder XSD-Struktur angeboten.
Die Abbildungen zeigen Ansichten der Werkzeuge XML Authority bzw. XML Spy





Web-Referenzen 7: Weiterführende Links und Werkzeuge

- XML Schema Part 0: Primer
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- XML Schema @ Cover-Pages
- Parsing the Atom -- Diskussion über die Vor- und Nachteile inhärent komplexer atomarer Typen
- Schema-Informationen @ jeckle.de
- XML-Authority (DTD- und XSD-Editor)
- XML Spy (DTD- und XSD-Editor)

2.4 XPath

Zur Extraktion beliebiger Teile eines wohl-geformten XML-Dokuments verabschiedete das W3C 1999 die Sprache *XPath*. Sie bildet eine pfadorientierte *Lokatorsprache*, die das Auffinden von Dokumentteilen (einzelnen Elementen, Attributen, etc.) durch Pfadausdrücke, die sich an der Struktur des XML-Dokuments orientieren, gestattet.
Die Grenze zwischen Lokatorsprache und „echter" Anfragesprache wie SQL sind fließend. Zwei Unterscheidungsmerkmale sollen jedoch hervorgehoben werden: XPath wird im üblichen Anwendungsfall nicht interaktiv oder in eine Programmiersprache als Wirtssprache eingebettet verwendet, sondern wurde (zunächst) nur für die Nutzung in Kombination mit der Transformationssprache *XSLT* und den erweiterten Verweisen der Sprache *XPointer* konzipiert. Zum zweiten fehlt XPath die üblicherweise mit dem reinen Anfrageteil verwobene Manipulationssprache zur Änderung bereits bestehender Daten; XPath ist allein für den lesenden Zugriff auf XML-Dokumente ausgelegt.
Hinweis: XPath unterscheidet XML-üblich zwischen Groß- und Kleinschreibung. Daher sind Element- und Attributnamen unbedingt in der im Dokument gewählten Schreibweise anzugeben.

Lokalisierungspfade:
Lokalisierungspfade dienen der abstrakten Beschreibung einer Menge von Informationsknoten innerhalb eines Dokuments.
Die einfachste Form eines Lokalisierungspfades beschreibt der *Wurzellokalisierungpfad* (*root location path*), ausgedrückt durch „/". Er liefert für jedes XML-Dokument den Wurzelknoten. Dieser ist nicht identisch mit dem Wurzelelement eines XML-Dokuments! Der (unbenannte) Wurzelknoten entspricht dem Document Information Item des Information Sets, während das erste benannte Element des Dokuments durch ein Element Information Item dargestellt wird.

Die Navigation zu den einzelnen Elementknoten, oder Knotenmengen, wird durch einen Pfadausdruck realisiert. Die explizite Variante erlaubt die Angabe aller zu traversierenden Knoten bis hin zu den zu extrahierenden. Hierzu werden die Knoten, von der Wurzel absteigend durch „/"-Symbole separiert, notiert. Wegen der Korrespondenz der voneinander abgetrennten Knotennamen und den Baumstufen, werden diese auch als *Lokalisierungsschritte* bezeichnet. Als weitere sprachliche Analoge spiegelt der XPath-Ausdruck, von links nach rechts gelesen, auch die Schritte -- ausgehend vom Wurzelelement des Dokuments -- zur Lokalisierung der gesuchten Knotenmenge wieder.
Das Beispiel zeigt eine solche Definition am Beispiel der Projektverwaltung.
Anmerkung: Das Resultat ist in XML-Notation dargestellt, obwohl genaugenommen eine Knotenmenge des Information Sets als Resultat zurückgeliefert wird. Die gewählte XML-Darstellung ist hierbei nur eine der möglichen Varianten zur Ergebnispräsentation.

Beispiel 33: XPath-Ausdruck zur Lokalisierung aller Vornamen

XPath-Ausdruck: `/ProjektVerwaltung/Person/Vorname`
Ergebnis:
```
<Vorname>Hans</Vorname>,
<Vorname>Franz</Vorname>,
<Vorname>Xaver</Vorname>,
<Vorname>Fritz</Vorname>
```

Die Einzelknoten werden entsprechend ihrer Auftrittsreihenfolge im Quelldokument (sog. *document order*) zurückgegeben.

Die expliziten Pfadausdrücke lassen sich in beliebiger Länge fortsetzen, jedoch zeigen sie fundamentale Schwächen in Puncto Flexibilität. Wie im Beispiel der XHTML-Verwendung innerhalb eines eigenen XML-Dokuments gesehen, kann Information desselben Typs (d.h. umschlossen durch denselben Tag) verschiedene Elternknoten besitzen. So im Beispiel, dort ist die `Qualifikation` auf derselben Baumstufe sowohl unterhalb des Elternelements `em` als auch `u` anzutreffen.
Als Lösung erlaubt XPath die Nutzung von Platzhaltern statt der expliziten Elementnamen innerhalb eines Lokalisierungsschrittes. In der Folge entstehen freie Lokalisierungsschritte, die alle Kindknoten einer im direkt vorhergehenden Lokalisierungsschritt selektierten Knotenmenge adressieren.

Der nachfolgende XPath-Ausdruck zeigt dies am Beispiel des `Qualifikationsprofils`.

Beispiel 34: Platzhalter in Lokalisierungsschritten

XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil/*/Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>
<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Der Pfadausdruck liefert die beiden Kindelemente `Qualifikation` -- unabhängig von der Benennung des Elternknotens -- die direkt unterhalb des Knotens `Qualifikationsprofil` angeordnet sind.
Allerdings enthält die Ausgabe nicht alle Knoten des Typs `Qualifikation`. Der gegebene Pfadausdruck gestattet lediglich das Überspringen einer Hierarchieebene. Daher wird der hierarchisch tieferstehende `Qualifikations`-Knoten mit Inhalt `Entwickler` nicht lokalisiert. Die (zunächst naheliegende) Lösung den Pfadausdruck zu `/ProjektVerwaltung/Person/Qualifikationsprofil/*/*/Qualifikation` zu erweitern liefert nicht das gewünschte Resultat aller `Qualifikations`-Knoten, sondern ausschließlich den zuvor nicht lokalisierbaren, da der modifizierte Ausdruck nun zwingend zwei freie Lokalisierungsschritte vorsieht.
Zur Variierung der Tiefe der freien Schritte sieht XPath die Schreibweise „//“ vor. Sie erlaubt die Lokalisierung der Kindknoten auf einer beliebigen Hierarchiestufe.

Definition 11: Lokalisierungsschritt

Ein Lokalisierungsschritt setzt sich aus dem Namen der Achse gefolgt von zwei Doppelpunkten und einem *Knotentest*, optional ergänzt um ein auszuwertendes Prädikat, zusammen.
Wird keine Achse spezifiziert, so gilt vorgabegemäß die Achse `child`.
Ein *Knotentest* ist syntaktisch ein QName, der genau dann erfüllt ist, wenn der Knotenname mit dem Namen des Knotentests übereinstimmt.
Das *Prädikat* filtert die Ergebnismenge hinsichtlich verschiedener Charakteristika wie Existenz von Kindknoten oder Attributen, Position in der Ergebnismenge, etc.

Das Beispiel zeigt die korrekte XPath-Formulierung zur Lokation aller `Qualifikations`-Knoten:

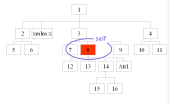
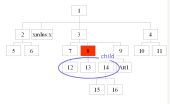
Beispiel 35: Hierarchieunabhängige Knoten-Lokalisierung

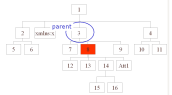
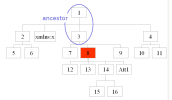
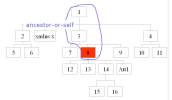
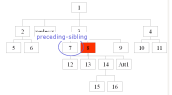
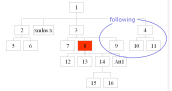
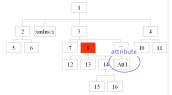
XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil//Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>

<Qualifikation>Entwickler</Qualifikation>

<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Durch die abkürzende Schreibweise „//“ entsteht ein Muster zur Selektion aller nachfolgenden Knoten. In Verallgemeinerung dieses Konzepts bietet XPath sog. *Achsen* an, um relativ zum aktuellen Knoten beliebige Teilbäume zu lokalisieren.
Die Abbildung zeigt die verschiedenen durch Achsen zugänglichen Knotenmengen relativ zum rot hervorgehobenen aktuellen Knoten.

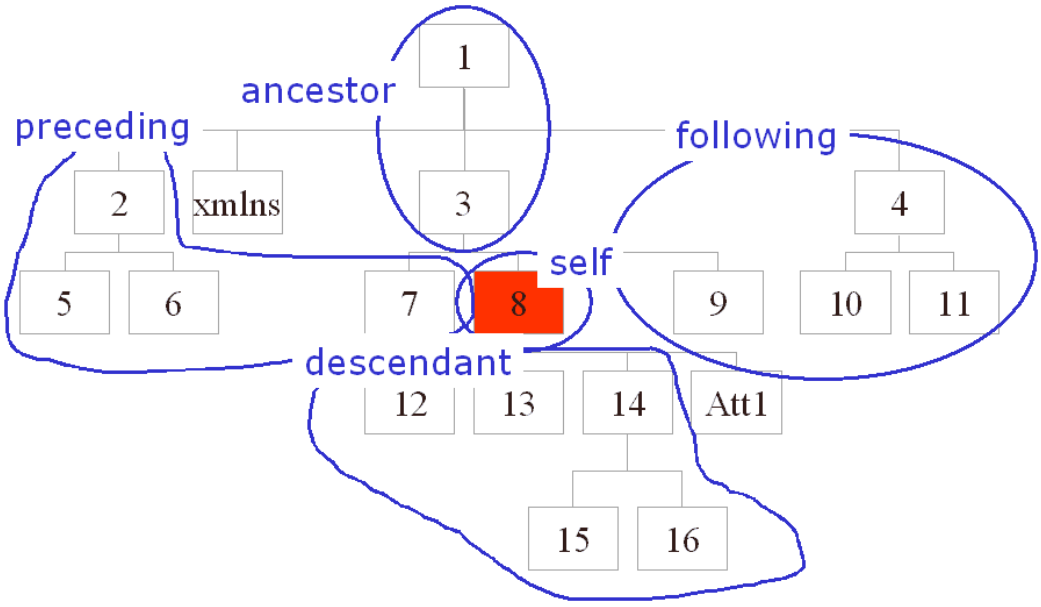
Download der XML-Datei mit dem Beispiel der Graphik

Tabelle 18: XPath-Achsen und ihre Bedeutung

Achse	Semantik	Im Beispiel selektierte Knoten	Graphik
`self`	Lokalisiert den aktuellen Knoten Als abkürzende Schreibweise kann der Punkt „.“ verwendet werden.	**XPath-Ausdruck:** `/node1/node3/node8/self::node8` **Ergebnisknotenmenge:** {8}	
`child`	Lokalisiert die (direkten) Kindknoten des aktuellen Knotens	**XPath-Ausdruck:** `/node1/node3/node8/child::*` **Ergebnisknotenmenge:** {12, 13, 14}	
`descendant`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens, außer Attribut- und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/descendant::*` **Ergebnisknotenmenge:** {12, 13, 14, 15, 16}	
`descendant-or-self`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens (außer Attribut- und Namensraumknoten), sowie den Knoten selbst	**XPath-Ausdruck:** `/node1/node3/node8/descendant-or-self::*` **Ergebnisknotenmenge:** {8, 12, 13, 14, 15, 16}	

`parent`	Lokalisiert den Elternknoten des aktuellen Knotes, falls existent	**XPath-Ausdruck:** `/node1/node3/node8/parent::*` **Ergebnisknotenmenge:** {3}	
`ancestor`	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes. Die `ancestor`-Achse enthält daher immer den Wurzelknoten, außer der aktuelle Knoten ist es selbst; in diesem Falle liefert die Achse die leere Menge	**XPath-Ausdruck:** `/node1/node3/node8/ancestor::*` **Ergebnisknotenmenge:** {1, 3}	
`ancestor-or-self`	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes, sowie den aktuellen Knoten. Diese Achse enthält immer den Wurzelknoten des Dokuments.	**XPath-Ausdruck:** `/node1/node3/node8/ancestor-or-self::*` **Ergebnisknotenmenge:** {1, 3, 8}	
`preceding`	Lokalisiert alle dem aktuellen Knoten vorausgehenden Knoten, ohne seine Vorfahren sowie Attribut- und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/preceding::*` **Ergebnisknotenmenge:** {2, 5, 6, 7}	
`preceding-sibling`	Lokalisiert die im Dokument vor dem aktuellen Knoten auftretenden Geschwisterknoten	**XPath-Ausdruck:** `/node1/node3/node8/preceding-sibling::*` **Ergebnisknotenmenge:** {7}	
`following`	Lokalisiert alle dem aktuellen Knoten nachfolgenden Knoten ohne dessen Kind-, Attribut und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/following::*` **Ergebnisknotenmenge:** {9, 4, 10, 11}	
`following-sibling`	Lokalisiert alle „Geschwister“ des aktuellen Knotens, d. h. Knoten auf derselben Hierarchieebene.	**XPath-Ausdruck:** `/node1/node3/node8/following-sibling::*` **Ergebnisknotenmenge:** {9}	
`attribute`	Lokalisiert Attribut(e) eines Knotens	**XPath-Ausdruck:** `/node1/node3/node8/attribute::*` **Ergebnisknotenmenge:** {Att1}	
`namespace`	Lokalisiert Namensraum-Attribut eines Knotens	**XPath-Ausdruck:** `/node1/node3/node8/namespace::*` **Ergebnisknotenmenge:** `{xmlns:xml="http://www.w3.org/XML/1998/namespace", xmlns:x="namespace:www.jeckle.de/vorlesung/xml"}`	

Anmerkung:
Die Achsen `ancestor`, `descendant`, `following`, `preceding` und `self` partitionieren ein Dokument (unter Auslassung der Attribut- und Namensraumknoten): sie überschneiden sich nicht und enthalten alle Elementknoten des Dokuments.



Filterung durch Prädikate:
Ein -- durch eckige Klammern abgegrenztes -- Prädikat kann innerhalb jedes Lokalisierungsschrittes eines XPath-Ausdrucks angegeben werden. Fehlt es, wird die bisher ermittelte Knotenmenge nicht modifiziert. Das Prädikat kann selbst ein gültiger XPath-Ausdruck sein.
Das prinzipielle Vorgehen kann folgendermaßen beschrieben werden:
Beginnend von links nach rechts für jeden Lokalisierungsschritt: (1) Ermittlung der zur Anfrage passenden Knotenmenge
(2) Reduzierung der Ergebnismenge um diejenigen Knoten, für die das Prädikat `false` liefert.
Befinden sich rechts vom aktuell bearbeiteten Lokalisierungsschritt weitere Ausdrücke, so wird die Resultatmenge als Eingabe eines weiteren Schritts (1) übergeben.

Beispiel 36: Selektion unter Anwendung eines Prädikats
XPath-Ausdruck: `//Person[Qualifikationsprofil]/Nachname`
Ergebnis:
```
<Nachname>Obermüller</Nachname>
```

Der Ausdruck selektiert an beliebiger Stelle des Dokuments („//“) alle Knoten des Typs `Person`. Die Knotenmenge wird um diejenigen `Personen` vermindert, zu denen kein `Qualifikationsprofil` angelegt ist. D.h. Es werden nur diejenigen Knoten selektiert, die über einen Kindknoten des Typs `Qualifikationsprofil` verfügen. Von dieser Knotenmenge (des Typs `Person`!) werden anschließend im zweiten Lokalisierungsschritt die Kindknoten des Typs `Nachname` selektiert.
Mithin liefert der XPath-Ausdruck alle `Nachnamen` von `Personen`, zu denen ein `Qualifikationsprofil` abgelegt ist.
Anmerkung: Das Beispiel nutzt im Prädikat die abkürzende Schreibweise zur Angabe der Vorgabeachse `child`. Die ausführliche Schreibweise -- mit unveränderter Semantik -- des XPath-Ausdruckes lautet daher: `//Person[child::Qualifikationsprofil]/Nachname`

Durch die zusätzliche Definition eines Prädikats für den zweiten Lokalisierungsschritt kann eine weitere Filterung der Ergebnismenge realisiert werden. Zusätzlich können innerhalb eines Prädikats neben XPath-Ausdrücken auch einige vordefinierte Funktionen verwendet werden.
Das Beispiel zeigt die Selektion der `Vornamen` als Kind eines `Personen`-Knotens (Test der Elternschaft durch erstes Prädikat), wenn dieser mit „O“ beginnt (Test durch *starts-with*-Funktion innerhalb des zweiten Prädikats). Die Struktur der Eingabedatei zwingt zusätzlich zur Anwendung der `following`-Achse, da Knoten des Typs `Nachname` in der Dokumentreihenfolge nach Knoten des Types `Vornamen` auftreten.

Beispiel 37: Schrittweise Berechnung einer Selektion unter Verwendung mehrerer Prädikate
XPath-Ausdruck: `//Person[parent::ProjektVerwaltung]/Vorname[starts-with(following::Nachname,'O')]`

Ausgewerteter XPath:`//Person`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname`
Ergebnis:

```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname[following::Nachname]`
Ergebnis:

```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:

```
//Person[parent::ProjektVerwaltung]/Vorname[starts-with(following::Nachname,'O')]
```

Ergebnis:

```
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
```

Die durch die XPath-Spezifikation vordefinierten Funktionen lauten in der Übersicht:

Tabelle 19: XPath-Funktionen für Knotenmengen (node-sets)

Funktionsprototyp	Funktionalität
number last()	Liefert die Größe der aktuellen Knotenmenge; damit den Index des letzten Elements
number position()	Liefert die Position des aktuellen Knotens innerhalb der Knotenmenge. Die erste Knoten trägt die Positionsnummer 1.
number count(*node-set*)	Liefert Elementzahl der übergebenen Knotenmenge
node-set id(*object*)	Liefert denjenigen Knoten, dessen ID-typisiertes Attribut den Argumentwert aufweist. *Anmerkung*: Zur Nutzung dieser Funktion muß zwingend eine Dokument-Grammatik (DTD oder Schema) zum Eingangsdokument vorliegen.
string local-name (*node-set*?)	Liefert den local name (oder die Menge der Namen) der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt.
string namespace-uri (*node-set*?)	Liefert die Namensraum-URI der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt. *Anmerkung*: Handelt es sich nicht um einen Element- oder Attributknoten, so ist die retournierte Zeichenkette leer.
string name (*node-set*?)	Liefert die QName(n) (=qualifizierte(r) Name(n) aus Namensraumkürzel und local name) der übergebenen Knotenmenge, oder des aktuellen Knotens bei leerer Knotenmenge. *Anmerkung*: Nur für Element- und Attributknoten liefert name andere Resultate als local-name.

Tabelle 20: XPath-Funktionen für Zeichenketten

Funktionsprototyp	Funktionalität
string string (*object*)?	Liefert Zeichenkettenrepräsentation einer Knotenmenge. Dabei wird der Zeichenkettenwert des ersten Knotens in der Dokumentreihenfolge zurückgegeben, andernfalls die leere Zeichenkette.
string concat (*string*, *string*, *string**)	Verkettet mindestens zwei Zeichenketten.
boolean starts-with (*string1*, *string2*)	Liefert true falls string1 das zweite Argument string2 als Präfix enthält; andernfalls false
boolean contains (*string1*, *string2*)	Liefert true falls string1 die Zeichenkette aus string2 enthält; andernfalls false.
string substring-before (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich vor dem ersten Auftreten der Zeichenkette *string2* befindet.

string substring-after (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich nach dem ersten Auftreten der Zeichenkette *string2* befindet.
string substring (*string*, *number1*, *number2*?)	Liefert eine Zeichenkette der Länge *number2* aus *string*, beginnend mit der Position *number1*. Fehlt das dritte Argument, so wird der Teilstring bis zum Ende der Zeichenkette *string* zurückgegeben. *Anmerkung*: Das erste Zeichen trägt die Indexnummer 1, nicht 0 wie in Java und C üblich.
number string-length(*string*?)	Liefert die Länge der übergebenen Zeichenkette. Wird kein Argument übergeben, so wird die Länge des zuvor in eine Zeichenkette konvertierten aktuellen Knotens zurückgegeben.
string normalize-space (*string*?)	Liefert die übergebene Zeichenkette unter Entfernung führender, schließender und mehrfacher Leerzeichen zurück. Ferner werden noch evtl. in der Argumentzeichenkette enthaltenen Entitätsreferenzen aufgelöst. *Anmerkung*: Der Normalisierungsvorgang entspricht damit der Attributwertenormalisierung nach Abschnitt 3.3.3 der XML-Spezifikation.
string translate (*string1*, *string2*, *string3*)	Liefert die Zeichenkette *string1* wobei jedes Zeichen aus *string2* durch das Zeichen an derselben Position aus *string3* ersetzt wurde.

Tabelle 21: Boole'sche XPath-Funktionen

Funktionsprototyp	Funktionalität
boolean boolean (*object*)	Liefert die Boole'sche Repräsentation des übergebenen Arguments. Hierbei gilt: •Eine Zahl wird genau dann nach true konvertiert, wenn sie weder Null (unbeachtlich ihres Vorzeichens) noch eine nicht darstellbare Zahl (NaN) ist. •Eine Knotenmenge ergibt true, wenn sie nicht leer ist. •Eine Zeichenkette ergibt true, wenn sie nicht leer (d.h. Länge größer Null) ist. •Die Konvertierung anderer Typen ist typabhängig, und nicht durch den Standard festgelegt
boolean not (*boolean*)	Negiert das übergebene Argument
boolean true()	Liefert statisch den Wert true
boolean false()	Liefert statisch den Wert false
boolean lang (*string*)	Liefert true wenn der aktuelle Knoten ein xml:lang-Attribut gemäß der als Argument übergebenen Sprache besitzt

Tabelle 22: Zahlenorientierte XPath-Funktionen

Funktionsprototyp	Funktionalität
number number (*object*?)	Konvertiert ein Objekt in eine Zahl gemäß folgender Regeln: •Eine Zeichenkette wird in eine Fließkommazahl gemäß IEEE 754 konvertiert, wenn sie aus einem optionalen Leerzeichen, gefolgt durch ein optionales Minuszeichen, gefolgt von einem optionalen Leerzeichen und einer Ziffernfolge besteht. •Der Boole'sche Wert true wird zu 1, der Wert false zu 0 konvertiert. •Eine Knotenmenge wird zunächst in eine Zeichenkette übersetzt, und dann gemäß der oben definierten Regeln umgesetzt. •Die Konvertierung anderer Typen erfolgt typabhängig, und ist nicht durch den Standard geregelt. Wird kein Argument übergeben, so wird stattdessen der aktuelle Knoten als einziges Element einer Knotenmenge interpretiert.
number sum (*node-set*)	Liefert die Summe aller Elemente der übergebenen Knotenmenge, die zuvor in eine Zahl konvertiert werden.
number floor (*number*)	Liefert die größte ganze Zahl, die nicht größer als das Argument ist. *Anmerkung*: Entspricht dem Abschneiden beliebiger Nachkommastellen
number ceiling (*number1*)	Liefert die kleinste ganze Zahl, die nicht kleiner als das Argument ist. *Anmerkung*: Entspricht floor(number1+0.999...)
number round (*number*)	Liefert das Argument auf die nächste ganze Zahl gerundet. Gibt es zwei solche -- wie bei Nachkommastelle gleich 0.5 immer der Fall -- so wird die größere zurückgeliefert.

Für mathematische Berechnungen auf zahlenartigen Knoten stehen folgende Operatoren zur Verfügung.

Tabelle 23: Mathematische Operatoren

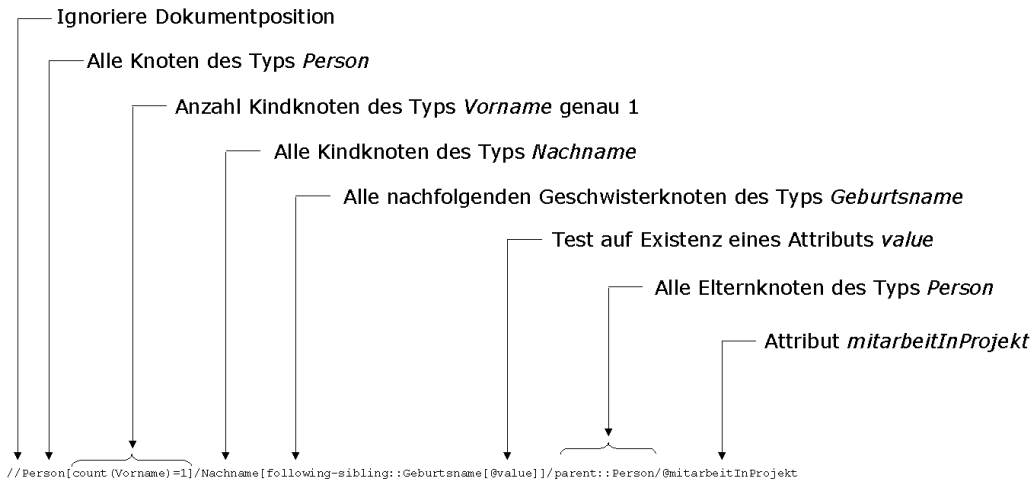
Operator	Funktionalität
+	Addition
-	Subtraktion als zweistelliger Operator. Der einstellige Operator - ist nicht spezifiziert, er liefert üblicherweise die negative Zahlendarstellung.
*	Multiplikation. Außer wenn innerhalb von XPath-Ausdrücken als Knotentest eingesetzt.
div	Division Achtung: Das Symbol / dient ausschließlich als Trennzeichen zur Separierung von Lokalisierungspfaden!
mod	Rest einer ganzzahligen Division

Ein umfangreiches Beispiel: Für das nachfolgende Beispiel wird das Projektverwaltungsdokument erweitert zu:

Beispiel 38: Erweiterte Projektverwaltung

```
(1)<?xml version="1.0" encoding="ISO-8859-15"?>
(2)<ProjektVerwaltung xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="L:\vorlesung\xml\examples
\projektverwaltung3.xsd">
(3)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(4)             <Vorname>Hans</Vorname>
(5)             <Nachname>Hinterhuber</Nachname>
(6)     </Person>
(7)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(8)             <Vorname>Franz</Vorname>
(9)             <Vorname>Xaver</Vorname>
(10)            <Nachname>Obermüller</Nachname>
(11)            <Qualifikationsprofil>
(12)                    <xhtml:u>IT-Kompetenz</xhtml:u>
(13)                    <xhtml:em>verschiedene</xhtml:em> Betriebssysteme und
(14)        <Leistungsstufe>professionelle</Leistungsstufe>
(15)                    <xhtml:em>
(16)                            <Qualifikation>Programmierung</Qualifikation>
(17)                    </xhtml:em>
(18)        verschiedener Programmiersprachen
(19)        <xhtml:em>
(20)                            <xhtml:u>
(21)                                    <Qualifikation>Entwickler</Qualifikation>
(22)                            </xhtml:u>
(23)                    </xhtml:em> von 1988-1990
(24)        <xhtml:u>
(25)                            <Qualifikation>Projektleiterfunktion</Qualifikation>
(26)                    </xhtml:u>
(27)        von <xhtml:b>1990-93</xhtml:b> im X42-Projekt in Abteilung AB&amp;amp;C
(28)      </Qualifikationsprofil>
(29)    </Person>
(30)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(31)            <Vorname>Fritz</Vorname>
(32)            <Nachname>Meier</Nachname>
(33)            <Geburtsname value="Huber"/>
(34)    </Person>
(35)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(36)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(37)</ProjektVerwaltung>
```

Download des Beispiels



Der XPath-Ausdruck der Abbildung 20 lokalisiert den Attributknoten des Inhalts `Prj02`.

Übung 2: Einige Übungen

Welches Ergebnis liefern folgende XPath-Ausdrücke?
(a) `//Person[//child::Qualifikationsprofil]/Nachname`
(b) `//Person[parent::ProjektVerwaltung]/Vorname[following-sibling::Vorname]`
(c) `/ProjektVerwaltung/Person[attribute::PersID='Pers01']//Nachname`

Wie muß ein XPath-Ausdruck lauten, um folgendes zu selektieren?
(d) Selektion aller Personen mit Nachnamen „Obermüller“.
(e) Selektion aller Nachnamen von Personen die über mehr als eine Qualifikation verfügen.
(f) Selektion der Nachnamen aller Projektleiter.

Anwendungsbeispiel: Integritätsbedingungen in XML-Schema

Über die Möglichkeiten der Datentypen hinausgehend bietet XML-Schema das Element `unique` zur Definition eindeutiger Wertbelegungen an. Hierbei wird auf die Lokatorsprache XPath zurückgegriffen um die abzuprüfenden Knoten innerhalb des Dokuments zu bezeichnen.

Die Syntax verwendet XPath-Ausdrücke eingeschränkter Mächtigkeit sowohl zur Festlegung des der Knotenmenge, auf die sich die Einschränkung bezieht (`selector`), als auch zur Angabe der eingeschränkten Knoten (`field`) selbst.

```
(1)<xsd:unique name="aName">
(2)     <xsd:selector xpath="aValidXPath"/>
(3)     <xsd:field xpath="aFieldStatement"/>
(4)     ...
(5)</xsd:unique>
```

Die Mächtigkeit der XPath-Ausdrücke ist dahingehend eingeschränkt, daß für das `selector`-Element ausschließlich Ausdrücke erlaubt sind, die Kindelemente des Knotens liefern, in dessen Kontext die durch `unique` formulierte Einschränkung angegeben wird. Als Konsequenz ist die Nutzung der verfügbaren XPath-Achsen auf diejenigen beschränkt, die Element-Knotenmengen zurückliefern.
Die Lokationsausdrücke in den -- möglicherweise mehrfach auftretenden -- `field`-Elementen werden relativ zum Pfad des `selector`-Knotens interpretiert. Hintereinandergesetzt muß der Pfad eines `selector`-Elements, gefolgt von einem Pfad eines `field`-Elements, einen gültigen Lokationsausdruck ergeben, der genau einen Knoten oder genau ein Attribut in der Ergebnismenge liefert. Sind mehrere `field`-Elemente zu einem `selector`-Element gegeben, so werden diese als durch logisches *und* verknüpft interpretiert. Mithin entspricht diese Semantik einem *concatenated primary key* aus den relationalen Datenbanken.

Das Beispiel zeigt die Nutzung des `unique`-Konstrukts zur Angabe der Eindeutigkeitsbedingung für das Attribut `PersID` des Elements `Person`.
Zunächst selektiert der Pfad `/Person` alle Knoten des gleichnamigen Typs; durch das `field`-Element wird die Eindeutigkeitsbedingung auf alle Attribut-Kindnoten des Typs `PersID` der Knoten in der selektierten Knotenmenge angewendet.
Die Semantik ist damit zur bisherigen `ID`-Typisierung identisch.

Beispiel 39: Unique-Einschränkung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/
>
(10)                    <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="@PersID"/>
(17)    </xsd:unique>
(18)</xsd:element>
(19)
(20)<xsd:complexType name="PersonType">
(21)    <xsd:attribute name="PersID" type="xsd:token"/>
(22)</xsd:complexType>
(23)
(24)<xsd:complexType name="ProjektType"/>
(25)
(26)</xsd:schema>
```

Download des Beispiels

Das nächste Beispiel zeigt die Verwendung mehrerer `field`-Elemente zur Realisierung zusammengesetzter Schlüssel.
Hierzu wird die Kombination aus dem Inhalt des `Nachnamen`- und des `Vornamen`-Elements zusammen als eindeutig deklariert.
Überdies zeigt das Beispiel die Anwendung des Schlüsselmechanismus auf Elemente ohne Änderung der Basissyntax, abgesehen von der geänderten XPath-Achse.

Beispiel 40: Zusammengesetzter Schlüssel innerhalb eines unique-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/
>
(10)                    <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="Vorname"/>
(17)            <xsd:field xpath="Nachname"/>
(18)    </xsd:unique>
(19)</xsd:element>
(20)
(21)<xsd:complexType name="PersonType">
(22)    <xsd:sequence>
(23)            <xsd:element name="Vorname" type="xsd:token" minOccurs="1"
maxOccurs="unbounded"/>
(24)            <xsd:element name="Nachname" type="xsd:token" maxOccurs="1"/>
(25)    </xsd:sequence>
(26)</xsd:complexType>
(27)
(28)<xsd:complexType name="ProjektType"/>
(29)
(30)</xsd:schema>
```

Download des Beispiels

Zur Realisierung von wertdefinierenden Schlüsselbeziehungen bietet XML-Schema die Elemente key und keyref an. Sie werden verwendet um sicherzustellen, daß ein Element oder Attribut nur einen Wert annehmen darf, der bereits an anderer Stelle im Instanzdokument auftritt.
Hierzu lokalisiert key auf der Basis eines XPath-Ausdruckes eine Referenzmenge, während keyref diejenige Knotenmenge lokalisiert, in der ausschließlich Elemente der Referenzmenge enthalten sein dürfen.
Das Beispiel zeigt die Anwendung auf das Element ProjektVerwaltung. Der mit *projectKey* benannte Schlüssel definiert die Referenzmenge als das Ergebnis der Anfrage Projekt/@ID, worauf die *projectReference* Bezug nimmt.

Beispiel 41: Schlüsselbasierte Referenzierung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xsd:element name="ProjektVerwaltung">
(7)             <xsd:complexType>
(8)                     <xsd:sequence>
(9)                             <xsd:element name="Person" type="PersonType"
maxOccurs="unbounded"/>
(10)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)                    </xsd:sequence>
(12)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)            </xsd:complexType>
(14)
(15)            <xsd:key name="projectKey">
(16)                    <xsd:selector xpath="Projekt"/>
(17)                    <xsd:field xpath="@ID"/>
(18)            </xsd:key>
(19)            <xsd:keyref name="projectReference" refer="projectKey">
(20)                    <xsd:selector xpath="Person"/>
(21)                    <xsd:field xpath="@mitarbeitInProjekt"/>
(22)            </xsd:keyref>
(23)    </xsd:element>
(24)
(25)    <xsd:complexType name="PersonType">
(26)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:token"/>
(27)    </xsd:complexType>
(28)    <xsd:complexType name="ProjektType">
(29)            <xsd:attribute name="ID" type="xsd:token"/>
(30)    </xsd:complexType>
(31)</xsd:schema>
```

Download des Beispiels

Web-Referenzen 8: Weiterführende Links

- XPath Spezifikation
- Deutsche Übersetzung der XPath-Spezifikation
- XPath Visualisierer (Java-basiert)
- Visual XPath (.NET Windows-Applikation)

Originalbezugsquelle

- XPath Explorer (Java-basiert)

Originalbezugsquelle

- Online Experimentieren mit XPath

3 Datenbankzugriff

3.1 Java Database Connectivity (JDBC)

Motivation

Häufig besteht der Wunsch oder die Notwendigkeit, auf bereits vorliegende Datenbestände, die durch ein

Datenbankmanagementsystem (DBMS) verwaltet werden, in einer Applikationsprogrammiersprache zuzugreifen. Dabei soll die Anbindung der benötigten Datenquelle nicht problemspezifisch wieder und wieder neu entwickelt werden, sondern sollte sich auf ähnliche Datenanbindungsprobleme übertragen lassen.
Vor diesem Hintergrund liegt es nahe, sich an den Typen der verfügbaren und kommerziell bedeutsamen DBMS zu orientieren und herstellerspezifische Entwicklungen außer Acht zu lassen. Gleichzeitig offenbaren sich hierbei Standardisierungsbemühungen wie die Sprache *SQL* zum Zugriff auf relationale DBMS als lohnenswerter Ansatz der Etablierung einer generischen und übertragbaren Schnittstelle.

Die Idee zur Schaffung einer solchen generischen Schnittstelle für den Zugriff auf relationale DBMS geht zurück auf eine Initiative der *SQL Access Group*, welche später in der Vereinigung mit der *X/Open* Group aufging, die zwischenzeitlich in *Open Group* umbenannt wurde. Das dort konzipierte programmiersprachenunabhängige *SQL Call Level Interface* (SQL/CLI) konnte sich dank der Umsetzung unter dem Namen Open Database Connectivity (ODBC) durch die Firma Microsoft und die parallel erfolgte internationale Normierung unter dem Titel *SQL/CLI* breit am Markt etablieren.

Die für die Programmiersprache Java adaptierte Variante des Zugriffs auf relationale DBMS wird durch SUN Microsystems unter dem Namen *Java Database Connectivity* (JDBC) propagiert und stellt eine auf ODBC konzeptionell aufbauende und auf die spezifischen Bedürfnisse dieser Applikationsprogrammiersprache optimierte Untermenge des SQL/CLI-Standards dar.

Konzept und Grundidee

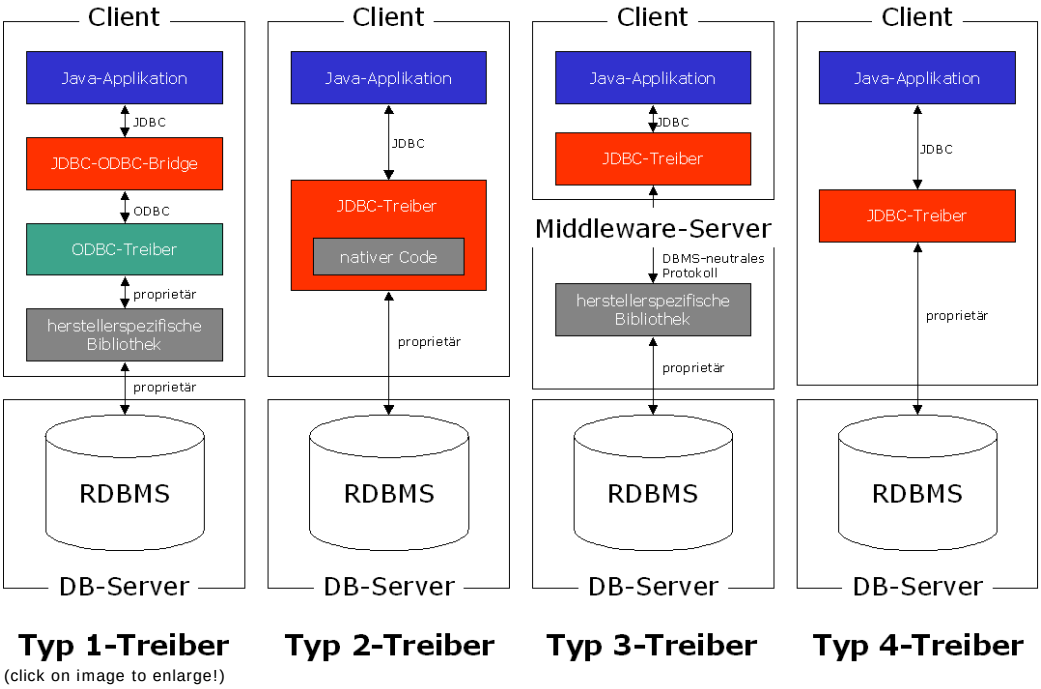
Von den Vorgängeransätzen übernommene Grundidee der Schnittstelle ist es den physischen Zugriff auf das Datenbankmanagementsystem durch eine von der Applikation spearierte wiederverwendbare Softwarekomponente, den sog. *JDBC-Treiber*, abzuwickeln.

Dieser Treiber vermittelt zwischen der Javaapplikation und dem verwendeten DBMS. Hierbei muß für jedes DBMS ein auf es abgestimmter JDBC-Treiber verwendet werden, da lediglich die Schnittstelle zur Applikation, nicht jedoch die zum DBMS, standardisiert ist.

Diesem Treiber obliegt die Abwicklung der gesamten Kommunikationsvorgänge mit dem DBMS. Er setzt jedoch selbst keine datenbankspezifischen Funktionalitäten, wie Syntax- oder Plausibilitätsprüfungen der übermittelten Kommandos um. Etwaige Fehlerprüfungen können, ebenso wie Anfrageoptimierungen, daher erst seitens des DBMS vorgenommen werden.
Der Vorteil dieses Vorgehens liegt in der Generizität des JDBC-Treibers. Er kann ohne aufwendige Logikanteile als reine uninterpretierende Vermittlungsschicht zwischen Applikation und DBMS umgesetzt werden, wodurch schlanke Implementierungen ermöglicht werden.

Die JDBC-Spezifikation detailliert den Treiberbegriff zusätzlich hinsichtlich der gewählten technischen Umsetzung aus. So werden die vier in Abbildung 3 dargestellten Treibertypen gemäß ihrer Charakteristika beschrieben und unterschieden.

Abbildung 3: *JDBC-Treibertypen*



(click on image to enlarge!)

Die historisch älteste Variante bildet der **Typ 1 Treiber**. Strenggenommen verkörpert er selbst keinen Datenbanktreiber, sondern lediglich eine Umsetzungsschicht die einem existierenden ODBC-Treiber vorgeschaltet wird.
Die Abbildung belegt diesen Treibertyp daher mit dem Begriff *JDBC-ODBC-Bridge*, da er lediglich den

Brückenschlag zwischen den beiden Standards vornimmt und sich in der konkreten Anwendung auf die Umsetzung zwischen den beiden Protokollen beschränkt, ohne realen Zugriff auf die Datenbank zu erhalten.
Dieser ist dem ODBC-Treiber vorbehalten, der im allgemeinen Falle mit einer weiteren Umsetzungsstufe kommuniziert, welche die generischen ODBC-Aufrufe in konkrete DBMS-spezifische wandelt.
Während sowohl der JDBC-ODBC-Brückentreiber als auch der ODBC-Treiber selbst für verschiedene DBMS verwendet werden können, muß für jedes konkrete DBMS eine herstellerspezifische, d.h. an das verwendete DBMS angepaßte, Bibliothek vorliegen.

Für den Fall eines **Typ 2 Treiber**s entfällt diese durch ODBC geschaffene zusätzliche Indirektionsstufe zugunsten der Adaption der Konversionskomponente, welcher die Wandlung der Aufrufe in das DBMS-native Protokoll obliegt, an das JDBC-Protokoll und ihrer Integration in den JDBC-Treiber selbst.
Die Natur der Kommunikation des Java-Anteils des Treibers mit den Nativen ist im Rahmen der durch die JDBC-Spezifikation gegebenen Definition nicht festgelegt.
Durch die integration der DBMS-nativen Treiberanteile in den JDBC-Treiber muß dieser für jedes anzusprechende DBMS neu erstellt werden. Eine Wiederverwendung der JDBC-spezifischen Anteile, die für die Clientkommunikation eingesetzt werden, kann hierbei nicht erfolgen.

Der Fall der (partiellen) Konkretisierung dieser Kommunikationsbeziehung zu einem beliebigen *DBMS-neutralen Protokoll* wird durch einen **Typ 3 Treiber** aufgegriffen.
Hier wird die DBMS-spezifische Komponente (in der Abbildung grau dargestellt) als vom JDBC-Treiber separiertes Modul aufgefaßt, daß mit diesem mittels eines festgelegten neutralen Protokolls kommuniziert.
Durch diese Separierung, die auch durch Installation auf physisch getrennten Maschinen --- der DBMS-spezifische Anteil könnte beispielsweise auf einem Middleware-Server untergebracht werden --- fundiert werden kann, gelingt die Wiederverwendung des JDBC-Treiberanteils, der mit verschiedenen DBMS-spezifischen Bibliotheken über das gewählte Protokoll kommunizieren kann.

Der **Typ 4 Treiber** stellt die letzte durch die JDBC-Spezifikation vorgesehene Ausprägung dar. Er konzipiert eine vollständig in Java implementierte Zugriffsschicht, die in sich geschlossen ist. Sie besitzt daher lediglich die notwendige JDBC-Schnittstelle zur Kommunikation mit der Java-Applikation und eine DBMS-Spezifische zum Zugriff auf die Datenquelle.
Die Vorteile dieser Architekturvariante liegen in ihrer Portabilität und den geringen Installations und Wartungsaufwänden, die aus der Reduktion der Kommunikationsbeziehungen resultieren. So kann ein solcher Treiber durch einfache Integration in die Java-Applikation verwendet werden und bedarf keiner Installationen oder Modifikationen an der verwendeten Ausführungsumgebung.
Gleichzeitig offenbart sich diese Lösung jedoch als technisch aufwendig in der Umsetzung, sobald DBMS verschiedener Hersteller angesprochen werden sollen, da die JDBC-Anteile des Treibers nicht separat wiederverwendet werden können.

Hinsichtlich des Laufzeitverhaltens zeigt sich deutlich die Schwäche der Typ 1 Treiber, welche in der inhärent notwendigen Doppelkonversion (JDBC zu ODBC und ODBC zu nativem Aufruf) begründet liegt.
Daher sind Treiber dieses Typs als Übergangserscheinung hin zu „echten“ JDBC-Treibern, d.h. Treibern der restlichen Typen, anzusehen und sollten in Produktivumgebungen nicht eingesetzt werden.
Die Vorteile der Typ 2 und 3 Treiber seitens der Ausführungsgeschwindigkeit liegen in den nativen Codeanteilen begründet, welche für das jeweilige verwendete DBMS optimiert werden können.
Zwar spricht der leichte Installations- und Adminstrationsaufwand eindeutig für Typ 4 Treiber, jedoch fallen diese in ihrer Leistungsfähigkeit durch die ausschließliche Verwendung der Programmiersprache Java teilweise deutlich hinter Treiber des Typs 2 und 3, mit unter sogar hinter solche des Typs 1, zurück. Sie verkörpern jedoch den aus konzeptioneller Sicht zu bevorzugenden Ansatz hinsichtlich Portabilität und Vergleichbarkeit der erzielten quantitativen Ergebnisse.
Typischerweise kommen im produktiven Einsatz jedoch Treiber der Typen 2 und 4 zum Einsatz, die entweder durch den Hersteller des DBMS mitgeliefert werden (Typ 2) oder auf der Basis publizierter Schnittstellen plattformunabhängig für genau ein spezifisches DBMS entwickelt wurden (Typ 4).

Generell formuliert das JDBC-Konzept auf dieser Ebene noch keine Einschränkung hinsichtlich der unterstützten DBMS-Typen und ist generell auf verschiedenste Datenquellen anwendbar. Durch die Struktur des API und die verfügbaren Treiber kristallisieren sich jedoch relationale DBMS als Hauptanwendungsgebiet dieser Zugriffsschnittstelle heraus.

Im folgenden wird die Verwendung des Typ 4 Treibers Connector/J im Zusammenspiel mit dem RDBMS *MySQL* betrachtet.

Die Beispiele basieren auf einer Demodatenbank, deren Struktur und Inhalte nachfolgend angegeben sind.

Die Tabelle EMPL0YEE

```
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
| FNAME    | MINIT | LNAME   | SSN       | BDATE      | ADDRESS                  | SEX  |
SALARY   | SUPERSSN  | DNO  |
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
```

```
| John     | B     | Smith   | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M    |
30000.00 | 333445555 |    5 |
| Franklin | T     | Wong    | 333445555 | 1955-12-08 | 638 Voss, Houston, TX    | M    |
40000.00 | 888665555 |    5 |
| Joyce    | A     | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX   | F    |
25000.00 | 333445555 |    5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M    |
38000.00 | 333445555 |    5 |
| James    | E     | Borg    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX   | M    |
55000.00 |      NULL |    1 |
| Jennifer | S     | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX  | F    |
43000.00 | 888665555 |    4 |
| Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX  | M    |
25000.00 | 987654321 |    4 |
| Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX  | F    |
25000.00 | 987654321 |    4 |
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
```

Umsetzung in der Java-API

Das Klassendiagramm der Abbildung 4 zeigt die zentralen Klassen des Paketes `java.sql`.
Auffallend ist, daß alle Elemente des dargestellten Pakets -- abgesehen von den definierten Exceptionklassen -- als Schnittstellen ausgelegt sind. Durch diese Mimik wird die Organisation der JDBC-Schnittstelle deutlich. Die API legt lediglich das Verhalten hinsichtlich seiner Semantik und die Einzeloperationen durch Definition ihrer Parameter fest, die konkrete DBMS-spezifische Implementierung dieser Operationen wird durch den JDBC-Treiber bereitgestellt.

Zentrale Klasse der JDBC-API ist die Schnittstelle `Connection`. Sie bildet die Kommunikationsverbindungen zum DBMS ab und bietet notwendige Verwaltungsoperationen. Hierunter fallen insbesondere auch die Aufrufe zur Transaktionssteuerung.

Die Schnittstelle `Statement` realisiert genau eine aus Javasicht atomare Datenbankaktion. Diese muß hierbei aus minimal einem Aufruf an das DBMS bestehen, kann aber eine Reihe separater Aufrufe zu einem *Batch* bündeln.
Als Sonderform sieht die API die Spezialisierung `PreparedStatement` vor, die es gestattet, parametrisierte Anfragen zwischenzuspeichern, die nach Belegung der Parameterfelder an das DBMS übergeben werden. Hierdurch wird ein einfacher Mechanismus zur Wiederverwendung von DBMS-Aufrufen etabliert.

Liefert eine DBMS-Anfrage Ergebnistupel, so werden diese konform zur Schnittstelle `ResultSet` verwaltet. Diese Schnittstelle erlaubt die lesende Traversierung der vom DBMS gelieferten Tupel ebenso wie ihre Aktualisierung im Hauptspeicher und das anschließende Zurückschreiben in die Datenbank.
Die in der Abbildung nur durch *getXXX* und *updateXXX* angedeuteten Operationen existieren in Ausprägungen für alle unterstützten Datentypen, wobei *XXX* den Namen des Typs bezeichnet.

Ferner definiert die API mit `SQLWarning` eine Ausnahme zur Behandlung auftretender Fehlersituationen sowie eine Reihe weiterer, in der Abbildung 4 nicht dargestellter Klassen wie beispielsweise verschiedene Datentypen.

Die Klasse `SQLException` bietet durch ihre Methoden `getErrorCode` und `getSQLState` Möglichkeiten an um die nähere Ursache eines datenbankseitigen Fehlers zu ermitteln.
Zusätzlich gestatten Objekte dieses Ausnahmetyps die Verschachtelung von Ausnahmen, d.h. die rekursive Einbettung eines Ausnahmeereignisobjekts in ein bestehendes. Auf diesem Wege können aufgetretene Fehler durch mehrere Ausnahmeobjekte näher spezifiziert werden.
Beispiel 42 zeigt die Abfrage von Details der empfangenen und aller eingebetteten Ausnahmeereignisobjekte mittels der durch die JDBC-API vorgesehenen Methoden.

Beispiel 42: Ermittlung von Fehlerdetails

```
(1)try {
(2)     // JDBC code
(3)} catch (SQLException e) {
(4)     while (e != null) {
(5)             System.err.println("SQLState: " + e.getSQLState());
(6)      System.err.println("Message:  " + e.getMessage());
(7)      System.err.println("Vendor:   " + e.getErrorCode());
(8)      System.err.println("-----");
(9)      e = e.getNextException();
(10)    }
(11)}
```

Download des Beispiels

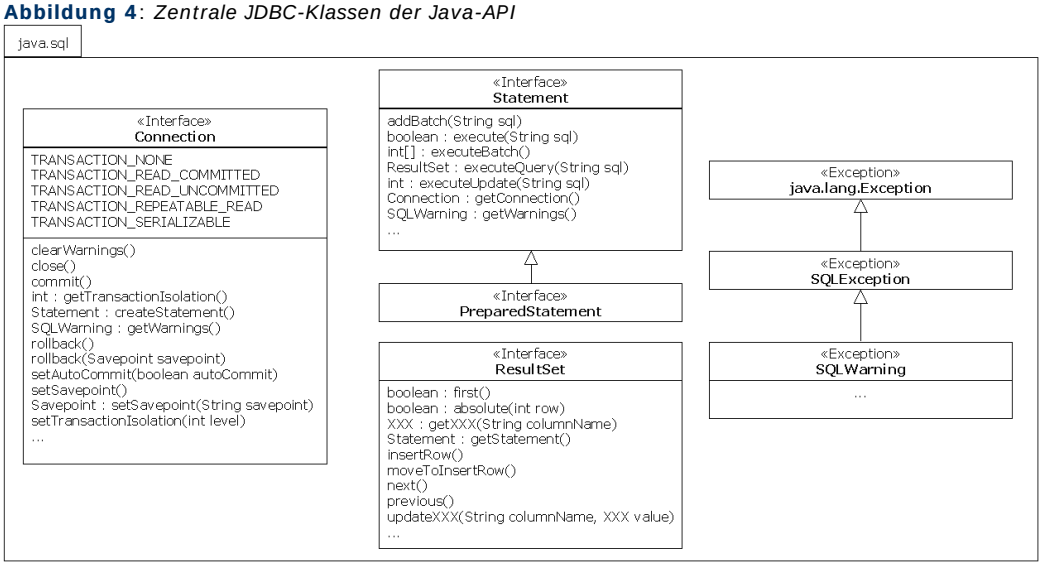
Mit der Version 1.4 der Java-Standard-Edition wurde die zuvor nur in der JDBC-API zur Verfügung stehende Möglichkeit zur Schachtelung von Ausnahmeereignissen auch für beliebige Ausnahmeereignisobjekte des Typs `Throwable` definiert.
Anders als die JDBC-API sieht die generische Lösung jedoch die Nutzung der Methode `getCause` zur Extraktion der eingebetteten Ausnahmeereignisobjekte vor.
Der Code des Beispiels 43 spiegelt daher die Standard-API-konforme Realisierung wieder. Zusätzlich wendet die Lösung die Standard-Methode `getMessage` zur Ermittlung der deskriptiven Fehlerbeschreibung an.

Beispiel 43: Standard-API-konforme Ermittlung von Fehlerdetails

```
(1)try {
(2)     // Normal code
(3)} catch (SQLException e) {
(4)     Throwable t = e;
(5)     while (t != null) {
(6)     System.err.println("Type: " + t.getClass().getName());
(7)      System.err.println("Message: " + t.getMessage());
(8)      System.err.println("-----");
(9)      t = t.getCause();
(10)    }
(11)}
```

Download des Beispiels

Abbildung 4: *Zentrale JDBC-Klassen der Java-API*



(click on image to enlarge!)

Zugriff auf die Datenbank

Beispiel 44 zeigt den Ablauf zur Aufnahme einer Verbindung mit der Datenbank `jdbctest` auf dem lokalen Rechner (`localhost`).

Zunächst muß die Klasse des gewählten JDBC-Treibers (im Beispiel `com.mysql.jdbc.Driver` vor ihrer Verwendung geladen werden. Dies geschieht durch den Aufruf der statischen Methode `forName` auf der Klasse `Class`.
Der zu ladende Treiber muß hierbei die JDBC-Schnittstellenklasse `Driver` implementieren um später durch die JDBC-API verwendet werden zu können.
Gleichzeitig mit dem dynamischen Ladevorgang erfolgt die Registrierung des Treibers beim JDBC-`DriverManager`, der die Verwaltung der geladenen DB-Treiber übernimmt.

Nach dem erfolgreichen Laden des Treibers wird durch den Aufruf von `getConnection` (Zeile 16) die Verbindung zur Datenbank hergestellt. Die anzusprechende Datenbank wird hierbei durch eine URI der Form `jdbc:mysql://DB-Server/DB-Name` repräsentiert (Zeile 17). Zusätzlich können ein zur Anmeldung am DB-System benötiger Benutzer (Zeile 18) und sein Paßwort (Zeile 19) übergeben werden.

Beispiel 44: Aufbau einer Datenbankverbindung

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)
(5)public class JDBCConnect {
(6)     public static void main(String[] args) {
(7)             try {
(8)                     Class.forName("com.mysql.jdbc.Driver");
(9)             } catch (ClassNotFoundException e) {
(10)                    System.err.println("Driver class not found");
(11)                    e.printStackTrace();
(12)            }
(13)            Connection con = null;
(14)            try {
(15)                    con =
(16)                            (Connection) DriverManager.getConnection(
(17)                                    "jdbc:mysql://localhost/jdbctest/",
(18)                                    "mario",
(19)                                    "thePassword");
(20)            } catch (SQLException e1) {
(21)                    System.err.println("Error establishing database connection");
(22)                    Throwable t = e1;
(23)                    while (t != null) {
(24)                    System.err.println("Type: " + t.getClass().getName());
(25)                    System.err.println("Message: " + t.getMessage());
(26)                    System.err.println("-----");
(27)                    t = t.getCause();
(28)                    }
(29)            }
(30)    }
(31)}
```

Download des Beispiels

Zusätzlich stellen die Klassen `Driver` und `DriverManager` die Möglichkeit der Abfrage von verbindungsunabhängigen Verwaltungsinformationen zur Verfügung.

Beispiel 45: Ermittlung von Informationen über Treiber und Treibermanager

```
(1)import java.sql.Driver;
(2)import java.sql.DriverManager;
(3)import java.util.Enumeration;
(4)
(5)public class JDBCDriver {
(6)
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)
(15)            System.out.println(
(16)                    "DriverManager:\nlogin timeout=" + DriverManager.getLoginTimeout
()));
(17)
(18)            Enumeration e = DriverManager.getDrivers();
(19)            while (e.hasMoreElements()) {
(20)                    Driver drv = (Driver) e.nextElement();
(21)
(22)                    System.out.println(
(23)                            "Driver="
(24)                                    + drv.getClass().getName()
(25)                                    + "\nmajor version="
(26)                                    + drv.getMajorVersion()
(27)                                    + "\nminor version="
(28)                                    + drv.getMinorVersion()
(29)                                    + "\nJDBC compliant="
(30)                                    + drv.jdbcCompliant());
(31)            }
(32)
(33)    }
(34)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 45 zeigt die Ermittlung des durch den DriverManager für alle durch ihn verwalteten Treiber global definierten Login Timouts, der angibt wie lange beim Anmeldevorgang an der Datenbank auf eine Rückmeldung gewartet wird.
Zusätzlich werden für alle verwalteten Treiber der Klassenname sowie Daten zur Version und zum Stand der JDBC-Unterstützung ermittelt und ausgegeben.
Der JDBC-Unterstützungsstand gibt an, ob ein gegebener Treiber die Konformitätstests der Firma SUN bestanden hat. Voraussetzung hierfür ist u.a. die vollständige Unterstützung des SQL 92-Standards (entry level).
Diese Interpreatation von Spezifikationskonformität verwundert etwas, da alle JDBC-Treiber mit Ausnahme der inhärent DB-neutralen Typ 1 Treiber DBMS-spezifisch realisiert sind. Aus diesem Grunde bewertet der Konformitätstest vielmehr den Umsetzungsgrad des SQL-Standards in dem via JDBC genutzten DBMS als die Güte des JDBC-Treibers selbst.

Seit der JDBC-Schnittstellenversion 2 ist neben der „klassischen" Zugriffsvariante auch eine auf dem Java Naming and Directory Interface (JNDI) basierende Zugriffsmethodik definiert, deren Verwendung --- abgesehen von der geänderten Mimik im Aufbau der DB-Verbindung --- identisch gestaltet ist.

Jedoch ist, wie in JNDI üblich, vor dem Zugriff ein benanntes Objekt beim JNDI-Dienst zu registrieren.
Im Falle von JDBC ist dies ein Objekt welches die Schnittstelle DataSource implementiert.

Der Code des Beispiels 46 zeigt die notwendigen Schritte zur Registrierung eines MysqlDataSource-Objekts, der durch den MySQL-JDBC-Treiber gelieferten Implementierung der Schnittstelle DataSource.

Beispiel 46: Ablage von Verbindungsinformation in einem JNDI-Verzeichnis

```
(1)import java.util.Hashtable;
(2)import javax.naming.Context;
(3)import javax.naming.InitialContext;
(4)import javax.naming.NamingException;
(5)import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
(6)
(7)public class JDBCConnect2Server {
(8)
(9)     public static void main(String[] args) {
(10)            Hashtable env = new Hashtable();
(11)            env.put(
(12)                    Context.INITIAL_CONTEXT_FACTORY,
(13)                    "com.sun.jndi.fscontext.RefFSContextFactory");
(14)            env.put(Context.PROVIDER_URL, "file:/tmp/registry");
(15)
(16)            MysqlDataSource ds = new MysqlDataSource();
(17)            ds.setDatabaseName("jdbctest");
(18)            Context ctx = null;
(19)            try {
(20)                    ctx = new InitialContext(env);
(21)            } catch (NamingException ne) {
(22)                    ne.printStackTrace();
(23)            }
(24)
(25)            try {
(26)                    ctx.rebind("jdbc/mySrc", ds);
(27)            } catch (NamingException ne) {
(28)                    ne.printStackTrace();
(29)            }
(30)    }
(31)}
```

Download des Beispiels

Entsprechend der modifizierten Ablage der Verwaltungsinformation ändert sich die Erzeugung der Datenbankverbindung beim Zugriff. Hier wird nun zunächst über einen Zugriff auf den JNDI-Verzeichnisdienst das benannte DataSource-Objekt (es trägt den Namen jdbc/mySrc ermittelt.
Anschließend wird durch das dem Verzeichnisdienst entnommene DataSource-Objekt die Datenbankverbindung (d.h. das Connection-Objekt) erzeugt.
Alle weiteren Schritte zur Interaktion mit der Datenbank verlaufen dann identisch zur im Beispiel 44 gezeigten Verbindungsaufnahme.
Der Code des Beispiels 47 zeigt die notwendigen Schritte zur Ermittlung der Referenz auf das Objekt des Typs DataSource aus dem JNDI-Verzeichnis, sowie die Erzeugung des Connection-Objekts.

Beispiel 47: Verbindungsaufbau unter Nutzung von JNDI

```
(1)import java.sql.Connection;
(2)import java.sql.SQLException;
(3)import java.util.Hashtable;
(4)import javax.naming.Context;
(5)import javax.naming.InitialContext;
(6)import javax.naming.NamingException;
(7)import javax.sql.DataSource;
(8)
(9)public class JDBCConnect2 {
(10)    public static void main(String[] args) {
(11)            Hashtable env = new Hashtable();
(12)            env.put(
(13)                    Context.INITIAL_CONTEXT_FACTORY,
(14)                    "com.sun.jndi.fscontext.RefFSContextFactory");
(15)            env.put(Context.PROVIDER_URL, "file:/tmp/registry");
(16)            Context ctx = null;
(17)            try {
(18)                    ctx = new InitialContext(env);
(19)            } catch (NamingException ne) {
(20)                    ne.printStackTrace();
(21)            }
(22)            DataSource ds = null;
(23)            try {
(24)                    ds = (DataSource) ctx.lookup("jdbc/mySrc");
(25)            } catch (NamingException ne) {
(26)                    ne.printStackTrace();
(27)            }
(28)            Connection con = null;
(29)            try {
(30)                    con = ds.getConnection("mario", "thePassword");
(31)            } catch (SQLException sqle) {
(32)                    Throwable t = sqle;
(33)                    while (t != null) {
(34)                    System.err.println("Type: " + t.getClass().getName());
(35)                    System.err.println("Message: " + t.getMessage());
(36)                    System.err.println("-----");
(37)                    t = t.getCause();
(38)                    }
(39)            }
(40)    }
(41)}
```

Download des Beispiels

Auffallend ist die Ablage des Datenbanknamens im Verzeichnisdienst mittels des Methodenaufrufs `setDatabaseName`. Diese Verschiebung der Information wird durch die geänderte Mimik der Erzeugung des `Connection`-Objekts impliziert. So sieht die Implementierung dieser Methode für die Klasse `DataSource` keine Möglichkeit zur gleichzeitigen Übergabe von Anmeldenamen, Paßwort und Datenbank vor. Vielmehrnoch ist es sogar möglich diese Daten allesamt innerhalb des JNDI-Verzeichnisdienstes abzulegen. (Für diesen Zweck stehen die Methoden `setUser` bzw. `setPassword` zur Verfügung.) Als Konsequenz hiervon kann der Verbinungswunsch durch Aufruf der Methode `getConnection` ohne weitere Parameter erfüllt werden.
Diese Umsetzungsweise ist vor ihrer Realisierung hinsichtlich des damit eintretenden Verlustes an Sicherheit zu prüfen, da in ihrer Folge eine Datenbankverbindung allein durch Kenntnis des JNDI-residenten Namens des `DataSource`-Objektes erfolgen kann.

Generell wählen JDBC-Umsetzungen den Weg, jede Ausprägung eines `Connection`-Objekts in eine physische Datenbankverbindung abzubilden. Dieses, durchaus der intuitiven Semantik der `Connection`-Klasse entsprechende Vorgehen kann jedoch in realen Applikationen, begründet in der Vielzahl der durch das DBMS zu verwaltenden Verbindungen, zu Zugriffsengpässen führen.
Aus diesem Grunde definiert die JDBC-Schnittstelle Operationen zur Zusammenfassung „gleichartiger“ Zugriffe. Hierzu zählen Zugriffe die unter derselben Nutzerkennung auf dieselbe Datenbank abgewickelt werden. Diese Zugriffsform tritt insbesondere bei Anwendungen auf, die über nur einen in der Datenbank eingetragenen Anwender verfügen und die gesamte Nutzerverwaltung datenbanktransparent applikationsseitig abwickeln.
Zur Optimierung von Zugriffen dieser Natur sieht die JDBC-Schnittstelle das sog. *Connection Pooling* vor, welches gleichartige Zugriffe bündelt.
Das Beispiel 48 zeigt eine Umsetzung:

Beispiel 48: Verbindungsaufbau unter Nutzung von Connection Pooling

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import javax.sql.PooledConnection;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.jdbc2.optional.MysqlPooledConnection;
(6)
(7)public class JDBCConnection3 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            PooledConnection pc = new MysqlPooledConnection(con);
(34)
(35)            java.sql.Connection con1 = null;
(36)            try {
(37)                    con1 = pc.getConnection();
(38)            } catch (SQLException sqle) {
(39)                    Throwable t = sqle;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)    }
(48)}
```

Download des Beispiels

Statt für jede gewünschte Datenbankverbindung ein zusätzliches Objekt des Type `Connection` zu erzeugen, wird die erzeugte Verbindung zur Konstruktion eines Objektes, welches Konform zur Schnittstelle `PooledConnection` definiert ist, verwendet. Dieses sorgt für die Verwaltung der DB-Verbindung und stellt dieselbe physische Verbindung verschiedenen Anfragern zur Verfügung. Konsequenterweise wird daher eine neue Verbindung nicht mehr vom `DriverManager` angefordert, sondern durch die Methode `getConnection` der aus der Verwaltungsstruktur entnommenen `PooledConnection` beantragt.

Aufgrund der Unterstützung des SQL-Sprachumfanges, durch unveränderte textuelle Propagation an das DBMS sind durch JDBC im Allgemeinen alle Facetten der Datenbanksprache nutzbar, sofern sie durch das verwendete DBMS Unterstützung finden. Hierunter fallen:

- Data Definition Language.
  Zur Erzeugung eines Datenmodells.
- Data Manipulation Language.
  Zur Modifikation der verwalteten Daten.
- Data Retrieval Language.
  Zur Anfrage der in einer Datenbank gespeicherten Daten.
- Data Control Language.
  Zur Festlegung und Kontrolle von Zugriffsberechtigungen.

JDBC reflektiert jedoch nicht diese Sprach(-sub-)klassen selbst in der API, sondern sieht vielmehr ausschließlich zwei Formen des Zugriffs vor. Solche die tabellenwerte Resultate liefern und solche, deren

Ausführung lediglich primitivwertige Rückgabewerte liefert.

Primitivwertige Zugriffe

Primitivwertige Datenbankzugriffe liefern, abgesehen von Fehler- oder Warnmeldungen, lediglich die Anzahl der geänderten Tupel, falls zutreffend, oder 0 zurück.
Aus dieser Festlegung lassen sich diejenigen SQL-Anweisungstypen ableiten, welche als primitivwertiger Zugriff realisiert sind. Hierunter fallen alle Operationen der Datendefinition wie `CREATE` oder `ALTER TABLE` sowie alle Einfüge- (`INSERT`) Änderungs- (`UPDATE`) und Löschvorgänge (`DELETE`). Darüberhinaus alle Operationen zur Administration der Datenbank durch Rechtevergabe (`GRANT`, `REVOKE`).

Zugriffe dieser Art werden generell durch die Methode `executeUpdate`, oder einer Abart davon, realisiert.

Beispiel 49: Erstellung einer neuen Tabelle

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCCreateTable {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement  stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)            String createTab = new String("CREATE TABLE EMPLOYEE(" +
(47)                    "FNAME VARCHAR(10) NOT NULL," +
(48)                    "MINIT VARCHAR(1)," +
(49)                    "LNAME VARCHAR(10) NOT NULL," +
(50)                    "SSN INTEGER(9) NOT NULL," +
(51)                    "BDATE DATE," +
(52)                    "ADDRESS VARCHAR(30)," +
(53)                    "SEX ENUM('M','F')," +
(54)                    "SALARY REAL(7,2) UNSIGNED," +
(55)                    "SUPERSSN INTEGER(9)," +
(56)                    "DNO INTEGER(1));");
(57)            try {
(58)                    System.out.println("result="+stmt.executeUpdate(createTab));
(59)            } catch (SQLException e3) {
(60)                    System.err.println("Error creating table EMPLOYEE");
(61)                    Throwable t = e3;
```

```
(62)                    while (t != null) {
(63)                    System.err.println("Type: " + t.getClass().getName());
(64)                    System.err.println("Message: " + t.getMessage());
(65)                    System.err.println("-----");
(66)                    t = t.getCause();
(67)                    }
(68)            }
(69)    }
(70)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 49 zeigt die notwendigen Schritte zur Erstellung der Tabelle EMPLOYEE in der Datenbank.

Nach dem (üblichen) Verbindungsaufbau (Zeile 8-24) wird in Zeile 27 eine Variable des Typs Statement deklariert. Auch bei Statement handelt es sich um eine durch die JDBC-API vordefinierte Schnittstelle, die als Bestandteil des JDBC-Treibers von einer Klasse implementiert wird.
Ausgehend von der etablierten Datenbankverbindung wird durch Aufruf der Methode createStatement eine konkrete Ausprägung konform zur Statement-Schnittstelle erzeugt (Zeile 29).

Der Aufruf von executeUpdate übergibt das als Zeichenkette abgelegte SQL-Kommando an die Datenbank zur Ausführung.
Da durch CREATE TABLE keine Tupeländerungen vorgenommen werden ist das Resultat des Aufrufs der Rückgabewert 0.

Beispiel 50 zeigt mit dem ALTER TABLE-Kommando eine weitere Anwendung der executeUpdate-Methode. Auch in diesem Falle wird als Resultat 0 geliefert, da die Definition des Primärschlüssels keine Änderungen an den verwalteten Datensätzen vornimmt.

Beispiel 50: Modifikation der Tabellendefinition

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCAlterTable {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                            System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
```

```
(44)                    }
(45)            }
(46)            String createTab =
(47)                    new String("ALTER TABLE EMPLOYEE ADD PRIMARY KEY (SSN);");
(48)            try {
(49)                    System.out.println("result=" + stmt.executeUpdate(createTab));
(50)            } catch (SQLException e3) {
(51)                    System.err.println("Error altering table EMPLOYEE");
(52)                    Throwable t = e3;
(53)                    while (t != null) {
(54)                    System.err.println("Type: " + t.getClass().getName());
(55)                    System.err.println("Message: " + t.getMessage());
(56)                    System.err.println("-----");
(57)                    t = t.getCause();
(58)                    }
(59)            }
(60)    }
(61)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 51: Einfügen von Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCInsert1 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('John', 'B', 'Smith', 123456789, '1965-01-09', '731 Fondren, Houston, TX',
'M', 30000, 333445555, 5);"));
(49)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
```

```
EMPLOYEE VALUES('Franklin', 'T', 'Wong', 333445555, '1955-12-08', '638 Voss, Houston, TX',
'M', 40000, 888665555, 5);"));
(50)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Alicia', 'J', 'Zelaya', 999887777, '1968-07-19', '3321 Castle, Spring,
TX', 'F', 25000, 987654321, 4);"));
(51)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Jennifer', 'S', 'Wallace', 987654321, '1941-06-20', '291 Berry, Bellaire,
TX', 'F', 43000, 888665555, 4);"));
(52)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Ramesh', 'K', 'Narayan', 666884444, '1962-09-15', '975 Fire Oak, Humble,
TX', 'M', 38000, 333445555, 5);"));
(53)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Joyce', 'A', 'English', 453453453, '1972-07-31', '5631 Rice, Houston,
TX', 'F', 25000, 333445555, 5);"));
(54)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Ahmad', 'V', 'Jabbar', 987987987, '1969-03-29', '980 Dallas, Houston,
TX', 'M', 25000, 987654321, 4);"));
(55)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('James', 'E', 'Borg', 888665555, '1937-11-10', '450 Stone, Houston, TX',
'M', 55000, null, 1);"));
(56)            } catch (SQLException e3) {
(57)                    System.err.println("Error inserting values into table EMPLOYEE");
(58)                    Throwable t = e3;
(59)                    while (t != null) {
(60)                    System.err.println("Type: " + t.getClass().getName());
(61)                    System.err.println("Message: " + t.getMessage());
(62)                    System.err.println("-----");
(63)                    t = t.getCause();
(64)                    }
(65)            }
(66)    }
(67)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 51 zeigt den Einfügevorgang von acht Werten in die durch die vorangegangenen Beispiele erzeugte Tabelle `EMPLOYEE`.
Jeder der Einfügevorgänge der Zeilen 36-43 führt im Rahmen einer separaten Datenbankkommunikation sequentiell genau einen Einfügevorgang durch, was durch den Rückgabewert 1 dokumentiert wird.

Zwar ist dieses Verfahren praktikabel und erzielt die angestrebten Resultate, jedoch ist es unter Zeiteffizienzgesichtspunkten inadäquat, da sich Einfüge- und Kommunikationsvorgänge zahlenmäßig entsprechen.

Aus diesem Grunde bietet die Schnittstelle `Statement` die Möglichkeit zur Bündelung einzelner SQL-Aufrufe in einem sog. *Batch* an.

Beispiel 52 zeigt die entsprechende Umgestaltung des vorangegangenen Beispiels.

Beispiel 52: Einfügen von Werten mittels eines Batches

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCInsert2 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
```

```
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('John', 'B', 'Smith',
123456789, '1965-01-09', '731 Fondren, Houston, TX', 'M', 30000, 333445555, 5);");
(49)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Franklin', 'T',
'Wong', 333445555, '1955-12-08', '638 Voss, Houston, TX', 'M', 40000, 888665555, 5);");
(50)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Alicia', 'J',
'Zelaya', 999887777, '1968-07-19', '3321 Castle, Spring, TX', 'F', 25000, 987654321, 4);");
(51)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Jennifer', 'S',
'Wallace', 987654321, '1941-06-20', '291 Berry, Bellaire, TX', 'F', 43000, 888665555,
4);");
(52)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ramesh', 'K',
'Narayan', 666884444, '1962-09-15', '975 Fire Oak, Humble, TX', 'M', 38000, 333445555,
5);");
(53)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Joyce', 'A',
'English', 453453453, '1972-07-31', '5631 Rice, Houston, TX', 'F', 25000, 333445555, 5);");
(54)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ahmad', 'V', 'Jabbar',
987987987, '1969-03-29', '980 Dallas, Houston, TX', 'M', 25000, 987654321, 4);");
(55)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('James', 'E', 'Borg',
888665555, '1937-11-10', '450 Stone, Houston, TX', 'M', 55000, null, 1);");
(56)                    int[] insertCounts = stmt.executeBatch();
(57)            } catch (SQLException e3) {
(58)                    System.err.println("Error inserting values into table EMPLOYEE");
(59)                    Throwable t = e3;
(60)                    while (t != null) {
(61)                    System.err.println("Type: " + t.getClass().getName());
(62)                    System.err.println("Message: " + t.getMessage());
(63)                    System.err.println("-----");
(64)                    t = t.getCause();
(65)                    }
(66)            }
(67)    }
(68)}
```

Download des Beispiels

Statt der Einzelübergabe der SQL `INSERT`-Anweisungen werden diese nun (in Zeile 36-43) in in einem Batch gesammelt. Hierzu werden die SQL-Zeichenketten durch den Aufruf `addBatch` innerhalb des `Statement`-Objekts abgelegt und durch Aufruf der Methode `executeBatch` gesammelt an das DBMS übergeben.
Statt der Einzelresultate wird durch diese Aufrufvariante ein Array geliefert, das die Einzelrückgabewerte der als Batch übergebenen Aufrufe versammelt.

Dies verdeutlicht nochmals das nachfolgende Beispiel. In ihm wird zunächst mittels `ALTER TABLE` eine neue Tabellenspalte zur Aufnahme des Wochentages der Geburt erstellt und anschließend durch SQL `UPDATE`-Anweisungen die benötigten Daten aus dem vorhandenen Geburtsdatum ermittelt.
Auch dieses Beispiel bedient sich zur Performancebeschleunigung der Möglichkeiten des Batchaufrufes.

Beispiel 53: Aktualisieren von Tabellendefinitionen und Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCUpdate1 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
(49)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Sunday' WHERE DAYOFWEEK
(BDATE)=1;");
(50)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Monday' WHERE DAYOFWEEK
(BDATE)=2;");
(51)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Tuesday' WHERE DAYOFWEEK
(BDATE)=3;");
(52)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Wednesday' WHERE DAYOFWEEK
(BDATE)=4;");
(53)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Thursday' WHERE DAYOFWEEK
(BDATE)=5;");
(54)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Friday' WHERE DAYOFWEEK
(BDATE)=6;");
(55)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Saturday' WHERE DAYOFWEEK
(BDATE)=7;");
(56)                    int[] result = stmt.executeBatch();
(57)                    for (int i=0; i<result.length;i++){
(58)                            System.out.println("Statement No "+i+" changed "+result[i]
+" rows");
(59)                    }
(60)            } catch (SQLException e3) {
(61)                    System.err.println("Error inserting values into table EMPLOYEE");
(62)                    Throwable t = e3;
(63)                    while (t != null) {
(64)                    System.err.println("Type: " + t.getClass().getName());
(65)                    System.err.println("Message: " + t.getMessage());
(66)                    System.err.println("-----");
(67)                    t = t.getCause();
(68)                    }
```

```
(69)            }
(70)    }
(71)}
```

Download des Beispiels
Download der Ergebnisdatei

Die Ausführung liefert als Resultat:

```
Statement No 0 changed 8 rows
Statement No 1 changed 0 rows
Statement No 2 changed 1 rows
Statement No 3 changed 0 rows
Statement No 4 changed 1 rows
Statement No 5 changed 1 rows
Statement No 6 changed 2 rows
Statement No 7 changed 3 rows
```

So werden durch den ALTER TABLE-Aufruf (Indexnummer 0) alle acht Tupel der Tabelle modifiziert, während die nachfolgenden Aufrufe nur Teilmengen davon verändern.

Die nähere Betrachtung der Zeilen 37-43 des Quellcodes von Beispiel 53 zeigt sich, daß diese im Kern denselben Vorgang ausführen, nur jeweils mit variierenden Parametern.
Zur Behandlung von Fällen dieser Problemstellung definiert die JDBC-API die Schnittstelle `PreparedStatement` als Spezialisierung von `Statement`.

Diese Schnittstelle gestattet es, Anweisungen, die später an die Datenbank übermittelt werden sollen, mit Platzhaltern zu versehen und diese vor der Übermittlung mit Werten zu befüllen.
Beispiel 54 zeigt die entprechende Modifikation des vorangegangenen Beispiels.

Beispiel 54: Aktualisieren von Tabellendefinitionen und Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.PreparedStatement;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCUpdate2 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            PreparedStatement pstmt = null;
(36)            try {
(37)                    stmt = (Statement) con.createStatement();
(38)                    pstmt = (PreparedStatement) con.prepareStatement("UPDATE EMPLOYEE
SET BDAY=? WHERE DAYOFWEEK(BDATE)=?;");
(39)
(40)            } catch (SQLException e2) {
(41)                    System.err.println("Error creating SQL-Statement");
(42)                    Throwable t = e2;
```

```
(43)                    while (t != null) {
(44)                    System.err.println("Type: " + t.getClass().getName());
(45)                    System.err.println("Message: " + t.getMessage());
(46)                    System.err.println("-----");
(47)                    t = t.getCause();
(48)                    }
(49)            }
(50)
(51)            try {
(52)                    String[] days=
{"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"} ;
(53)                    stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
(54)                    for (int i=1; i<8;i++){
(55)                            pstmt.setString(1,days[i-1]);
(56)                            pstmt.setInt(2,i);
(57)                            pstmt.addBatch();
(58)                    }
(59)                    int[] result = stmt.executeBatch();
(60)                    for (int i=0; i<result.length;i++){
(61)                            System.out.println("Statement No "+i+" changed "+result[i]
+" rows");
(62)                    }
(63)            } catch (SQLException e3) {
(64)                    System.err.println("Error inserting values into table EMPLOYEE");
(65)                    Throwable t = e3;
(66)                    while (t != null) {
(67)                    System.err.println("Type: " + t.getClass().getName());
(68)                    System.err.println("Message: " + t.getMessage());
(69)                    System.err.println("-----");
(70)                    t = t.getCause();
(71)                    }
(72)            }
(73)    }
(74)}
```

Download des Beispiels
Download der Ergebnisdatei

Im Beispiel wird neben dem Objekt des Typs `Statement` zusätzlich eines des Typs `PreparedStatement` erzeugt (Zeile 32).
Die dem Konstruktor übergebene Anweisung enthält als Sonderzeichen zur Markierung der Platzhalter das Fragezeichen (?).
Die Wochentage werde in Zeile 40, des vereinfachten Zugriffs wegen, als Array definiert.
In den Zeilen 42 mit 46 werden die benötigten SQL-`UPDATE`-Anweisungen dynamisch durch Einsetzen der geeigneten Werte in den vorpräparierten Änderungsausruck erzeugt und einem eigenen Batch zugeordnet.
Der Einsetzungsvorgang der benötigten Werte geschieht durch die Methoden `setString` für zeichenkettenartige bzw. `setInt` für den ganzzahlige Parameter. Den Methoden wird jeweils die Position des Parameters, gezählt ab 1 sowie die zu wählende Wertbelegung übermittelt.
Zur Ausführung müssen beide Batches getrennt angefordert werden.

Tabellenwertige Zugriffe

Die in der Praxis quantitativ bedeutendste Klasse von Datenbankzugriffen dürfte zweifellos auf die lesende Ermittlung von bestehenden Daten darstellen, kurzum alle Spielarten der SQL `SELECT`-Anweisung.

Für Anfragen an die Datenbank steht prinzipiell der gesamte durch das DBMS unterstützte SQL-Umfang zur Verfügung.

Anfragen werden im Gegensatz zu den bisher betrachteten lesenden Zugriffen nicht als primivwerte Methoden realisiert, sondern liefern als Resultat immer eine Tabelle zurück.
Diese wird durch den API-Typ `ResultSet` dargestellt.
Zusätzlich werden Anfragen durch die Methode executeQuery ausgeführt.

Das Beispiel 55 zeigt die generische Extraktion von DB-Daten und den Zugriff auf Metadaten.
Die aus der Datenbank gelesenen Ergebnistupel werden im durch `rs` benannten `ResultSet` abgelegt (Zeile 39). Die Resultatmenge wird mithilfe eines *Cursors* (Datensatzzeiger) traversiert. Hierzu wird der initial auf eine Ausgangsstellung vor dem ersten empfangenen Tupel positionierte Cursor durch Aufruf der Methode `next` solange weitergerückt, bis der letzte Datensatz verarbeitet wurde.

Der Aufruf der Methode`getMetaData` liefert deskriptive Metadaten wie Spaltenzahl sowie deren Bezeichner und Typen für die erstellte Resultattupelmenge.
In Zeile 43 werden diese Metadaten verwendet um die Spaltennamen der extrahierten Attribute

anzuzeigen.
Zeile 47-52 liest die einzelnen Werte jedes Tupels mittels getObject aus und stellt sie am Bildschirm dar.

Beispiel 55: Auslesen von Daten und Metadaten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.ResultSetMetaData;
(4)import java.sql.SQLException;
(5)
(6)import com.mysql.jdbc.Connection;
(7)import com.mysql.jdbc.Statement;
(8)
(9)public class JDBCSelect1 {
(10)    public static void main(String[] args) {
(11)            try {
(12)                    Class.forName("com.mysql.jdbc.Driver");
(13)            } catch (ClassNotFoundException e) {
(14)                    System.err.println("Driver class not found");
(15)                    e.printStackTrace();
(16)            }
(17)            Connection con = null;
(18)
(19)            try {
(20)                    con =
(21)                            (Connection) DriverManager.getConnection(
(22)                                    "jdbc:mysql://localhost/jdbctest/",
(23)                                    "mario",
(24)                                    "thePassword");
(25)            } catch (SQLException e1) {
(26)                    System.err.println("Error establishing database connection");
(27)                    Throwable t = e1;
(28)                    while (t != null) {
(29)                    System.err.println("Type: " + t.getClass().getName());
(30)                    System.err.println("Message: " + t.getMessage());
(31)                    System.err.println("-----");
(32)                    t = t.getCause();
(33)                    }
(34)            }
(35)
(36)            Statement stmt = null;
(37)            try {
(38)                    stmt = (Statement) con.createStatement();
(39)            } catch (SQLException e2) {
(40)                    System.err.println("Error creating SQL-Statement");
(41)                    Throwable t = e2;
(42)                    while (t != null) {
(43)                    System.err.println("Type: " + t.getClass().getName());
(44)                    System.err.println("Message: " + t.getMessage());
(45)                    System.err.println("-----");
(46)                    t = t.getCause();
(47)                    }
(48)            }
(49)
(50)            try {
(51)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(52)                    ResultSetMetaData rsmd = rs.getMetaData();
(53)                    int noColumns = rsmd.getColumnCount();
(54)                    for (int i = 1; i < noColumns; i++) {
(55)                            System.out.print(rsmd.getColumnLabel(i) + "\t");
(56)                    }
(57)                    System.out.println();
(58)
(59)                    while (rs.isLast() == false) {
(60)                            rs.next();
(61)                            for (int i = 1; i < noColumns; i++) {
(62)                                    System.out.print( rs.getObject(i)+"\t" );
(63)                            }
(64)                            System.out.println();
(65)                    }
(66)
(67)            } catch (SQLException e3) {
(68)                    System.err.println("Error selecting values from table EMPLOYEE");
(69)                    Throwable t = e3;
(70)                    while (t != null) {
(71)                    System.err.println("Type: " + t.getClass().getName());
(72)                    System.err.println("Message: " + t.getMessage());
```

```
(73)                    System.err.println("-----");
(74)                    t = t.getCause();
(75)                    }
(76)            }
(77)    }
(78)}
```

Download des Beispiels
Download der Ergebnisdatei

Neben im Beispiel 55 gezeigten Verarbeitung in exakter der Ablagereihenfolge der Datenbank kann auch durch Definition eines Cursors die Traversierung in inverser Ablagerichtung erreicht werden.
Das nachfolgende Beispiel illustriert das entsprechende Vorgehen durch anfängliche Positionierung des Cursors ans Ende der empfangenen Daten (d.h. nach dem letzten Datensatz) und anschließendes schrittweises Rückpositionieren durch Aufruf der Methode `previous`.

Beispiel 56: Auslesen von Daten in invertierter Reihenfolge

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect5 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt =
(37)                            (Statement) con.createStatement();
(38)            } catch (SQLException e2) {
(39)                    System.err.println("Error creating SQL-Statement");
(40)                    Throwable t = e2;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            try {
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    rs.afterLast();
(52)                    while (rs.previous()){
(53)                            System.out.println(rs.getString("FNAME"));
(54)                    }
(55)            } catch (SQLException e3) {
(56)                    System.err.println("Error selecting values from table EMPLOYEE");
(57)                    Throwable t = e3;
(58)                    while (t != null) {
```

```
(59)                    System.err.println("Type: " + t.getClass().getName());
(60)                    System.err.println("Message: " + t.getMessage());
(61)                    System.err.println("-----");
(62)                    t = t.getCause();
(63)                    }
(64)            }
(65)    }
(66)}
```

Download des Beispiels
Download der Ergebnisdatei

Ferner kann der Cursor wahlfrei auf eine beliebige Position der Ergebnisrelation gesetzt werden. Das nachfolgende Beispiel zeigt dies. Ferner illustriert es das Vorgehen zur Größenermittlung des resultierenden `ResultSets` durch das Aufrufpaar `last` und `getRow`, welches zunächst den Cursor auf den letzten aus der Datenbank extrahierten Datensatz positioniert und anschließend dessen Nummer liefert.

Beispiel 57: Auslesen von Daten in wahlfreier Reihenfolge

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect6 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt = (Statement) con.createStatement();
(37)            } catch (SQLException e2) {
(38)                    System.err.println("Error creating SQL-Statement");
(39)                    Throwable t = e2;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)
(48)            try {
(49)                    int position = 0;
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    rs.last();
(52)                    int size = rs.getRow();
(53)                    for (int i = 0; i < size; i++) {
(54)                            position = (position + 3) % size;
(55)                            rs.absolute(position + 1);
(56)                            System.out.println(
```

```
(57)                                    "position=" + (position + 1) + ": " + rs.getString
("FNAME"));
(58)                    }
(59)            } catch (SQLException e3) {
(60)                    System.err.println("Error selecting values from table EMPLOYEE");
(61)                    Throwable t = e3;
(62)                    while (t != null) {
(63)                    System.err.println("Type: " + t.getClass().getName());
(64)                    System.err.println("Message: " + t.getMessage());
(65)                    System.err.println("-----");
(66)                    t = t.getCause();
(67)                    }
(68)            }
(69)    }
(70)}
```

Download des Beispiels
Download der Ergebnisdatei

Wird der benötigte `ResultSet` geeignet (d.h. mit den Parameter `CONCUR_UPDATABLE`) (siehe Zeile 49) initialisiert, so können Änderungen, die im Hauptspeicher durch die JDBC-API durchgeführt werden, in die Datenbank persistiert werden.
Beispiel 58 zeigt dies exemplarisch für den Einfügevorgang eines neuen Tupels.

Die Voraussetzungen für Einfüge- und Aktualisierungsvorgänge entstprechen denen von *updatable views*, d.h. die Daten dürfen nur aus genau einer Tabelle entnommen sein und müssen den Primärschlüssel enthalten.

Beispiel 58: Auslesen und Einfügen von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.ResultSetMetaData;
(4)import java.sql.SQLException;
(5)import java.sql.Statement;
(6)
(7)import com.mysql.jdbc.Connection;
(8)
(9)public class JDBCSelect2 {
(10)    private static void printResultSet(ResultSet rs) throws SQLException {
(11)            ResultSetMetaData rsmd = rs.getMetaData();
(12)            int noColumns = rsmd.getColumnCount();
(13)            for (int i = 1; i < noColumns; i++) {
(14)                    System.out.print(rsmd.getColumnLabel(i) + "\t");
(15)            }
(16)            System.out.println();
(17)
(18)            while (rs.isLast() == false) {
(19)                    rs.next();
(20)                    for (int i = 1; i < noColumns; i++) {
(21)                            System.out.print( rs.getObject(i)+"\t" );
(22)                    }
(23)                    System.out.println();
(24)            }
(25)
(26)    }
(27)    public static void main(String[] args) {
(28)            try {
(29)                    Class.forName("com.mysql.jdbc.Driver");
(30)            } catch (ClassNotFoundException e) {
(31)                    System.err.println("Driver class not found");
(32)                    e.printStackTrace();
(33)            }
(34)            Connection con = null;
(35)
(36)            try {
(37)                    con =
(38)                            (Connection) DriverManager.getConnection(
(39)                                    "jdbc:mysql://localhost/jdbctest/",
(40)                                    "mario",
(41)                                    "thePassword");
(42)            } catch (SQLException e1) {
(43)                    System.err.println("Error establishing database connection");
(44)                    Throwable t = e1;
(45)                    while (t != null) {
```

```
(46)                    System.err.println("Type: " + t.getClass().getName());
(47)                    System.err.println("Message: " + t.getMessage());
(48)                    System.err.println("-----");
(49)                    t = t.getCause();
(50)                    }
(51)            }
(52)
(53)            Statement stmt = null;
(54)            try {
(55)                    stmt = (Statement) con.createStatement(ResultSet.
TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
(56)            } catch (SQLException e2) {
(57)                    System.err.println("Error creating SQL-Statement");
(58)                    Throwable t = e2;
(59)                    while (t != null) {
(60)                    System.err.println("Type: " + t.getClass().getName());
(61)                    System.err.println("Message: " + t.getMessage());
(62)                    System.err.println("-----");
(63)                    t = t.getCause();
(64)                    }
(65)            }
(66)
(67)            try {
(68)                    ResultSet uprs = (ResultSet) stmt.executeQuery("SELECT * FROM
EMPLOYEE;");
(69)                    printResultSet(uprs);
(70)                    uprs.moveToInsertRow();
(71)                    uprs.updateString("FNAME","Mario");
(72)                    uprs.updateString("LNAME","Jeckle");
(73)                    uprs.updateInt("SSN",111111111);
(74)                    uprs.insertRow();
(75)                    uprs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(76)                    printResultSet(uprs);
(77)            } catch (SQLException e3) {
(78)                    System.err.println("Error selecting values from table EMPLOYEE");
(79)                    Throwable t = e3;
(80)                    while (t != null) {
(81)                    System.err.println("Type: " + t.getClass().getName());
(82)                    System.err.println("Message: " + t.getMessage());
(83)                    System.err.println("-----");
(84)                    t = t.getCause();
(85)                    }
(86)            }
(87)    }
(88)}
```

Download des Beispiels
Download der Ergebnisdatei

Auf dieselbe Weise können auch Tupel einer Relation verändert werden. Hierzu stehen eine Reihe von updateXXX-Methoden zur Verfügung, wobei XXX für den Typ des zu aktualisierenden Attributs steht.
Nach durchgeführter Modifikation der hauptspeicherresidenten Werte werden diese durch updateRow in die Datenbank rückgeschrieben.
Beispiel 59 zeigt dies:

Beispiel 59: Modifizieren von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Statement;
(5)
(6)import com.mysql.jdbc.Connection;
(7)
(8)public class JDBCSelect3 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException e) {
(13)                    System.err.println("Driver class not found");
(14)                    e.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
```

```
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException e1) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = e1;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt =
(38)                            (Statement) con.createStatement(
(39)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(40)                                    ResultSet.CONCUR_UPDATABLE);
(41)            } catch (SQLException e2) {
(42)                    System.err.println("Error creating SQL-Statement");
(43)                    Throwable t = e2;
(44)                    while (t != null) {
(45)                    System.err.println("Type: " + t.getClass().getName());
(46)                    System.err.println("Message: " + t.getMessage());
(47)                    System.err.println("-----");
(48)                    t = t.getCause();
(49)                    }
(50)            }
(51)
(52)            try {
(53)                    ResultSet uprs =
(54)                            (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(55)                    int namePos = uprs.findColumn("LNAME");
(56)
(57)                    while (uprs.isLast() == false) {
(58)                            uprs.next();
(59)                            if (uprs.getString(namePos).compareTo("Wallace") == 0) {
(60)                                    uprs.updateString(namePos, "Doe");
(61)                                    uprs.updateRow();
(62)                            }
(63)                    }
(64)
(65)            } catch (SQLException e3) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = e3;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            }
(75)    }
(76)}
```

Download des Beispiels

Analog vollzieht sich der Löschvorgang mittels `deleteRow`:

Beispiel 60: Löschen von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Statement;
(5)
(6)import com.mysql.jdbc.Connection;
(7)
(8)public class JDBCSelect4 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException e) {
(13)                    System.err.println("Driver class not found");
(14)                    e.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException e1) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = e1;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt =
(38)                            (Statement) con.createStatement(
(39)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(40)                                    ResultSet.CONCUR_UPDATABLE);
(41)            } catch (SQLException e2) {
(42)                    System.err.println("Error creating SQL-Statement");
(43)                    Throwable t = e2;
(44)                    while (t != null) {
(45)                    System.err.println("Type: " + t.getClass().getName());
(46)                    System.err.println("Message: " + t.getMessage());
(47)                    System.err.println("-----");
(48)                    t = t.getCause();
(49)                    }
(50)            }
(51)
(52)            try {
(53)                    ResultSet uprs =
(54)                            (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(55)                    int namePos = uprs.findColumn("LNAME");
(56)
(57)                    while (uprs.isLast() == false) {
(58)                            uprs.next();
(59)                            if (uprs.getString(namePos).compareTo("Smith") == 0) {
(60)                                    uprs.deleteRow();
(61)                            }
(62)                    }
(63)
(64)            } catch (SQLException e3) {
(65)                    System.err.println("Error selecting values from table EMPLOYEE");
(66)                    Throwable t = e3;
(67)                    while (t != null) {
(68)                    System.err.println("Type: " + t.getClass().getName());
(69)                    System.err.println("Message: " + t.getMessage());
(70)                    System.err.println("-----");
(71)                    t = t.getCause();
(72)                    }
(73)            }
(74)    }
(75)}
```

Download des Beispiels

Die bisher betrachteten Varianten extrahieren Daten aus der Datenbank im Stile einer Momentaufnahme (*snapshot*) zum Zeitpunkt der Anfrage. Die einmal angefragten Inhalte können sich jedoch noch zur Laufzeit der zugreifenden JDBC-Applikation datenbankseitig ändern, wenn sie durch eine andere Applikation neu geschrieben werden. Zur Gewährleistung der Konsistenz des extrahierten Snapshots mit den tatsächlichen Datenbankinhalten steht die Operation `rowUpdated` zur Verfügung. Sie ermittelt ob der im Hauptspeicher befindliche Wert mit dem aktuellen Datenbankinhalt übereinstimmt, d.h. ob der DB-Inhalt aktualisiert wurde.
Beispiel 61 zeigt ein Umsetzungsbeispiel.

Beispiel 61: Test auf geänderte Daten

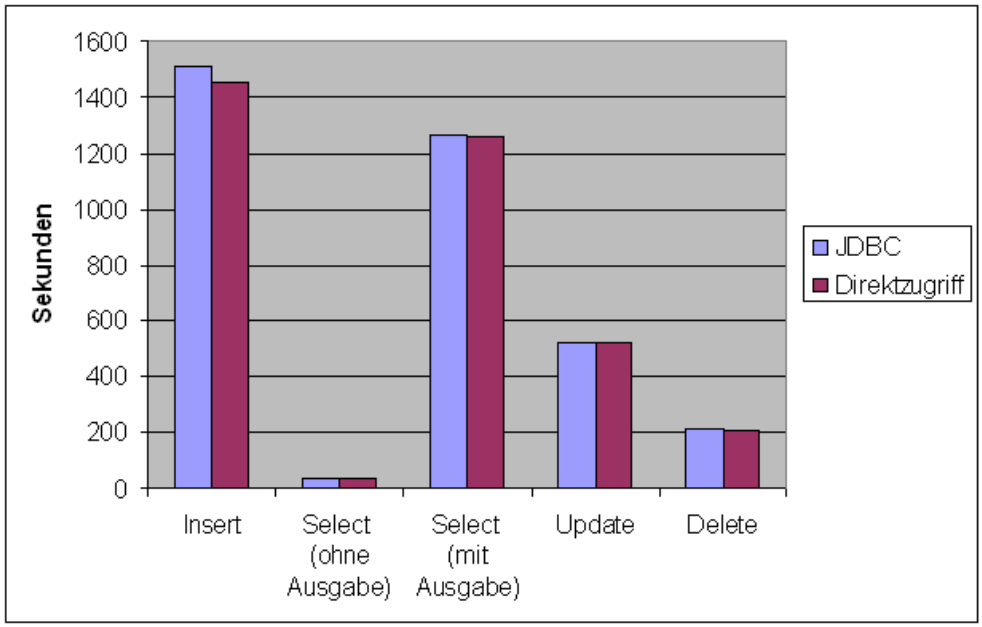
```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect7 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt =
(37)                            (Statement) con.createStatement(
(38)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(39)                                    ResultSet.CONCUR_UPDATABLE);
(40)            } catch (SQLException e) {
(41)                    System.err.println("Error creating SQL-Statement");
(42)                    Throwable t = e;
(43)                    while (t != null) {
(44)                    System.err.println("Type: " + t.getClass().getName());
(45)                    System.err.println("Message: " + t.getMessage());
(46)                    System.err.println("-----");
(47)                    t = t.getCause();
(48)                    }
(49)            }
(50)
(51)            try {
(52)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(53)                    rs.absolute(5);
(54)                    System.out.println(rs.getString("FNAME"));
(55)
(56)                    System.out.println("sleeping ...");
(57)                    Thread.sleep(6000);
(58)                    System.out.println("awake ...");
(59)
(60)                    if (rs.rowUpdated() == true) {
(61)                            rs.refreshRow();
(62)                            System.out.println(rs.getString("FNAME"));
```

```
(63)                    }
(64)
(65)            } catch (SQLException e) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = e;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            } catch (InterruptedException ie) {
(75)                    ie.printStackTrace();
(76)            }
(77)    }
(78)}
```

Download des Beispiels

Performancebetrachtungen

Abbildung 5: *JDBC-Geschwindigkeitsvergleich*



(click on image to enlarge!)

Die Abbildung zeigt die Ergebnisse einiger Geschwindigkeitsmessungen als Vergleich zwischen dem Zugriff auf eine MySQL-Datenbank unter Nutzung der Textschnittstelle und der Abwicklung derselben Zugriffe mittels JDBC.

Zur Messung wurde eine nicht-indexierte Datenbank mit 10^7 Einträgen verwendet die aus der Relation `tab` bestand. Deren Tupel wurden aus Paaren von 36-Byte großen UUIDs gemäß dem Spezifikationsentwurf der IETF gebildet.

Zur Zeitmessung wurden folgende Einzeloperationen betrachtet:

- **Insert**: `INSERT INTO tab VALUES(...)`
  Die Werte wurden unter Deaktivierung des Autocommit sequentiell eingefügt.
- **Select (ohne Ausgabe)**: `SELECT COUNT(*) FROM tab`
  Die ermittelte Gesamtzahl wurde nicht am Bildschirm ausgegeben.
- **Select (mit Ausgabe)**: `SELECT * FROM tab`
  Die Resultate wurden durch den Standarclient bzw. eine selbsterstellte (textmode) Java-Implementierung ausgegeben.
- **Update**: `UPDATE tab SET UUID1="X" WHERE UUID1<>"X"`
  Durch die Initialisierung der Werte mit UUID-Einträgen wird sichergestellt, daß alle Tupel aktualisiert werden, da sie in keinem Fall den Wert `X` enthalten.
- **Delete**: `DELETE FROM tab WHERE UUID2<>"X"`
  Durch die Initialisierung der Werte mit UUID-Einträgen wird sichergestellt, daß alle Tupel aktualisiert werden, da sie in keinem Fall den Wert `X` enthalten.

Insgesamt zeigt sich ein ausgewogenes Bild, in welchem der JDBC-Zugriff lediglich bei datenintensiven Zugriffen (große Mengen schreibender Zugriffe bei `INSERT` bzw. große Mengen lesender Operationen bei `SELECT`) im Bereich von fünf Prozent zurückliegt.

Diese enge Vergleichbarkeit der beiden Zugriffsmodi rührt von den Realisierung des eingesetzten JDBC-Treibers her; insbesondere von der Handhabung der physischen Datenbankverbindung auf Ebene des Netzwerkprotokolls.

SQL3-Datentypen

Die JDBC-API unterstützt mit Zugriffsmethoden auf die Datentypen `BLOB`, `CLOB`, `ARRAY`, `Object` und `Ref` bereits eine Untermenge des SQL:1999-Standards. So können, vorausgesetzt das durch JDBC angesprochene DBMS unterstützt dies, große unstrukturierte Binär- oder Textdaten sowie einfache verschachtelte Tabellen, mithin NF^2-Strukturen verwaltet werden.

Beispiel 62 zeigt den Zugriff auf ein als eingebettete Tabelle realisiertes mengenwertiges Attribut. Die Beispieldatenbank wurde hierfür wie folgt modifiziert:

```
alter table EMPLOYEE ADD CAR SET('53M91','521R4', 'LLO415', 'XNU457');
update EMPLOYEE set CAR='XNU457' where SSN=123456789;
update EMPLOYEE set CAR='XNU457,521R4'  where SSN="999887777";
```

Beispiel 62: Zugriff auf ein mengenwertiges Attribut

```
(1)import java.sql.Array;
(2)import java.sql.DriverManager;
(3)import java.sql.ResultSet;
(4)import java.sql.SQLException;
(5)import com.mysql.jdbc.Connection;
(6)import com.mysql.jdbc.Statement;
(7)
(8)public class JDBCSelect8 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException cnfe) {
(13)                    System.err.println("Driver class not found");
(14)                    cnfe.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException sqle) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = sqle;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt = (Statement) con.createStatement();
(38)            } catch (SQLException sqle) {
(39)                    System.err.println("Error creating SQL-Statement");
(40)                    Throwable t = sqle;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            try {
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    while (!rs.isLast()) {
(52)                            rs.first();
(53)                            System.out.print(rs.getString("FNAME") + "\t");
(54)                            Array cars = rs.getArray("CAR");
```

```
(55)                            ResultSet carsRS = cars.getResultSet();
(56)                            System.out.print("(");
(57)                            while (!carsRS.isLast()) {
(58)                                    rs.first();
(59)                                    System.out.print(carsRS.getString("CAR"));
(60)                                    carsRS.next();
(61)                            }
(62)                            System.out.println(")");
(63)                            rs.next();
(64)                    }
(65)            } catch (SQLException sqle) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = sqle;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            }
(75)    }
(76)}
```

Download des Beispiels

Das Beispiel unterstreicht die Rolle der mengenwertigen Attribute als eingebettete Tabellen. So erfolgt der Zugriff auf die Einzelwerte des Attributs `CAR` identisch zur Ermittlung der Resultatmenge der SQL-Anfrage mittels `getResultSet`. Auch die Traversierung der einzelnen `CAR`-Elemente erfolgt äquivalent.

Die Aufnahme der *large objects* in ihrer Ausprägungsform als *Character Large Objects* (CLOB) oder *Binary Large Objects* (BLOB) stellen eine der zentralen Erweiterungen des SQL:1999-Standards gegenüber seinen Vorgängern dar.
Zwar ist die Ablage großer unstrukturierter Datenobjekte in relationalen Datenbanken konzeptionell durchaus diskussionswert, jedoch in der Praxis oftmals, trotz der teilweise erheblichen Geschwindigkeitseinbußen im Zugriff (so benötigt die Ausführung der Beispielapplikation mit einem 10^6 Byte großen Datenstrom 1,1 Sekunden, während dieselbe Operation dateisystembasiert in 0,1 Sekunde abläuft), gewünscht.
Beispiel 63 zeigt die notwendigen Schritte zur Ablage und erneuten Auslese eines aus einer Datei gewonnen Binärdatenstroms in der Datenbank.
Die Beispieldatenbank wurde hierfür um ein Attribut zur Aufnahme binärer Daten erweitert:
`ALTER TABLE EMPLOYEE ADD binData blob;`

Beispiel 63: Verarbeitung unstrukturierter Binärdaten

```
(1)import java.io.File;
(2)import java.io.FileInputStream;
(3)import java.io.FileOutputStream;
(4)import java.io.IOException;
(5)import java.sql.DriverManager;
(6)import java.sql.PreparedStatement;
(7)import java.sql.ResultSet;
(8)import java.sql.SQLException;
(9)
(10)import com.mysql.jdbc.Connection;
(11)import com.mysql.jdbc.Statement;
(12)
(13)public class JDBCSelect9 {
(14)    public static void main(String[] args) {
(15)            try {
(16)                    Class.forName("com.mysql.jdbc.Driver");
(17)            } catch (ClassNotFoundException cnfe) {
(18)                    System.err.println("Driver class not found");
(19)                    cnfe.printStackTrace();
(20)            }
(21)            Connection con = null;
(22)
(23)            try {
(24)                    con =
(25)                            (Connection) DriverManager.getConnection(
(26)                                    "jdbc:mysql://localhost/jdbctest/",
(27)                                    "mario",
(28)                                    "thePassword");
(29)            } catch (SQLException sqle) {
(30)                    System.err.println("Error establishing database connection");
(31)                    Throwable t = sqle;
```

```
(32)                    while (t != null) {
(33)                    System.err.println("Type: " + t.getClass().getName());
(34)                    System.err.println("Message: " + t.getMessage());
(35)                    System.err.println("-----");
(36)                    t = t.getCause();
(37)                    }
(38)            }
(39)
(40)            try {
(41)                    File file = new File(args[0]);
(42)                    FileInputStream fis = new FileInputStream(args[0]);
(43)                    PreparedStatement pstmt =
(44)                            con.prepareStatement(
(45)                                    "UPDATE EMPLOYEE SET binData =? WHERE
SSN=123456789");
(46)                    pstmt.setBinaryStream(1, fis, (int) file.length());
(47)                    pstmt.executeUpdate();
(48)                    fis.close();
(49)
(50)                    //read it back from the database
(51)                    Statement stmt = (Statement) con.createStatement();
(52)                    ResultSet rs =
(53)                            stmt.executeQuery(
(54)                                    "SELECT binData FROM EMPLOYEE WHERE
SSN='123456789';");
(55)
(56)                    FileOutputStream fos = new FileOutputStream(args[1]);
(57)                    if (rs.next())
(58)                            fos.write(rs.getBytes(1));
(59)                    fos.close();
(60)
(61)            } catch (SQLException sqle) {
(62)                    System.err.println("Error selecting values from table EMPLOYEE");
(63)                    Throwable t = sqle;
(64)                    while (t != null) {
(65)                    System.err.println("Type: " + t.getClass().getName());
(66)                    System.err.println("Message: " + t.getMessage());
(67)                    System.err.println("-----");
(68)                    t = t.getCause();
(69)                    }
(70)            } catch (IOException ioe) {
(71)                    ioe.printStackTrace();
(72)            }
(73)    }
(74)}
```

Download des Beispiels

Die Binärdaten können naturgemäß nicht direkt in die SQL-UPDATE-Anweisung eingebunden werden, sie werden daher einer mittels prepareStatement vorerzeugten Anweisung durch Aufruf der Methode setBinaryStream übergeben.

Transaktionssteuerung

Zur Steuerung des transaktionalen Verhaltens einer JDBC-Anfrage bietet die Klasse Connection verschiedene Methoden an:

- Abfrage der aktuellen Isolationsstufe: getIsolationLevel.
  Hierbei werden fünf Stufen unterschieden:
  - TRANSACTION_NONE: Keinerlei Transaktionsunterstützung
  - TRANSACTION_READ_UNCOMMITTED: Auch nicht durch commit freigegebene Daten werden gelesen. Es können daher *dirty reads*, Nicht-wiederholbare- und Phantomlesevorgänge auftreten.
  - TRANSACTION_READ_COMMITTED: Nur durch commit freigegebene Daten werden gelesen. nichtwiederholbare- und Phantomlesevorgänge können jedoch auftreten.
  - TRANSACTION_REPEATABLE_READ: Innerhalb einer Transaktion können die verarbeiteten Daten nicht durch eine andere Transaktion verändert werden. Das Auftreten von *dirty reads* und nichtwiederholbaren Lesevorgängen ist daher ausgeschlossen, Phantomlesevorgänge sind jedoch weiterhin möglich.
  - TRANSACTION_SERIALIZABLE: Strikte Isolation aller Transaktionen, auf dieser Stufe sind auch Phantomlesevorgänge ausgeschlossen.

  Jede Stufe geht zwar mit einer gesteigerten Qualität der durch eine Transaktion verarbeiteten Daten einher, jedoch senkt gleichzeitig eine strikterere Isolationsstufe die Anzahl der gleichzeitigen Zugriffe auf die Datenbank und damit die Gesamtsystemperformance.

- Setzen der Isolationsstufe: `setTransactionIsolation`
- (De-)Aktivierung der automatischen Freigabe: `setAutoCommit`. Die Übergabe von `true` bewirkt die Aktivierung des Modus bei dem jede Einzelanweisung sofort persistent übernommen und für andere Transaktionen sichtbar wird.
  Standardmäßig ist diese Option aktiviert. Ihr aktueller Zustand kann per `getAutoCommit` ermittelt werden.
- Freigabe von Änderungen: `commit`.
- Rücknahme von Änderungen: `rollback`.
- Rücknahme von Änderungen bis zu definiertem Sicherungspunkt: `rollback(Savepoint s)`.
- Setzen eines Sicherungspunktes: `setSavepoint`.

Beispiel 1 zeigt die Nutzung des Transaktionskonzepts.
Zunächst wird die aktuelle Isolationsstufe ermittelt und geprüft ob das angesprochene DBMS die höchste durch JDBC vorhergesehene Isolationsstufe unterstützt.
Nach Abschaltung der automatischen Änderungsübernahme (`setAutoCommit(false)`) werden zunächst zwei Tupel in die Tabelle `EMPLOYEE` eingefügt, die jedoch nur innerhalb der laufenden Transaktion sichtbar werden, für alle anderen Transaktionen innerhalb des DBMS bleiben die neuen Werte (zunächst) unsichtbar.
Eine angenommene Fehlersituation führt zum Rücksetzen der Transaktion durch (`rollback`).
Nach Abschluß des Programms wurden zwar die beiden ersten Werte lokal in die Datenbank übernommen, aber noch innerhalb der laufenden Transaktion wieder daraus entfernt, weshalb sie zu keinem Zeitpunkt für andere Datenbankbenutzer sichtbar waren.

Beispiel 64: Transaktionsverarbeitung

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)
(5)import com.mysql.jdbc.Connection;
(6)import com.mysql.jdbc.Statement;
(7)
(8)public class JDBCTransact1 {
(9)     private static void printContent(Statement stmt) throws SQLException {
(10)            ResultSet rs =
(11)                    stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
(12)            while (!rs.isLast()) {
(13)                    rs.next();
(14)                    System.out.println(
(15)                            rs.getString("LNAME")
(16)                                    + "\t"
(17)                                    + rs.getString("MINIT")
(18)                                    + "\t"
(19)                                    + rs.getString("LNAME"));
(20)            }
(21)    }
(22)    public static void main(String[] args) {
(23)            try {
(24)                    Class.forName("com.mysql.jdbc.Driver");
(25)            } catch (ClassNotFoundException cnfe) {
(26)                    System.err.println("Driver class not found");
(27)                    cnfe.printStackTrace();
(28)            }
(29)            Connection con = null;
(30)
(31)            try {
(32)                    con =
(33)                            (Connection) DriverManager.getConnection(
(34)                                    "jdbc:mysql://localhost/jdbctest/",
(35)                                    "mario",
(36)                                    "thePassword");
(37)            } catch (SQLException sqle) {
(38)                    System.err.println("Error establishing database connection");
(39)                    Throwable t = sqle;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)
(48)            Statement stmt = null;
(49)            try {
(50)                    stmt = (Statement) con.createStatement();
(51)            } catch (SQLException sqle) {
(52)                    System.err.println("Error creating SQL-Statement");
```

```
(53)                    Throwable t = sqle;
(54)                    while (t != null) {
(55)                    System.err.println("Type: " + t.getClass().getName());
(56)                    System.err.println("Message: " + t.getMessage());
(57)                    System.err.println("-----");
(58)                    t = t.getCause();
(59)                    }
(60)            }
(61)
(62)            try {
(63)                    int transactionIsolation = con.getTransactionIsolation();
(64)                    switch (transactionIsolation) {
(65)                            case Connection.TRANSACTION_NONE :
(66)                                    System.out.println("Transactions are not
supported");
(67)                                    break;
(68)                            case Connection.TRANSACTION_READ_UNCOMMITTED :
(69)                                    System.out.println(
(70)                                            "Dirty reads, non-repeatable reads and
phantom reads can occur");
(71)                                    break;
(72)                            case Connection.TRANSACTION_READ_COMMITTED :
(73)                                    System.out.println(
(74)                                            "Dirty reads are prevented; non-repeatable
reads and phantom reads can occur");
(75)                                    break;
(76)                            case Connection.TRANSACTION_REPEATABLE_READ :
(77)                                    System.out.println(
(78)                                            "Dirty reads and non-repeatable reads are
prevented; phantom reads can occur");
(79)                                    break;
(80)                            case Connection.TRANSACTION_SERIALIZABLE :
(81)                                    System.out.println(
(82)                                            "Dirty reads, non-repeatable reads and
phantom reads are prevented");
(83)                                    break;
(84)                    }
(85)                    if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
(86)                            con.setTransactionIsolation(
(87)                                    Connection.TRANSACTION_SERIALIZABLE);
(88)                            if (con.getTransactionIsolation()
(89)                                    != Connection.TRANSACTION_SERIALIZABLE) {
(90)                                    System.out.println(
(91)                                            "cannot set Connection.
TRANSACTION_SERIALIZABLE");
(92)                            } else {
(93)                                    System.out.println(
(94)                                            "reached highest possible isolation
level");
(95)                            }
(96)                    }
(97)
(98)                    con.setAutoCommit(false);
(99)                    stmt.executeUpdate(
(100)                           "INSERT INTO EMPLOYEE VALUES
('Hans','X','Hinterhuber','111111111',NULL,NULL,NULL,NULL,NULL,NULL);");
(101)                   stmt.executeUpdate(
(102)                           "INSERT INTO EMPLOYEE VALUES
('Franz','X','Obermüller','222222222',NULL,NULL,NULL,NULL,NULL,NULL);");
(103)                   printContent(stmt);
(104)                   //suppose error happens here
(105)                   Thread.sleep(5000);
(106)                   boolean error = true;
(107)                   if (error) {
(108)                           con.rollback();
(109)                   } else {
(110)                           stmt.executeUpdate(
(111)                                   "INSERT INTO EMPLOYEE VALUES
('Fritz','X','Meier','333333333',NULL,NULL,NULL,NULL,NULL,NULL);");
(112)                   }
(113)                   printContent(stmt);
(114)           } catch (SQLException sqle) {
(115)                   Throwable t = sqle;
(116)                   while (t != null) {
(117)                           System.err.println("Type: " + t.getClass().getName());
(118)                   System.err.println("Message: " + t.getMessage());
```

```
(119)                   System.err.println("-----");
(120)                   t = t.getCause();
(121)                   }
(122)           } catch (InterruptedException ie) {
(123)                   ie.printStackTrace();
(124)           }
(125)   }
(126)}
```

Download des Beispiels
Download der Ergebnisdatei

Neben dem Zurücksetzen einer vollständigen Transaktion bietet die JDBC-API auch die Möglichkeit alle Schritte bis zu einem anwenderdefierten aus der Datenbank zu entfernen.
Beispiel 65 zeigt dies unter Verwendung der Methode `setSavepoint` zur Definition eines Sicherungspunktes und `rollback(sp)` zum Zurücksetzen bis zu diesem Sicherungspunkt.

Beispiel 65: Transaktionsverarbeitung mit Sicherungspunkten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Savepoint;
(5)
(6)import com.mysql.jdbc.Connection;
(7)import com.mysql.jdbc.Statement;
(8)
(9)public class JDBCTransact2 {
(10)    private static void printContent(Statement stmt) throws SQLException {
(11)            ResultSet rs =
(12)                    stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
(13)            while (!rs.isLast()) {
(14)                    rs.next();
(15)                    System.out.println(
(16)                            rs.getString("LNAME")
(17)                                    + "\t"
(18)                                    + rs.getString("MINIT")
(19)                                    + "\t"
(20)                                    + rs.getString("LNAME"));
(21)            }
(22)    }
(23)    public static void main(String[] args) {
(24)            try {
(25)                    Class.forName("com.mysql.jdbc.Driver");
(26)            } catch (ClassNotFoundException cnfe) {
(27)                    System.err.println("Driver class not found");
(28)                    cnfe.printStackTrace();
(29)            }
(30)            Connection con = null;
(31)
(32)            try {
(33)                    con =
(34)                            (Connection) DriverManager.getConnection(
(35)                                    "jdbc:mysql://localhost/jdbctest/",
(36)                                    "mario",
(37)                                    "thePassword");
(38)            } catch (SQLException sqle) {
(39)                    System.err.println("Error establishing database connection");
(40)                    Throwable t = sqle;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            Statement stmt = null;
(50)            try {
(51)                    stmt = (Statement) con.createStatement();
(52)            } catch (SQLException sqle) {
(53)                    System.err.println("Error creating SQL-Statement");
(54)                    Throwable t = sqle;
(55)                    while (t != null) {
(56)                    System.err.println("Type: " + t.getClass().getName());
(57)                    System.err.println("Message: " + t.getMessage());
```

```
(58)                    System.err.println("-----");
(59)                    t = t.getCause();
(60)                    }
(61)            }
(62)
(63)            try {
(64)                    int transactionIsolation = con.getTransactionIsolation();
(65)                    switch (transactionIsolation) {
(66)                            case Connection.TRANSACTION_NONE :
(67)                                    System.out.println("Transactions are not
supported");
(68)                                    break;
(69)                            case Connection.TRANSACTION_READ_UNCOMMITTED :
(70)                                    System.out.println(
(71)                                            "Dirty reads, non-repeatable reads and
phantom reads can occur");
(72)                                    break;
(73)                            case Connection.TRANSACTION_READ_COMMITTED :
(74)                                    System.out.println(
(75)                                            "Dirty reads are prevented; non-repeatable
reads and phantom reads can occur");
(76)                                    break;
(77)                            case Connection.TRANSACTION_REPEATABLE_READ :
(78)                                    System.out.println(
(79)                                            "Dirty reads and non-repeatable reads are
prevented; phantom reads can occur");
(80)                                    break;
(81)                            case Connection.TRANSACTION_SERIALIZABLE :
(82)                                    System.out.println(
(83)                                            "Dirty reads, non-repeatable reads and
phantom reads are prevented");
(84)                                    break;
(85)                    }
(86)                    if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
(87)                            con.setTransactionIsolation(
(88)                                    Connection.TRANSACTION_SERIALIZABLE);
(89)                            if (con.getTransactionIsolation()
(90)                                    != Connection.TRANSACTION_SERIALIZABLE) {
(91)                                    System.out.println(
(92)                                            "cannot set Connection.
TRANSACTION_SERIALIZABLE");
(93)                            } else {
(94)                                    System.out.println(
(95)                                            "reached highest possible isolation
level");
(96)                            }
(97)                    }
(98)
(99)                    con.setAutoCommit(false);
(100)                   stmt.executeUpdate(
(101)                           "INSERT INTO EMPLOYEE VALUES
('Hans','X','Hinterhuber','111111111',NULL,NULL,NULL,NULL,NULL,NULL);");
(102)                   Savepoint sp = con.setSavepoint();
(103)                   stmt.executeUpdate(
(104)                           "INSERT INTO EMPLOYEE VALUES
('Franz','X','Obermüller','222222222',NULL,NULL,NULL,NULL,NULL,NULL);");
(105)                   printContent(stmt);
(106)                   //suppose error happens here
(107)                   Thread.sleep(5000);
(108)                   boolean error = true;
(109)                   if (error) {
(110)                           con.rollback(sp);
(111)                   }
(112)                   stmt.executeUpdate(
(113)                           "INSERT INTO EMPLOYEE VALUES
('Fritz','X','Meier','333333333',NULL,NULL,NULL,NULL,NULL,NULL);");
(114)                   printContent(stmt);
(115)                   con.commit();
(116)           } catch (SQLException sqle) {
(117)                   Throwable t = sqle;
(118)                   while (t != null) {
(119)                           System.err.println("Type: " + t.getClass().getName());
(120)                   System.err.println("Message: " + t.getMessage());
(121)                   System.err.println("-----");
(122)                   t = t.getCause();
(123)                   }
```

```
(124)           } catch (InterruptedException ie) {
(125)                   ie.printStackTrace();
(126)           }
(127)   }
(128)}
```

Download des Beispiels

Netzwerkverkehr

Der Netzwerkmitschnitt zeigt den TCP-Kommunikationsverlauf der SQL-Anfrage `SELECT * FROM EMPLOYEE;`.
Die vom Anfrager an den MySQL-Server übermittelten Datenanteile sind rot hervorgehoben. Zusätzlich sind die für eine gleichwertige native Kommunikation anfallenden Daten berücksichtigt. Ihre Anteile entsprechen den zusätzlich durch Fettdruck hervorgehobenen.

```
0000 4  00 00 00 \n 4  .  0  .  1  2  -  s  t  a  n  d  a  r  d  -  l  o  g  00 '  00 00 00
N  4  V
0020 5  G  D  a  9  00 ,     \b 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 00 00 01 87
00 ÿ  ÿ
0040 ÿ  m  a  r  i  o  00 N  W  N  J  S  ]  L  K  00 03 00 00 02 00 00 00 \t 00 00 00 02 j
d  b  c
0060 t  e  s  t  03 00 00 01 00 00 00 0f 00 00 00 03 S  H  O  W     V  A  R  I  A  B  L  E
S  01 00
0080 00 01 02 19 00 00 02 00 \r V  a  r  i  a  b  l  e  _  n  a  m  e  03 1e 00 00 01 þ  03
01 00 1f
00a0 11 00 00 03 00 05 V  a  l  u  e  03 00 01 00 01 þ  03 01 00 1f 01 00 00 04 þ  \f 00 00
05 \b b
00c0 a  c  k  _  l  o  g  02 5  0  7  00 00 06 07 b  a  s  e  d  i  r  .  /  o  p  t  /  r
a  i  d
00e0 /  m  y  s  q  l  -  s  t  a  n  d  a  r  d  -  4  .  0  .  1  2  -  p  c  -  l  i  n
u  x  -
0100 i  6  8  6  /  18 00 00 07 11 b  i  n  l  o  g  _  c  a  c  h  e  _  s  i  z  e  05 3
2  7  6
0120 8     00 00 \b 17 b  u  l  k  _  i  n  s  e  r  t  _  b  u  f  f  e  r  _  s  i  z  e
07 8  3
0140 8  8  6  0  8  15 00 00 \t \r c  h  a  r  a  c  t  e  r  _  s  e  t  06 l  a  t  i  n
1  á  00
0160 00 \n 0e c  h  a  r  a  c  t  e  r  _  s  e  t  s  Ñ  l  a  t  i  n  1     b  i  g
5     c  z
0180 e  c  h     e  u  c  _  k  r     g  b  2  3  1  2     g  b  k     l  a  t  i  n  1  _
d  e
01a0 s  j  i  s     t  i  s  6  2  0     u  j  i  s     d  e  c  8     d  o  s     g  e  r
m  a  n
01c0 1     h  p  8     k  o  i  8  _  r  u     l  a  t  i  n  2     s  w  e  7     u  s  a
7     c
01e0 p  1  2  5  1     d  a  n  i  s  h     h  e  b  r  e  w     w  i  n  1  2  5  1     e
s  t  o
0200 n  i  a     h  u  n  g  a  r  i  a  n     k  o  i  8  _  u  k  r     w  i  n  1  2  5
1  u  k
0220 r     g  r  e  e  k     w  i  n  1  2  5  0     c  r  o  a  t     c  p  1  2  5  7
l  a  t
0240 i  n  5  15 00 00 0b 11 c  o  n  c  u  r  r  e  n  t  _  i  n  s  e  r  t  02 O  N  12
00 00 \f
0260 0f c  o  n  n  e  c  t  _  t  i  m  e  o  u  t  01 5  17 00 00 \r 15 c  o  n  v  e  r
t  _  c
0280 h  a  r  a  c  t  e  r  _  s  e  t  00 <  00 00 0e 07 d  a  t  a  d  i  r  3  /  o  p
t  /  r
02a0 a  i  d  /  m  y  s  q  l  -  s  t  a  n  d  a  r  d  -  4  .  0  .  1  2  -  p  c  -
l  i  n
02c0 u  x  -  i  6  8  6  /  d  a  t  a  /  13 00 00 0f 0f d  e  l  a  y  _  k  e  y  _  w
r  i  t
02e0 e  02 O  N  19 00 00 10 14 d  e  l  a  y  e  d  _  i  n  s  e  r  t  _  l  i  m  i  t
03 1  0
0300 0  1b 00 00 11 16 d  e  l  a  y  e  d  _  i  n  s  e  r  t  _  t  i  m  e  o  u  t  03
3  0  0
0320 18 00 00 12 12 d  e  l  a  y  e  d  _  q  u  e  u  e  _  s  i  z  e  04 1  0  0  0  \n
00 00 13
0340 05 f  l  u  s  h  03 O  F  F  \r 00 00 14 \n f  l  u  s  h  _  t  i  m  e  01 0  !  00
00 00 15 11
0360 f  t  _  b  o  o  l  e  a  n  _  s  y  n  t  a  x  0e +     -  >  <  (  )  ~  *  :  "
"  &  |
0380 12 00 00 16 0f f  t  _  m  i  n  _  w  o  r  d  _  l  e  n  01 4  14 00 00 17 0f f  t
```

```
_ m a
03a0 x _ w o r d _ l e n 03 2 5 4 1c 00 00 18 18 f t _ m a x _ w o r
d _ l
03c0 e n _ f o r _ s o r t 02 2 0 1c 00 00 19 10 f t _ s t o p w o r
d _ f
03e0 i l e \n ( b u i l t - i n ) \f 00 00 1a \b h a v e _ b d b 02 N
O 0f 00
0400 00 1b \n h a v e _ c r y p t 03 Y E S 10 00 00 1c 0b h a v e _ i n
n o d
0420 b 03 Y E S 0e 00 00 1d \t h a v e _ i s a m 03 Y E S \r 00 00 1e \t h
a v e
0440 _ r a i d 02 N O 16 00 00 1f \f h a v e _ s y m l i n k \b D I S
A B L
0460 E D 10 00 00 \f h a v e _ o p e n s s l 02 N O 15 00 00 ! 10 h a
v e _
0480 q u e r y _ c a c h e 03 Y E S 0b 00 00 " \t i n i t _ f i l e
00 ( 00
04a0 00 # 1f i n n o d b _ a d d i t i o n a l _ m e m _ p o o l
_ s i
04c0 z e 07 2 0 9 7 1 5 2 ! 00 00 $ 17 i n n o d b _ b u f f e r _
p o o
04e0 l _ s i z e \b 1 6 7 7 7 2 1 6 - 00 00 % 15 i n n o d b _ d a
t a _
0500 f i l e _ p a t h 16 i b d a t a 1 : 1 0 M : a u t o e x t
e n d
0520 16 00 00 & 14 i n n o d b _ d a t a _ h o m e _ d i r 00 19 00 00
' 16 i
0540 n n o d b _ f i l e _ i o _ t h r e a d s 01 4 18 00 00 ( 15 i
n n o
0560 d b _ f o r c e _ r e c o v e r y 01 0 1c 00 00 ) 19 i n n o d
b _ t
0580 h r e a d _ c o n c u r r e n c y 01 8 ! 00 00 * 1e i n n o d
b _ f
05a0 l u s h _ l o g _ a t _ t r x _ c o m m i t 01 1 18 00 00 + 14
i n n
05c0 o d b _ f a s t _ s h u t d o w n 02 O N 15 00 00 , 13 i n n o
d b _
05e0 f l u s h _ m e t h o d 00 1c 00 00 - 18 i n n o d b _ l o c k
_ w a
0600 i t _ t i m e o u t 02 5 0 17 00 00 . 13 i n n o d b _ l o g _
a r c
0620 h _ d i r 02 . / 17 00 00 / 12 i n n o d b _ l o g _ a r c h i
v e 03
0640 O F F 1f 00 00 0 16 i n n o d b _ l o g _ b u f f e r _ s i z
e 07 8
0660 3 8 8 6 0 8 1d 00 00 1 14 i n n o d b _ l o g _ f i l e _ s i
z e 07
0680 5 2 4 2 8 8 0 1c 00 00 2 19 i n n o d b _ l o g _ f i l e s _
i n _
06a0 g r o u p 01 2 1d 00 00 3 19 i n n o d b _ l o g _ g r o u p _
h o m
06c0 e _ d i r 02 . / 1d 00 00 4 1a i n n o d b _ m i r r o r e d _
l o g
06e0 _ g r o u p s 01 1 1a 00 00 5 13 i n t e r a c t i v e _ t i m
e o u
0700 t 05 2 8 8 0 0 18 00 00 6 10 j o i n _ b u f f e r _ s i z e 06
1 3 1
0720 0 7 2 19 00 00 7 0f k e y _ b u f f e r _ s i z e \b 1 6 7 7 7
2 1 6
0740 L 00 00 8 \b l a n g u a g e B / o p t / r a i d / m y s q l
- s t
0760 a n d a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / s
h a r
0780 e / m y s q l / e n g l i s h / 17 00 00 9 13 l a r g e _ f i
l e s
07a0 _ s u p p o r t 02 O N 10 00 00 : \f l o c a l _ i n f i l e 02
O N 15
07c0 00 00 ; 10 l o c k e d _ i n _ m e m o r y 03 O F F \b 00 00 < 03
l o g
07e0 03 O F F 0f 00 00 = \n l o g _ u p d a t e 03 O F F 0b 00 00 > 07 l
o g _
0800 b i n 02 O N 16 00 00 ? 11 l o g _ s l a v e _ u p d a t e s 03
O F F
0820 15 00 00 @ 10 l o g _ s l o w _ q u e r i e s 03 O F F 11 00 00 A
\f l o
0840 g _ w a r n i n g s 03 O F F 13 00 00 B 0f l o n g _ q u e r y
```

```
_ t i
0860 m e 02 1 0 19 00 00 C 14 l o w _ p r i o r i t y _ u p d a t e
s 03 O
0880 F F 1b 00 00 D 16 l o w e r _ c a s e _ t a b l e _ n a m e s
03 O F
08a0 F 1b 00 00 E 12 m a x _ a l l o w e d _ p a c k e t 07 1 0 4 7
5 5 2
08c0 ! 00 00 F 15 m a x _ b i n l o g _ c a c h e _ s i z e \n 4 2
9 4 9
08e0 6 7 2 9 5 1b 00 00 G 0f m a x _ b i n l o g _ s i z e \n 1 0 7
3 7 4
0900 1 8 2 4 14 00 00 H 0f m a x _ c o n n e c t i o n s 03 1 0 0 16
00 00 I
0920 12 m a x _ c o n n e c t _ e r r o r s 02 1 0 17 00 00 J 13 m a
x _ d
0940 e l a y e d _ t h r e a d s 02 2 0 1d 00 00 K 13 m a x _ h e a
p _ t
0960 a b l e _ s i z e \b 1 6 7 7 7 2 1 6 19 00 00 L \r m a x _ j o
i n _
0980 s i z e \n 4 2 9 4 9 6 7 2 9 5 15 00 00 M 0f m a x _ s o r t _
l e n
09a0 g t h 04 1 0 2 4 17 00 00 N 14 m a x _ u s e r _ c o n n e c t
i o n
09c0 s 01 0 12 00 00 O 0e m a x _ t m p _ t a b l e s 02 3 2 00 00 P
14 m a
09e0 x _ w r i t e _ l o c k _ c o u n t \n 4 2 9 4 9 6 7 2 9 5
* 00 00
0a00 Q 1f m y i s a m _ m a x _ e x t r a _ s o r t _ f i l e _
s i z
0a20 e \t 2 6 8 4 3 5 4 5 6 % 00 00 R 19 m y i s a m _ m a x _ s o
r t _
0a40 f i l e _ s i z e \n 2 1 4 7 4 8 3 6 4 7 1b 00 00 S 16 m y i s
a m _
0a60 r e c o v e r _ o p t i o n s 03 O F F 00 00 T 17 m y i s a
m _ s
0a80 o r t _ b u f f e r _ s i z e 07 8 3 8 8 6 0 8 17 00 00 U 11 n
e t _
0aa0 b u f f e r _ l e n g t h 04 8 1 9 2 14 00 00 V 10 n e t _ r e
a d _
0ac0 t i m e o u t 02 3 0 13 00 00 W 0f n e t _ r e t r y _ c o u n
t 02 1
0ae0 0 15 00 00 X 11 n e t _ w r i t e _ t i m e o u t 02 6 0 \b 00 00
Y 03 n
0b00 e w 03 O F F 13 00 00 Z 10 o p e n _ f i l e s _ l i m i t 01 0
F 00 00
0b20 [ \b p i d _ f i l e < / o p t / r a i d / m y s q l - s t
a n d
0b40 a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / d a t
a / l i
0b60 n u x . p i d 0b 00 00 \ \t l o g _ e r r o r 00 \n 00 00 ] 04 p o
r t 04
0b80 3 3 0 6 14 00 00 ^ 10 p r o t o c o l _ v e r s i o n 02 1 0 18
00 00 _
0ba0 10 r e a d _ b u f f e r _ s i z e 06 1 3 1 0 7 2 1c 00 00 ` 14
r e a
0bc0 d _ r n d _ b u f f e r _ s i z e 06 2 6 2 1 4 4 14 00 00 a 11
r p l
0be0 _ r e c o v e r y _ r a n k 01 0 1a 00 00 b 11 q u e r y _ c a
c h e
0c00 _ l i m i t 07 1 0 4 8 5 7 6 13 00 00 c 10 q u e r y _ c a c h
e _ s
0c20 i z e 01 0 14 00 00 d 10 q u e r y _ c a c h e _ t y p e 02 O N
\f 00 00
0c40 e \t s e r v e r _ i d 01 1 17 00 00 f 11 s l a v e _ n e t _ t
i m e
0c60 o u t 04 3 6 0 0 19 00 00 g 15 s k i p _ e x t e r n a l _ l o
c k i
0c80 n g 02 O N 14 00 00 h 0f s k i p _ n e t w o r k i n g 03 O F F
17 00 00
0ca0 i 12 s k i p _ s h o w _ d a t a b a s e 03 O F F 13 00 00 j 10
s l o
0cc0 w _ l a u n c h _ t i m e 01 2 17 00 00 k 06 s o c k e t 0f / t
m p /
0ce0 m y s q l . s o c k 18 00 00 l 10 s o r t _ b u f f e r _ s i
z e 06
0d00 5 2 4 2 8 0 0b 00 00 m \b s q l _ m o d e 01 0 0f 00 00 n 0b t a b
```

```
l e _
0d20 c a c h e 02 6 4 12 00 00 o \n t a b l e _ t y p e 06 I N N O D
B 14 00
0d40 00 p 11 t h r e a d _ c a c h e _ s i z e 01 0 14 00 00 q \f t h
r e a
0d60 d _ s t a c k 06 1 9 6 6 0 8 1d 00 00 r \f t x _ i s o l a t i
o n 0f
0d80 R E P E A T A B L E - R E A D 0e 00 00 s \b t i m e z o n e 04
C E S
0da0 T 18 00 00 t 0e t m p _ t a b l e _ s i z e \b 3 3 5 5 4 4 3 2
\r 00 00
0dc0 u 06 t m p d i r 05 / t m p / 1c 00 00 v 07 v e r s i o n 13 4 .
0 . 1
0de0 2 - s t a n d a r d - l o g 13 00 00 w \f w a i t _ t i m e o
u t 05
0e00 2 8 8 0 0 01 00 00 x þ 11 00 00 00 03 S E T   a u t o c o m m i t
= 1 03
0e20 00 00 01 00 00 00 18 00 00 00 03 S E L E C T   *   F R O M   E M P L
O Y E
0e40 E ; 01 00 00 01 \n 19 00 00 02 \b E M P L O Y E E 05 F N A M E 03 \n 00
00 01 ý
0e60 03 01 00 00 19 00 00 03 \b E M P L O Y E E 05 M I N I T 03 01 00 00 01 þ
03 00 00
0e80 00 19 00 00 04 \b E M P L O Y E E 05 L N A M E 03 \n 00 00 01 ý 03 01 00
00 17 00
0ea0 00 05 \b E M P L O Y E E 03 S S N 03 \t 00 00 01 03 03 01 00 00 19 00 00 06
\b E M
0ec0 P L O Y E E 05 B D A T E 03 \n 00 00 01 \n 03 00 00 00 1b 00 00 07 \b E M
P L O
0ee0 Y E E 07 A D D R E S S 03 1e 00 00 01 ý 03 00 00 00 17 00 00 \b \b E M P
L O Y
0f00 E E 03 S E X 03 01 00 00 01 þ 03 00 01 00 1a 00 00 \t \b E M P L O Y E E
06 S A
0f20 L A R Y 03 07 00 00 01 05 03   00 02 1c 00 00 \n \b E M P L O Y E E \b S
U P E
0f40 R S S N 03 \t 00 00 01 03 03 00 00 00 17 00 00 0b \b E M P L O Y E E 03 D
N O 03
0f60 01 00 00 01 03 03 00 00 00 01 00 00 \f þ R 00 00 \r 04 J o h n 01 B 05 S m i
t h \t
0f80 1 2 3 4 5 6 7 8 9 \n 1 9 6 5 - 0 1 - 0 9 18 7 3 1   F o n d
r e n
0fa0 ,   H o u s t o n ,   T X 01 M \b 3 0 0 0 0 . 0 0 \t 3 3 3 4
4 5 5
0fc0 5 5 01 5 R 00 00 0e \b F r a n k l i n 01 T 04 W o n g \t 3 3 3 4
4 5 5
0fe0 5 5 \n 1 9 5 5 - 1 2 - 0 8 15 6 3 8   V o s s ,   H o u s t
o n ,
1000   T X 01 M \b 4 0 0 0 0 . 0 0 \t 8 8 8 6 6 5 5 5 5 01 5 T 00 00
0f 06 A
1020 l i c i a 01 J 06 Z e l a y a \t 9 9 9 8 8 7 7 7 7 \n 1 9 6 8
- 0 7
1040 - 1 9 17 3 3 2 1   C a s t l e ,   S p r i n g ,   T X 01 F
\b 2 5
1060 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 W 00 00 10 \b J e n n i f
e r 01
1080 S 07 W a l l a c e \t 9 8 7 6 5 4 3 2 1 \n 1 9 4 1 - 0 6 - 2
0 17 2
10a0 9 1   B e r r y ,   B e l l a i r e ,   T X 01 F \b 4 3 0 0
0 . 0
10c0 0 \t 8 8 8 6 6 5 5 5 5 01 4 V 00 00 11 06 R a m e s h 01 K 07 N a
r a y
10e0 a n \t 6 6 6 8 8 4 4 4 4 \n 1 9 6 2 - 0 9 - 1 5 18 9 7 5   F
i r e
1100   O a k ,   H u m b l e ,   T X 01 M \b 3 8 0 0 0 . 0 0 \t 3
3 3 4
1120 4 5 5 5 5 01 5 S 00 00 12 05 J o y c e 01 A 07 E n g l i s h \t 4
5 3 4
1140 5 3 4 5 3 \n 1 9 7 2 - 0 7 - 3 1 16 5 6 3 1   R i c e ,   H
o u s
1160 t o n ,   T X 01 F \b 2 5 0 0 0 . 0 0 \t 3 3 3 4 4 5 5 5 5 01
5 S 00
1180 00 13 05 A h m a d 01 V 06 J a b b a r \t 9 8 7 9 8 7 9 8 7 \n 1
9 6 9
11a0 - 0 3 - 2 9 17 9 8 0   D a l l a s ,   H o u s t o n ,   T
X 01 M
11c0 \b 2 5 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 G 00 00 14 05 J a m
```

```
e  s  01
11e0 E  04 B  o  r  g  \t 8  8  8  6  6  5  5  5  5  \n 1  9  3  7  -  1  1  -  1  0  16 4
5  0
1200 S  t  o  n  e  ,     H  o  u  s  t  o  n  ,     T  X  01 M  \b 5  5  0  0  0  .  0  0
û  01 1
1220 01 00 00 15 þ
```

Die Analyse des Datenverkehrs zeigt, daß im Falle der JDBC-basierten Kommunikation ein gegenüber der nativen Schnittstelle um 3529 Byte vergrößertes Datenaufkommen ausgetauscht wird. Diese zusätzliche Datenmenge fällt jedoch nur einmal zum Zeitpunkt des JDBC-Verbindungsaufbaus statisch an. (Vgl. Mitschnitt der mehrfachen Ausführung einer SQL-Anfrage innerhalb einer bestehenden JDBC-Verbindung)
Zusätzlich offenbart Zeile 0x40 des Datenverkehrs die verschlüsselte Übermittlung des Paßwortes des Anwenders *mario*. Allerdings werden die per Anfrage ermittelten Nutzdaten (ab Zeile 0xe40) unverschlüsselt über die Netzwerkschnittstelle übertragen und stellen somit ein potentielles Angriffsziel dar.
Abhilfe hierfür kann die Tunnelung des Datenverkehrs, beispielsweise mittels SSH, durch eine sichere Verbindung bieten.

Zugriff mit JDBC auf nicht-relationale Quellen

Zwar legt die JDBC-Schnittstelle inhärent eine relationale und damit tabellenorientierte Sicht der zu verarbeitenden Datenquelle zugrunde, jedoch ist die Nutzung der JDBC keinesweges auf jeden Anwendungsfall beschränkt.
Beispielsweise steht mit dem Produkt *Sunopsis XML Driver (JDBC Edition)* ein JDBC-Treiber für den dateibasierten Zugriff auf XML-Inhalte zur Verfügung.
Der Code des Beispiels 66 zeigt die Verwendung.

Als Beispiel dient folgende XML-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Employees>
        <Employee>
                <FName>John</FName>
                <Minit>B</Minit>
                <LName>Smith</LName>
        </Employee>
        <Employee>
                <FName>Franklin</FName>
                <Minit>T</Minit>
                <LName>Wong</LName>
        </Employee>
        <Employee>
                <FName>Alicia</FName>
                <Minit>J</Minit>
                <LName>Zelaya</LName>
        </Employee>
        <Employee>
                <FName>Jennnifer</FName>
                <Minit>S</Minit>
                <LName>Wallace</LName>
        </Employee>
        <Employee>
                <FName>Ramesh</FName>
                <Minit>K</Minit>
                <LName>Narayan</LName>
        </Employee>
        <Employee>
                <FName>Joyce</FName>
                <Minit>A</Minit>
                <LName>English</LName>
        </Employee>
</Employees>
```

Beispiel 66: JDBC-artige Verarbeitung von XML-Dateien

```
(1)import java.sql.Connection;
(2)import java.sql.DriverManager;
(3)import java.sql.ResultSet;
(4)import java.sql.SQLException;
(5)import java.sql.Statement;
(6)
(7)public class JDBCXML {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.sunopsis.jdbc.driver.xml.SnpsXmlDriver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)            try {
(17)                    con = (Connection) DriverManager.getConnection("jdbc:snps:xml");
(18)            } catch (SQLException e1) {
(19)                    System.err.println("Error establishing database connection");
(20)                    Throwable t = sqle;
(21)                    while (t != null) {
(22)                    System.err.println("Type: " + t.getClass().getName());
(23)                    System.err.println("Message: " + t.getMessage());
(24)                    System.err.println("-----");
(25)                    t = t.getCause();
(26)                    }
(27)            }
(28)            Statement stmt = null;
(29)            try {
(30)                    stmt = con.createStatement();
(31)            } catch (SQLException sqle) {
(32)                    Throwable t = sqle;
(33)                    while (t != null) {
(34)                    System.err.println("Type: " + t.getClass().getName());
(35)                    System.err.println("Message: " + t.getMessage());
(36)                    System.err.println("-----");
(37)                    t = t.getCause();
(38)                    }
(39)            }
(40)            try {
(41)                    stmt.execute("load file \"employee.xml\" on schema EMP readonly");
(42)                    stmt.execute("set schema EMP");
(43)                    ResultSet rs = con.getMetaData().getTables("", "EMP", "%", null);
(44)
(45)                    while (rs.next()) {
(46)                            System.out.println("TABLE=" + rs.getString("TABLE_NAME"));
(47)                            System.out.print("(");
(48)                            ResultSet colRS =
(49)                                    con.getMetaData().getColumns(
(50)                                            "",
(51)                                            "EMP",
(52)                                            rs.getString("TABLE_NAME"),
(53)                                            "%");
(54)                            while (colRS.next()) {
(55)                                    System.out.println(colRS.getString("COLUMN_NAME")
+ ",");
(56)                            }
(57)                            System.out.println(")\n");
(58)                    }
(59)            } catch (SQLException sqle) {
(60)                    Throwable t = sqle;
(61)                    while (t != null) {
(62)                    System.err.println("Type: " + t.getClass().getName());
(63)                    System.err.println("Message: " + t.getMessage());
(64)                    System.err.println("-----");
(65)                    t = t.getCause();
(66)                    }
(67)            }
(68)
(69)            ResultSet rs = null;
(70)            try {
(71)                    rs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(72)                    while (!rs.isLast()) {
(73)                            rs.next();
(74)                            System.out.println(rs.getString("FNAME"));
(75)                    }
```

```
(76)            } catch (SQLException sqle) {
(77)                    Throwable t = sqle;
(78)                    while (t != null) {
(79)                    System.err.println("Type: " + t.getClass().getName());
(80)                    System.err.println("Message: " + t.getMessage());
(81)                    System.err.println("-----");
(82)                    t = t.getCause();
(83)                    }
(84)            }
(85)
(86)            try {
(87)                    stmt.executeUpdate("INSERT INTO EMPLOYEE (FNAME, MINIT, LNAME)
VALUES('James', 'E', 'Borg');");
(88)            } catch (SQLException sqle) {
(89)                    Throwable t = sqle;
(90)                    while (t != null) {
(91)                    System.err.println("Type: " + t.getClass().getName());
(92)                    System.err.println("Message: " + t.getMessage());
(93)                    System.err.println("-----");
(94)                    t = t.getCause();
(95)                    }
(96)            }
(97)
(98)    }
(99)}
```

Download des Beispiels
Download der Ergebnisdatei

Web-Referenzen 9: Weiterführende Links

- JDBC @ SUN
- JDBC learning center @ SUN
- JDBC Tutorial
- JDBC FAQ @ JGuru.com
- G. Reese: *Database Programming with JDBC and Java*. O'Reilly, 1997
- Verhältnis von X/Open CLI und ODBC

3.2 Enterprise Java Beans (EJB)

Neben den bereits aus anderen Veranstaltungen bekannten Servlets und den davon abgeleiteten Java Server Pages bildet die Technik der *Enterprise Java Beans* (EJB) einen weiteren zentralen Baustein der Java 2 Enterprise Plattform. Als serverseitige Komponenten kommt den EJBs heute große Bedeutung in der Realisierung komplexer Anwendungen, insbesondere durch Umsetzung der sog. „Business Logik“, d.h. den nicht-interaktiven fachlichen Anwendungsteilen, zu.

Der Begriff der Enterprise Java Bean stützt sich auf dem historisch älteren der *Java Bean*. Eine solche stellt eine abgeschlossene wiederverwendbare Softwarekomponente dar, die nach ihrer Erstellung über festgelegte Schnittstellen parametrisiert und manipuliert werden kann. Hierzu muß eine Bean eine festgelegte Interaktionsschnittstelle bieten, die durch die Java Bean Spezifikation definiert ist. Es handelt sich dabei um eine Reihe von Konventionen, der eine Bean gehorchen muß, jedoch um keine festgelegte API, die durch eine Komponente zu implementieren ist.
Der Begriff der Enterprise Java Bean greift diese inhaltliche Fundierung auf und präzisiert gleichzeitig die technische Umsetzung. So stellt eine Enterprise Java Bean eine Softwarekomponente dar, die in einer festgelegten Ausführungsumgebung, welche durch die EJB-Spezifikation festgelegte Dienste zur Nutzung durch die Beans anbieten kann. Eine solche Ausführungsumgebung wird als *Container* bezeichnet.
Ziel der Trennung in Komponente und Ausführungsumgebung ist die Zielsetzung, die Enterprise Java Bean ausschließlich zur Umsetzung fachlicher Aufgaben heranzuziehen und alle infrastrukturellen Fragestellungen wie Betriebsmittelverwaltung, Persistenz oder Sicherheit durch die Ausführungsumgebung in gleicher Weise für alle Komponenten bereitzustellen.
Ein EJB-Container wird zumeist im Rahmen eines Application-Servers bereitgestellt.
Die gelegentlich anzutreffende Hervorhebung der anfänglich für Java Beans intendierten *visuellen Manipulationsmöglichkeit* trifft für Enterprise Java Beans nicht zu und hat sich für Java Beans auch nur begrenzte Bedeutung erlangt.

Spezifikationsgemäß können EJB-Container folgende Eigenschaften offerieren:

- **Betriebsmittelverwaltung**
  Typischerweise verwaltet ein einziger EJB-Container gleichzeitig eine Reihe verschiedener Enterprise Java Beans. Zur Organisation und Aufrechterhaltung der Ausführbarkeit obliegt dem Container die Zuteilung von Betriebsmitteln wie Hauptspeicher, CPU-Zeit oder Netzwerkressourcen an die verwalteten EJB.

Hierunter fällt insbesondere auch die Einlagerung, Instanziierung und Entfernung der EJBs selbst.

- **Zustandsverwaltung**
  In praktischen Anwendungen ist oft die Nutzung zustandsbehafteter Kommunikation, die sich über verschiedene Einzelinteraktionen erstreckt gewünscht. Die hierfür notwendigen technischen Voraussetzungen (Zustandsspeicherung, Korrelation der Einzelinteraktionen) werden durch den Container bereitgestellt.
- **Transaktionsverwaltung**
  Erweiterung der Zustandsverwaltung. Zur Gewährleistung des benötigten Verhaltens müssen EJBs keine eigenen Implementierungen zur Verfügung stellen, sondern können vorhandene Dienste des Containers nutzen.
- **Sicherheit**
  Die Sicherheit von EJBs kann durch Vergabe von Zugriffsrechten und Rollen auf Containerebene gesteuert werden.
- **Persistenz**
  Der interne Zustand einer verwalteten EJB kann wahlfrei in persistiert und zu einem späteren Zeitpunkt wiederhergestellt werden.
- **Entfernter Zugriff**
  Der Zugriff auf EJBs erfolgt mittels Remote Method Invocation und ist daher Lokationstransparent.

Neben den in der Aufzählung dargestellten Eigenschaften dürfen Container zusätzlich Weitere wahlfrei implementieren.

EJB-Typen

Grundsätzlich lassen sich alle EJBs drei Typen zuordnen: *Session Beans*, *Entity Beans* und *Message Driven Beans*. Während erstere hauptsächlich zur Abbildung von Abläufen eingesetzt werden, dienen Entity Beans der Abwicklung von Zugriffen auf Daten. Eine Sonderstellung nehmen die *Message Driven Beans* ein, die lediglich hinsichtlich ihres Kommunikationsverhaltens festgelegt sind.

Session Beans dienen der Abbildung von Abläufen im Rahmen der Programmierung der sog. Business Logik. Die Lebensdauer (d.h. Zeitspanne zwischen Erzeugung im und Entfernung aus dem Hauptspeicher) ist daher identisch mit der einer durch den Client erfolgenden Anfrage. Jede zu einem Zeitpunkt existierende Session Bean repräsentiert daher eine zugehörige Clientinstanz.
Nach ihrer internen Ausgestaltung werden *stateless* und *statefull* Session Beans unterschieden. Während Erstere keinen über einen einzigen Aufruf hinausgehenden Zustand verwalten und daher seiteneffektfrei lediglich auf den durch den Aufruf übermittelten Daten operieren erhält das zustandserhaltende Pendant die Daten eines Aufrufs und kann diese auch in nachfolgenden Aufrufen verarbeiten.

Entity Beans sind programmiersprachliche Stellvertreter datenbankresidenter Objekte. Sie dienen dem erleichterten Zugriff auf persistent vorliegende Datenbestände. Ihre interne Realisierung ist eng mit der Technik relationaler Datenbankmanagement System verbunden. So werden Sie durch einen anwenderdefinierten Primärschlüssel dauerhaft identifiziert.

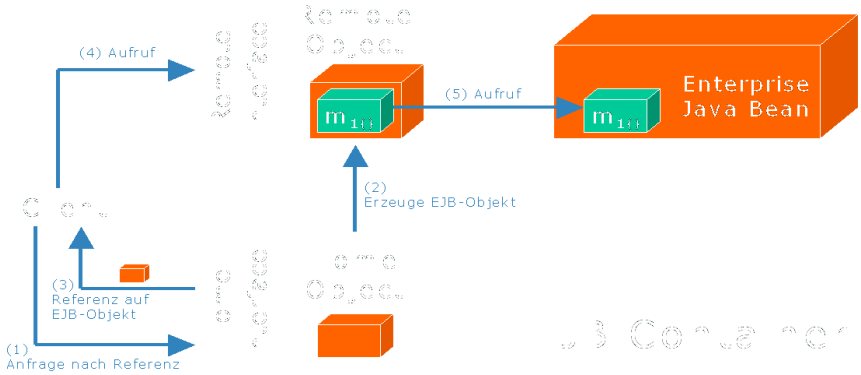
Message Driven Beans sind hinsichtlich ihres Kommunikationsverhaltens auf asynchrone Aufrufe beschränkt. Die Realisierung des eigentlichen Verhaltens wird durch eine Ausprägung eines der anderen Beantypen geboten.

Session Beans

Konzeptionell umfaßt jede EJB-Anwendung, die Session Beans einsetzt, die in Abbildung 6 dargestellten Teile:

- Implementierung der EJB selbst.
- *Remote*-Schnittstelle zum Zugriff auf die durch die Bean publizierten Methoden.
- *Home*-Schnittstelle zur Ermittlung einer Referenz auf das Bean-Objekt.

Abbildung 6: *Aufrufstruktur einer zustandslosen EJB*



(click on image to enlarge!)

Gegenüber der Realisierung als RMI-Anwendung benötigt die Umsetzung als zustandslose Session Bean die Erstellung einer sog. „Home-Schnittstelle“ (*Home Interface*), welche die Operation `create` zur Instanziierung des serverseitigen EJB-Objekts bietet.
Sie ist im Beispiel 67 dargestellt.

Beispiel 67: Home-Schnittstelle einer EJB

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.EJBHome;
(4)
(5)public interface SayHelloHome extends EJBHome {
(6)     public SayHello create() throws RemoteException, CreateException;
(7)}
```

Download des Beispiels

Die anwenderdefinierte Home-Schnittstelle erweitert die durch die Standard-API vorgegebene Schnittstelle `EJBHome`. Diese definiert Operationen zur Entfernung existierender EJB-Objekte aus dem Hauptspeicher (`remove`) sowie zur Ermittlung von Metadaten (`getEJBMetaData`) oder zum Erhalt eines netzwerkunabhängigen Verweises auf das EJB-Objekt (`getHomeHandle`).
Im Einzelnen sind dies die Operationen:

- `EJBMetaData getEJBMetaDate()`
  Liefert ein `EJBMetaData`-Objekt welches einzelne Eigenschaften einer EJB näher beschreibt. Hierzu zählen:
  - Klasse der Home-Schnittstelle
  - Klasse des Primärschlüssels (nur vorhanden sofern es sich um eine Entity Bean handelt)
  - Klasse der Remote-Schnittstelle
  - Boole'scher Wert, der angibt, ob es sich um eine Session Bean handelt
  - Boole'scher Wert, der angibt, ob es sich um eine zustandslose Session Bean handelt
- `HomeHandle getHomeHandle()`
  Liefert ein Objekt des Typs `HomeHandle` zurück, welches eine netzwerkunabhängige Abstraktion des Verweises auf das Home-Objekt realisiert.
- `void remove (Handle h)`
  Entfernt ein durch den Objektverweis (`Handle`) identifiziertes EJB-Objekt aus dem Hauptspeicher.
- `void remove (Object pk)`
  Entfernt ein durch das übergebene Primärschlüsselobjekt identifiziertes EJB-Objekt aus dem Hauptspeicher.

Interessanterweise definiert die Schnittstelle zwar Operationen zur Ermittlung von Daten über bestehende Objekte und zur Entfernung dieser Objekte aus dem Hauptspeicher, nicht jedoch zu ihrer Erzeugung.
Dies liegt in der durch die Programmiersprache Java angestrebten statischen Typsicherheit begründet, die es nicht gestattet Operationen mit variablen Parameterlisten --- wie sie für die zum API-Erstellungszeitpunkt unbekannten spezifischen Initialisierungsparameter aller denkbaren EJBs benötigt würden --- zu versehen.
Aus diesem Grunde definiert die EJB-Spezifikation informell, daß ein diese Schnittstelle erweiternde eigene Home-Schnittstelle zusätzlich die Methode `create`, deren Signatur als Rückgabetyp den Typ der Remote-Schnittstelle vorsehen muß definiert. Zusätzlich enthält diese Operation die zur Initialisierung der Bean benötigten Parameter in ihrer Parameterliste.

Die im Beispiel 68 ist Remote-Schnittstelle dargestellt, deren Ausprägungen von Home-Objekten angesprochenen werden:

Beispiel 68: Remote-Schnittstelle einer EJB

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.EJBObject;
(3)
(4)public interface SayHello extends EJBObject {
(5)     public String sayHello(String name) throws RemoteException;
(6)}
```

Download des Beispiels

Schnittstellen dieses Typs enthalten ausschließlich die fachlichen Operationen, d.h. die Signaturen der Methoden, die später durch den Client benutzt werden.
Jede Remote-Schnittstelle erweitert zusätzlich die vorgegebene Schnittstelle `EJBObject`, welche, ähnlich zur Home-Schnittstelle, einige Operationen zur Verwaltung eines EJB-Objektes vorgibt:

- `EJBHome getEJBHome()`
  Liefert die Home-Schnittstelle einer EJB.
- `Handle getHandle()`
  Liefert ein Objekt des Typs `HomeHandle` zurück, welches eine netzwerkunabhängige Abstraktion des Verweises auf das Home-Objekt realisiert.
- `Object getPrimaryKey()`

Liefert das Primärschlüsselobjekt einer Entity Bean.

- `boolean isIdentical(EJBObject eo)`
  Prüft ob das übergebene EJB-Objekt dasselbe wie das Objekt ist auf dem die Methode ausgeführt wird.
- `void remove()`
  Entfernt das EJB-Objekt aus dem Bean-Container.

Beispiel 69 zeigt die Implementierung der Bean selbst:

Beispiel 69: Realisierung einer Session Bean

```
(1)import java.rmi.RemoteException;
(2)import java.util.Date;
(3)import javax.ejb.EJBException;
(4)import javax.ejb.SessionBean;
(5)import javax.ejb.SessionContext;
(6)
(7)public class HelloWorldBean implements SessionBean {
(8)     public HelloWorldBean() {
(9)     }
(10)
(11)    public String sayHello(String name) {
(12)            return ("Hello " + name + " it is now " + new Date().toString());
(13)    }
(14)
(15)    public void setSessionContext(SessionContext arg0)
(16)            throws EJBException, RemoteException {
(17)    }
(18)    public void ejbRemove() throws EJBException, RemoteException {
(19)    }
(20)
(21)    public void ejbCreate() throws EJBException {
(22)    }
(23)
(24)    public void ejbActivate() throws EJBException, RemoteException {
(25)    }
(26)
(27)    public void ejbPassivate() throws EJBException, RemoteException {
(28)    }
(29)}
```

Download des Beispiels

Die programmiersprachliche Umsetzung der Bean enthält die Methoden der in der Remote-Schnittstelle bekanntgegebenen fachlichen Operationen. Zusätzlich muß ein Konstruktur expliziert werden, dessen Parameterliste mit den für die Operation `create` des Home-Interfaces gegebenen übereinstimmen. Spezifikationsgemäß muß jede Session Bean die gleichnamige API-Schnittstelle implementieren. Diese definiert einige Operationen zur Behandlung unterschiedlicher Lebenszyklusstadien einer EJB. Hierunter fallen Methoden, die beim Erzeugen (`ejbCreate`), Entfernen (`ejbRemove`), bei der Passivierung (d.h. Auslagerung auf Hintergrundspeicher) (`ejbPassivate`) und dessen Reaktivierung (`ejbActivate`) eines EJB-Objekts durch die Ausführungsumgebung aufgerufen werden.

Bei der zwingend zu implementierenden Schnittstelle `SessionBean` handelt es sich nicht nur um eine Konvention um die Umsetzung der Lebenszyklusschnittstelle sicherzustellen, sondern auch um die Kategorisierung der Bean selbst. So stellt die im Beispiel verwandte Schnittstelle `SessionBean` neben `EntityBean` und `MessageDrivenBean` eine Spezialisierung der (operationslosen) Schnittstelle `EnterpriseBean` dar, deren „Implementierung“ durch eine Klasse lediglich zur Kennzeichnung dieser als EJB herangezogen wird.
Die genannten Spezialisierungen dieser Schnittstelle erfüllen daher sowohl den Zweck der Ausübung des Implementierungszwanges für die in ihnen aufgeführten Operationen als auch den der typisierenden Kennzeichnung.
Darüberhinaus ist `EnterpriseBean` als Spezialisierung der Standard-Schnittstelle `Serializable` angelegt. In der Konsequenz muß jedes EJB-Objekt durch die Javasprachmechnismen serialisierbar sein. Diese Eigenschaft wird insbesondere für die Passivierung und im Rahmen der Entity Beans genutzt.

Konzeptionell erinnert die Trennung in publizierte fachliche Schnittstelle (Remote-Schnittstelle) und deren technischer Umsetzung durch die EJB an die aus der Betrachtung des Remote Method Invocation Mechanismus bekannte Struktur.
Allerdings weicht die Umsetzung der Bean von der dort anzutreffenden Konvention ab, die publizierte Schnittstelle selbst durch die realisierende Klasse zu implementieren. Dies liegt vor allem an der gegenüber RMI veränderten Struktur der publizierten Schnittstelle begründet. Während RMI für die Schnittstelle die Spezialisierung der operationslosen Standardschnittstelle `Remote` fordert die EJB-Spezifikation die Erweiterung der Schnittstelle `EJBObject`, welche selbst die oben dargestellten Operationen definiert. Aus diesem Grunde würde die Aufnahme der Remote-Schnittstelle, obwohl konzeptionell durchaus zu rechtfertigen, in die Umsetzungsliste der EJB gleichzeitig die Implementierung

von zumindest leeren Methodenrümpfen für die in EJBObject definierten Operationen notwendig werden lassen.
Abgesehen von dieser Ausnahme rekonstruiert das Verhältnis zwischen EJB und deren Remote-Schnittstelle die aus RMI bekannte Beziehung zwischen Schnittstelle und Umsetzung.

Die Nutzung einer durch eine Java Bean angebotenen Funktionalität erfolgt gemäß dem in Abbildung 6 dargestellten Schema. Ein dies umsetzender Client ist in Beispiel 70 dargestellt.

Beispiel 70: Zugriff auf eine Session Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.naming.Context;
(4)import javax.naming.InitialContext;
(5)import javax.naming.NamingException;
(6)import javax.rmi.PortableRemoteObject;
(7)
(8)public class CallHelloWorldBean {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Context initial = new InitialContext();
(12)                    Object objRef = initial.lookup("helloBean");
(13)
(14)                    SayHelloHome home =
(15)                            (SayHelloHome) PortableRemoteObject.narrow(
(16)                                    objRef,
(17)                                    SayHelloHome.class);
(18)                    SayHello sh = home.create();
(19)
(20)                    System.out.println(sh.sayHello("Mario"));
(21)
(22)            } catch (NamingException ne) {
(23)                    ne.printStackTrace();
(24)            } catch (RemoteException re) {
(25)                    re.printStackTrace();
(26)            } catch (CreateException ce) {
(27)                    ce.printStackTrace();
(28)            }
(29)    }
(30)}
```

Download des Beispiels

Zunächst ermittelt der Client unter Nutzung der JNDI-API eine Referenz auf die EJB. Dies geschieht durch Anfrage (lookup) an den JNDI-Dienst unter Übergabe des bekannten Klarnamens (*helloBean*).
Die erhaltene generische Referenz wird durch Aufruf der statischen Methode narrow der Klasse PortableRemoteObject typsicher in eine Ausprägung der Home-Schnittstelle konvertiert.
Der Aufruf der in dieser Schnittstelle durch den Anwender definierten create-Methode sorgt für die serverseitige Instanziierung der EJB, die als Ausprägung der Remote-Schnittstelle geliefert wird.
Tatsächlich wird nicht das EJB-Objekt selbst durch den Methodenaufruf retourniert, sondern lediglich ein netzwerktransparenter Verweis darauf, der jedoch clientseitig einer lokalen Objektreferenz gleichgestellt verwendet werden kann.
Ferner wird serverseitig zur Kommunikation mit der EJB ein Home-Objekt erzeugt, welchem eine Stellvertreterrolle für den anfragenden Client zukommt.
Der Aufruf der durch die EJB zur Verfügung gestellten Methode erfolgt identisch zu dem einer Lokalen.

Entity Beans

Die zweite zentrale Klasse von Enterprise Java Beans bilden die zur serverseitigen Persistierung von Objekten dienenden *Entity Beans*.
Sie kapseln Datenbankinhalte durch Objekte, die gemäß der EJB-Spezifikation instanziierbar und zugreifbar sind. Die Verwaltung der gekapselten Dateninhalte erfolgt durch ein frei festlegbares Datenbankmanagementsystem, die der Objekte selbst durch den EJB-Container.
Ziel dieser Technik ist es die Komplexität der Persistenzlogik für den Verwender der bereitgestellten Bean vollkommen transparent zu gestalten und serverseitig zu realisieren.
Die Familie der Entity Beans selbst zerfällt in zwei Untertypen, welche sich entlang des Realisierungspunktes der Persistenzlogik separieren: *Bean Managed Persistence*, der Bean obliegt die Umsetzung der Persistenzlogik, und *Container Managed Persistence*, hierbei wird die Persistenzlogik durch den EJB-Container realisiert.

Das nachfolgende Beispiel zeigt die Umsetzung einer Entity Bean mit Bean Managed Persistence. Es kapselt die Verwaltung und den Zugriff auf Objekte, die Personen beschreiben. Jedes Personen-Objekt enthält Daten zu Name, Geburtsdatum und Wohnstraße. Der Name dient als eindeutige Identifikation und

daher datenbankseitig als Primärschlüssel. Die notwendige Datenbanktabelle wurde erzeugt durch den SQL-Ausdruck:

```
CREATE TABLE PERSON( Name VARCHAR(20) PRIMARY KEY, Birthdate DATE, Street VARCHAR(30));
```

Wie bereits für die Realisierung von Session Beans eingeführt, werden auch zur Publikation der extern zugänglichen Schnittstellen Home und Remote Interfaces benötigt.

Struktur und Aufbau der Home-Schnittstelle ähnelt konzeptionell der für Session Beans eingeführten. Dieser Schnittstellentyp dient auch für Entity Beans zur Aufnahme der Verwaltungsoperationen zur Erzeugung (`create`) und zur Suche existierender EJBs (`findByPrimaryKey`).
Beispiel 71 zeigt die Home-Schnittstelle des Beispiels.

Beispiel 71: Home-Schnittstelle einer Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.EJBHome;
(4)import javax.ejb.FinderException;
(5)
(6)public interface PersonHome extends EJBHome {
(7)     public Person create(int year, int month, int day, String name, String street)
throws CreateException, RemoteException;
(8)     public Person findByPrimaryKey(String name) throws FinderException,
RemoteException;
(9)}
```

Download des Beispiels

Die Home-Schnittstelle zeigt die `create`-Operation zur Erzeugung einer neuen EJB-Instanz. Ihre Übergabeparameter dienen zur Konstruktion des neuen Objekts und werden durch die Bean-Implementierung interpretiert.
Ferner enthält die Schnittstelle mit `findByPrimaryKey` eine Operation, deren Implementierung eine Entity-Bean anhand ihres Primärschlüssels identifiziert und liefert. Aus diesem Grunde erhält die Methode den zu suchenden Wert vom Typ des Primärschlüssels übergeben.
Hinsichtlich der verwendeten Typen zeigt sich bereits hier, daß eine Abbildung der durch die Programmiersprache Java bereitgestellten Typen auf die des eingesetzten Persistenzsystems stattfinden muß. In der im Beispiel gewählten Ausführungsform der durch die EJB selbst verwalteten Persistenz muß diese Abbildung manuell durch den Programmierer bereitgestellt werden.

Die Remote-Schnittstelle gibt die für Nutzer der Bean zugänglichen Geschäftsfunktionen wieder. Daher enthält dieser Schnittstellentyp lediglich Operationen zum Zugriff auf die verwalteten Daten, nicht jedoch zur technischen Verwaltung und Interaktion mit dem Bean-Container.
Per Konvention muß diese Schnittstelle als Spezialisierung der Schnittstelle `EJBObject` definiert sein. Diese Standardschnittstelle definiert allgemeine Interaktionsformen, wie Löschen (`remove`), Vergleich (`isIdentical`) und Ermittlung des Primärschlüsselwertes (`getPrimaryKey`) die für alle Entity Bean Objekte gleichermaßen benötigt werden.
`isIdentical` liefert den Vergleich zweier serverseitiger EJB-Objekte und ermittelt so, ob zwei Java-Objektreferenzen auf dasselbe Datenbankobjekt zugreifen.
`getPrimaryKey` ermittelt den Wert des Primärschlüssels eines gegebenen EJB-Objekts aus der Datenbank.
Zur Lösung von datenbankresidenten Objekten wird `remove` eingesetzt. Der Aufruf dieser Methode entfernt ausschließlich die durch die EJB repräsentierten Datenbanktupel, die programmiersprachliche Objektrepräsentation bleibt jedoch über die gesamte Laufzeit (sofern nicht durch Gültigkeitsbereiche oder explizite `NULL`-Setzung explizit anders gehandhabt) intakt.

Beispiel 72: Remote-Schnittstelle einer Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.EJBObject;
(3)
(4)public interface Person extends EJBObject {
(5)     public int getAge() throws RemoteException;
(6)     public String getStreet() throws RemoteException;
(7)     public void setStreet(String street) throws RemoteException;
(8)}
```

Download des Beispiels

Beispiel 73 zeigt den vollständigen Code der Bean. Sie implementiert mit `EntityBean` die Standardschnittstelle aller Entity Beans, welche als Spezialisierung der ausschließlich markierenden Schnittstelle `EnterpriseBean` die notwendigen Basisoperationen zur Abwicklung der persistenten Speicherung bieten.

Im Falle einer *Bean Managed Persistence* enthalten die Methoden der durch die Schnittstelle definierten und der zusätzlich im Rahmen der Spezifikation textuell definierten Operationen die notwendigen Aufrufe zur Ablage eines Objekts in der Datenbank und zu seiner späteren Extraktion daraus.
Im Einzelnen sind dies die Operationen:

Tabelle 24: Persistenzoperationen einer Entity Bean

Operation	Semantik	Zugehörige SQL-Anweisung
`ejbCreate`	Wird nach dem Erzeugen eines Java-Objektes aufgerufen um dieses in der Datenbank abzulegen. Diese Operation ist nicht Bestandteil der Schnittstelle, da ihre Parameter, die den Übergabeparametern des Objektkonstruktors entsprechen, zum Schnittstellenerzeugungszeitpunkt nicht feststehen.	`INSERT`
`ejbFindByPrimaryKey`	Liefert den Wert des Primärschlüssels zurück, sofern ein Datenbankeintrag existiert, der durch diesen Primärschlüssel identifiziert wird. Diese Operation ist nicht Bestandteil der Schnittstelle, da ihre Parameter, die in Typ, Name und Reihenfolge der Zusammensetzung des Primärschlüssels entsprechen, zum Schnittstellenerzeugungszeitpunkt nicht feststehen. Diese Methode wird nicht durch den Anwender direkt aufgerufen, sondern stattdessen auf einer Ausprägung der Home-Schnittstelle das dort zur Verfügung stehende Analogon `findByPrimaryKey`, welches das durch den Primärschlüssel identifizierte EJB-Objekt zurückliefert. Diese Methode greift intern auf ausschließlich den Schlüssel liefernde Methode der Bean zu.	`SELECT`
`ejbRemove`	Entfernt das EJB-Objekt aus der Datenbank.	`DELETE`
`ejbStore`	Synchronisiert das Java-Objekt mit dem EJB-Objekt und aktualisiert so die Datenbankinhalte. Diese Methode wird nach jedem Zugriff mittels einer in der Remote-Schnittstelle aufgeführten Operation auf das EJB-Objekt ausgeführt.	`UPDATE`

Beispiel 73: Eine Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import java.sql.Connection;
(3)import java.sql.DriverManager;
(4)import java.sql.ResultSet;
(5)import java.sql.SQLException;
(6)import java.sql.Statement;
(7)import java.util.Calendar;
(8)import java.util.GregorianCalendar;
(9)
(10)import javax.ejb.CreateException;
(11)import javax.ejb.EJBException;
(12)import javax.ejb.EntityBean;
(13)import javax.ejb.EntityContext;
(14)import javax.ejb.FinderException;
(15)import javax.ejb.RemoveException;
(16)
(17)public class PersonBean implements EntityBean {
(18)    //persistent fields
(19)    private String name;
(20)    private GregorianCalendar birthdate;
(21)    private String street;
(22)
(23)    public PersonBean() {
(24)    }
(25)
```

```
(26)    //methods which are part of the remote interface
(27)    public int getAge() {
(28)            return (
(29)                    new GregorianCalendar().get(Calendar.YEAR)
(30)                            - birthdate.get(Calendar.YEAR));
(31)    }
(32)    public String getStreet() {
(33)            return street;
(34)    }
(35)    public void setStreet(String street) throws RemoteException {
(36)            if (street.length() <= 30) {
(37)                    this.street = street;
(38)            } else {
(39)                    throw new RemoteException("cannot update street");
(40)            }
(41)    }
(42)    //      ---------------------------------------------------------------------
(43)    public String ejbCreate(
(44)            int year,
(45)            int month,
(46)            int day,
(47)            String name,
(48)            String street)
(49)            throws CreateException {
(50)            if (year > 1900
(51)                    && month >= 1
(52)                    && month <= 12
(53)                    && day >= 1
(54)                    && day <= 31
(55)                    && name.length() <= 20
(56)                    && street.length() <= 30) {
(57)
(58)                    this.name = name;
(59)                    this.birthdate = new GregorianCalendar(year, month, day);
(60)                    this.street = street;
(61)                    Statement stmt = getStatement();
(62)                    String s =
(63)                            new String(
(64)                                    "INSERT INTO PERSON VALUES ('"
(65)                                            + name
(66)                                            + "','"
(67)                                            + birthdate.get(Calendar.YEAR)
(68)                                            + "-"
(69)                                            + birthdate.get(Calendar.MONTH)
(70)                                            + "-"
(71)                                            + birthdate.get(Calendar.DATE)
(72)                                            + "','"
(73)                                            + "'"
(74)                                            + street
(75)                                            + "'"
(76)                                            + ");");
(77)                    try {
(78)                            stmt.executeUpdate(s);
(79)                    } catch (SQLException e) {
(80)                            e.printStackTrace();
(81)                    }
(82)            } else {
(83)                    throw new CreateException("Invalid values supplied");
(84)            }
(85)            return name;
(86)    }
(87)    public void ejbPostCreate(
(88)            int year,
(89)            int month,
(90)            int day,
(91)            String name,
(92)            String street) {
(93)    }
(94)    //      ---------------------------------------------------------------------
(95)    public int ejbHomeGetAge() {
(96)            return 0;
(97)    }
(98)
(99)    public String ejbHomeGetStreet() {
(100)           return new String();
(101)   }
```

```
(102)
(103)   public void ejbHomeSetStreet(String street) {
(104)   }
(105)
(106)   public Statement getStatement() {
(107)           try {
(108)                   Class.forName("com.mysql.jdbc.Driver");
(109)           } catch (ClassNotFoundException e) {
(110)                   e.printStackTrace();
(111)           }
(112)           Connection con = null;
(113)           try {
(114)                   con =
(115)                           (Connection) DriverManager.getConnection(
(116)                                   "jdbc:mysql://10.0.0.1/Address/",
(117)                                   "mario",
(118)                                   "thePassword");
(119)           } catch (SQLException e1) {
(120)                   e1.printStackTrace();
(121)           }
(122)           try {
(123)                   return ((Statement) con.createStatement());
(124)           } catch (SQLException e2) {
(125)                   e2.printStackTrace();
(126)           }
(127)           return null; //never gets here
(128)   }
(129)
(130)   // --------------------------------------------------------------------
(131)   public void setEntityContext(EntityContext ectx)
(132)           throws EJBException, RemoteException {
(133)   }
(134)
(135)   public void unsetEntityContext() throws EJBException, RemoteException {
(136)   }
(137)
(138)   public void ejbRemove()
(139)           throws RemoveException, EJBException, RemoteException {
(140)           Statement stmt = getStatement();
(141)           String s = new String("DELETE FROM PERSON WHERE Name='" + name + "';");
(142)           try {
(143)                   stmt.executeUpdate(s);
(144)           } catch (SQLException e) {
(145)                   e.printStackTrace();
(146)           }
(147)   }
(148)
(149)   public void ejbActivate() throws EJBException, RemoteException {
(150)   }
(151)
(152)   public void ejbPassivate() throws EJBException, RemoteException {
(153)   }
(154)
(155)   public void ejbLoad() throws EJBException, RemoteException {
(156)   }
(157)
(158)   public void ejbStore() throws EJBException, RemoteException {
(159)           Statement stmt = getStatement();
(160)           String s =
(161)                   new String(
(162)                           "UPDATE PERSON SET Name='"
(163)                                   + name
(164)                                   + "', BDATE='"
(165)                                   + birthdate.get(Calendar.YEAR)
(166)                                   + "-"
(167)                                   + birthdate.get(Calendar.MONTH)
(168)                                   + "-"
(169)                                   + birthdate.get(Calendar.DATE)
(170)                                   + "', Street = '"
(171)                                   + street
(172)                                   + "' WHERE Name = '"
(173)                                   + name
(174)                                   + "';");
(175)           try {
(176)                   stmt.executeUpdate(s);
(177)           } catch (SQLException e) {
```

```
(178)                   e.printStackTrace();
(179)           }
(180)   }
(181)
(182)   public String ejbFindByPrimaryKey(String name) throws FinderException {
(183)           Statement stmt = getStatement();
(184)           String s =
(185)                   new String("SELECT Name FROM PERSON WHERE Name='" + name + "';");
(186)           try {
(187)                   ResultSet rs = stmt.executeQuery(s);
(188)                   rs.first();
(189)                   return rs.getString("Name");
(190)           } catch (SQLException e) {
(191)                   e.printStackTrace();
(192)           }
(193)           return null; //never gets here
(194)   }
(195)}
```

Download des Beispiels

Zusätzlich enthält die Bean des Beispiels mit `getAge` eine zwar in der Remote-Schnittstelle veröffentlichte Operation, die keinen direkt abgespeicherten Wert liefert, sondern diesen dynamisch zur Ausführungszeit anhand der verfügbaren Daten berechnet.

Alle anderen in der Remote-Schnittstelle aufgeführten Operationen (etwa: `getStreet`, `setStreet`) modifizieren lediglich, den durch die Attribute repräsentierten Java-Objektzustand und greifen nicht direkt auf die Datenbank zu.

Innerhalb der Datenbankzugreifenden Methoden muß durch den Anwender die Abbildung der Java-Datentypen auf die des verwendeten Datenbankmanagementsystems erfolgen. Die mit dem Präfix `ejb` versehenden Methoden zeigen dies für die lesenden und schreibenden DB-Zugriffe. So kann die im Beispiel für `name` und `street` verwendete Java-Repräsentation `String` vergleichsweise leicht in den SQL-Typ `VARCHAR` abgebildet werden, sofern alle Methoden, die Datenbankinhalte schreiben, sicherstellen, daß nur zum Datenbankschema konforme Werte eingefügt werden. Die Beispielimplementierung zeigt dies examplarisch anhand der Methoden `ejbCreate` und `setStreet`.
Für programmiersprachliche Typen, die nicht direkt in DB-Typen abbildbar sind, muß im Falle der Bean Managed Persistence der Bean-Entwickler selbst Sorge für die adäquate Abbildung tragen. Das Beispiel illustriert dies anhand des Java-Datumstyps `GregorianCalendar`, der manuell in die durch das DBMS erwartete ISO 8601-konforme Darstellung zu überführen ist.

Einige der möglichen Interaktionen mit der Bean zeigt der Code des Clients aus Beispiel 74:

Beispiel 74: Client der auf eine Entity Bean zugreift

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.FinderException;
(4)import javax.ejb.RemoveException;
(5)import javax.naming.Context;
(6)import javax.naming.InitialContext;
(7)import javax.naming.NamingException;
(8)import javax.rmi.PortableRemoteObject;
(9)
(10)public class CallPersonBean {
(11)    public static void main(String[] args) {
(12)            try {
(13)                    Context initial = new InitialContext();
(14)                    Object objRef = initial.lookup("personBean");
(15)
(16)                    PersonHome home =
(17)                            (PersonHome) PortableRemoteObject.narrow(
(18)                                    objRef,
(19)                                    PersonHome.class);
(20)                    Person p1 = home.create(1911, 1, 1, "Alice", "streetA");
(21)                    Person p2 = home.create(1922, 2, 2, "Bob", "streetB");
(22)                    System.out.println("Alice's Age: " + p1.getAge());
(23)
(24)                    System.out.println("Alice's primary key: "+p1.getPrimaryKey());
(25)                    p1.remove();
(26)
(27)                    Person p3 = home.findByPrimaryKey("Bob");
(28)                    System.out.println(
(29)                            "Bob's modified street (before modification): "
(30)                                    + p3.getStreet());
```

```
(31)                    p3.setStreet("streetC");
(32)                    System.out.println(
(33)                            "Bob's modified street (after modification): "
(34)                                    + p3.getStreet());
(35)                    System.out.println("also the other reference: " + p2.getStreet());
(36)
(37)                    System.out.println("Are both references the same Java object: "+
(p2==p3));
(38)                    System.out.println("Are both references the same EJBObject (calls
isIdentical): "+p2.isIdentical(p3));
(39)
(40)            } catch (NamingException ne) {
(41)                    ne.printStackTrace();
(42)            } catch (RemoteException re) {
(43)                    re.printStackTrace();
(44)            } catch (CreateException ce) {
(45)                    ce.printStackTrace();
(46)            } catch (RemoveException e) {
(47)                    e.printStackTrace();
(48)            } catch (FinderException e) {
(49)                    e.printStackTrace();
(50)            }
(51)    }
(52)}
```

Download des Beispiels

Der Client ermittelt zunächst per JNDI eine Referenz auf die Bean, welche unter dem Namen *personBean* im Verzeichnisdienst registriert ist.
Die erhaltene generische Referenz wird durch Aufruf der statischen Methode `narrow` der Klasse `PortableRemoteObject` typsicher in eine Ausprägung der Home-Schnittstelle (`PersonHome`) konvertiert.
Der Aufruf der in dieser Schnittstelle durch den Anwender definierten `create`-Methode sorgt für die serverseitige Instanziierung der EJB, die als Ausprägung der Remote-Schnittstelle geliefert wird.
Tatsächlich wird nicht das EJB-Objekt selbst durch den Methodenaufruf retourniert, sondern lediglich ein netzwerktransparenter Verweis darauf, der jedoch clientseitig einer lokalen Objektreferenz gleichgestellt verwendet werden kann.
Ferner wird serverseitig zur Kommunikation mit der EJB ein Home-Objekt erzeugt, welchem eine Stellvertreterrolle für den anfragenden Client zukommt.
Der Aufruf der durch die EJB zur Verfügung gestellten Methode erfolgt identisch zu dem einer Lokalen.

So dient der Aufruf der Methode `create` zur Erzeugung von serverseitig instanziiert und transparent persistierten EJB-Objekten sowie den lokalen Java-(Stellvertreter-)Objekten für den Zugriff darauf.
Der Aufruf von `getAge` zeigt die Nutzung einer in der Remote-Schnittstelle veröffentlichten Zugriffsmethode. Mit `getPrimaryKey` wird die, in der durch die Remote-Schnittstelle erweiterten Schnittstelle `EJBObject` angesiedelte, Operation zur Ermittlung des Primärschlüsselwertes eines EJB-Objektes aufgerufen.
Die Methode `remove` stellt dagegen eine durch die Home-Schnittstelle definierte Operation dar. Durch den Aufruf dieser Methode auf dem durch `p1` referenzierten Objekt wird durch Ausführung der Beanmethode `ejbRemove` die die Bean serverseitig repräsentierenden Datenbankeinträge entfernt sowie der durch die Bean belegte Speicherbereich als frei markiert. Alle Versuche nach Aufruf dieser Methode auf der clientseitigen hauptspeicherrepräsentation Wertänderungen durchzuführen führen daher zu einem Fehler.
Die Ermittlung von Referenzen auf existierende EJB-Objekte erfolgt durch die in der Remote-Schnittstelle definierte Methode `findByPrimaryKey`. Der EJB-Container stellt sicher, daß verschiedene Referenzen auf dasselbe EJB-Objekt synchronisiert in die Datenbank abgebildet werden, so daß keine Inkonsistenzen entstehen.

Für den Betrieb einer Enterprise Java Bean ist neben den bisher betrachteten Schnittstellen-Komponenten und der Realisierung der Bean selbst auch ein als *Deployment Deskriptor* bezeichnetes XML-Konfigurationsfile notwendig, welches verschiedene Einstellungsdaten sowie die Schnittstellendaten enthält.
Beispiel 75 zeigt ein Beispiel hierfür:

Beispiel 75: Deployment Deskriptor der Entity Bean

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)
(3)<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
(4)
(5)<ejb-jar>
(6)  <display-name>Ejb1</display-name>
(7)  <enterprise-beans>
(8)    <entity>
(9)      <display-name>PersonBean</display-name>
(10)     <ejb-name>PersonBean</ejb-name>
(11)     <home>PersonHome</home>
(12)     <remote>Person</remote>
(13)     <ejb-class>PersonBean</ejb-class>
(14)     <persistence-type>Bean</persistence-type>
(15)     <prim-key-class>java.lang.String</prim-key-class>
(16)     <reentrant>False</reentrant>
(17)     <security-identity>
(18)       <description></description>
(19)       <use-caller-identity></use-caller-identity>
(20)     </security-identity>
(21)   </entity>
(22)  </enterprise-beans>
(23)  <assembly-descriptor>
(24)    <method-permission>
(25)      <unchecked />
(26)      <method>
(27)        <ejb-name>PersonBean</ejb-name>
(28)        <method-intf>Remote</method-intf>
(29)        <method-name>getAge</method-name>
(30)        <method-params />
(31)      </method>
(32)      <method>
(33)        <ejb-name>PersonBean</ejb-name>
(34)        <method-intf>Home</method-intf>
(35)        <method-name>remove</method-name>
(36)        <method-params>
(37)          <method-param>javax.ejb.Handle</method-param>
(38)        </method-params>
(39)      </method>
(40)      <method>
(41)        <ejb-name>PersonBean</ejb-name>
(42)        <method-intf>Home</method-intf>
(43)        <method-name>getHomeHandle</method-name>
(44)        <method-params />
(45)      </method>
(46)      <method>
(47)        <ejb-name>PersonBean</ejb-name>
(48)        <method-intf>Remote</method-intf>
(49)        <method-name>isIdentical</method-name>
(50)        <method-params>
(51)          <method-param>javax.ejb.EJBObject</method-param>
(52)        </method-params>
(53)      </method>
(54)      <method>
(55)        <ejb-name>PersonBean</ejb-name>
(56)        <method-intf>Home</method-intf>
(57)        <method-name>remove</method-name>
(58)        <method-params>
(59)          <method-param>java.lang.Object</method-param>
(60)        </method-params>
(61)      </method>
(62)      <method>
(63)        <ejb-name>PersonBean</ejb-name>
(64)        <method-intf>Remote</method-intf>
(65)        <method-name>getHandle</method-name>
(66)        <method-params />
(67)      </method>
(68)      <method>
(69)        <ejb-name>PersonBean</ejb-name>
(70)        <method-intf>Home</method-intf>
(71)        <method-name>findByPrimaryKey</method-name>
(72)        <method-params>
(73)          <method-param>java.lang.String</method-param>
(74)        </method-params>
(75)      </method>
```

```
(76)      <method>
(77)        <ejb-name>PersonBean</ejb-name>
(78)        <method-intf>Home</method-intf>
(79)        <method-name>getEJBMetaData</method-name>
(80)        <method-params />
(81)      </method>
(82)      <method>
(83)        <ejb-name>PersonBean</ejb-name>
(84)        <method-intf>Remote</method-intf>
(85)        <method-name>getPrimaryKey</method-name>
(86)        <method-params />
(87)      </method>
(88)      <method>
(89)        <ejb-name>PersonBean</ejb-name>
(90)        <method-intf>Remote</method-intf>
(91)        <method-name>remove</method-name>
(92)        <method-params />
(93)      </method>
(94)      <method>
(95)        <ejb-name>PersonBean</ejb-name>
(96)        <method-intf>Remote</method-intf>
(97)        <method-name>getEJBHome</method-name>
(98)        <method-params />
(99)      </method>
(100)   </method-permission>
(101)   <container-transaction>
(102)     <method>
(103)       <ejb-name>PersonBean</ejb-name>
(104)       <method-intf>Home</method-intf>
(105)       <method-name>create</method-name>
(106)       <method-params>
(107)         <method-param>int</method-param>
(108)         <method-param>int</method-param>
(109)         <method-param>int</method-param>
(110)         <method-param>java.lang.String</method-param>
(111)         <method-param>java.lang.String</method-param>
(112)       </method-params>
(113)     </method>
(114)     <trans-attribute>Never</trans-attribute>
(115)   </container-transaction>
(116)   <container-transaction>
(117)     <method>
(118)       <ejb-name>PersonBean</ejb-name>
(119)       <method-intf>Remote</method-intf>
(120)       <method-name>setStreet</method-name>
(121)       <method-params>
(122)         <method-param>java.lang.String</method-param>
(123)       </method-params>
(124)     </method>
(125)     <trans-attribute>Never</trans-attribute>
(126)   </container-transaction>
(127)   <container-transaction>
(128)     <method>
(129)       <ejb-name>PersonBean</ejb-name>
(130)       <method-intf>Remote</method-intf>
(131)       <method-name>getStreet</method-name>
(132)       <method-params />
(133)     </method>
(134)     <trans-attribute>Never</trans-attribute>
(135)   </container-transaction>
(136)   <container-transaction>
(137)     <method>
(138)       <ejb-name>PersonBean</ejb-name>
(139)       <method-intf>Home</method-intf>
(140)       <method-name>remove</method-name>
(141)       <method-params>
(142)         <method-param>javax.ejb.Handle</method-param>
(143)       </method-params>
(144)     </method>
(145)     <trans-attribute>Never</trans-attribute>
(146)   </container-transaction>
(147)   <container-transaction>
(148)     <method>
(149)       <ejb-name>PersonBean</ejb-name>
(150)       <method-intf>Home</method-intf>
(151)       <method-name>remove</method-name>
```

```
(152)        <method-params>
(153)          <method-param>java.lang.Object</method-param>
(154)        </method-params>
(155)      </method>
(156)      <trans-attribute>Never</trans-attribute>
(157)    </container-transaction>
(158)    <container-transaction>
(159)      <method>
(160)        <ejb-name>PersonBean</ejb-name>
(161)        <method-intf>Remote</method-intf>
(162)        <method-name>remove</method-name>
(163)        <method-params />
(164)      </method>
(165)      <trans-attribute>Never</trans-attribute>
(166)    </container-transaction>
(167)    <container-transaction>
(168)      <method>
(169)        <ejb-name>PersonBean</ejb-name>
(170)        <method-intf>Remote</method-intf>
(171)        <method-name>getAge</method-name>
(172)        <method-params />
(173)      </method>
(174)      <trans-attribute>Never</trans-attribute>
(175)    </container-transaction>
(176)    <container-transaction>
(177)      <method>
(178)        <ejb-name>PersonBean</ejb-name>
(179)        <method-intf>Home</method-intf>
(180)        <method-name>findByPrimaryKey</method-name>
(181)        <method-params>
(182)          <method-param>java.lang.String</method-param>
(183)        </method-params>
(184)      </method>
(185)      <trans-attribute>Never</trans-attribute>
(186)    </container-transaction>
(187)  </assembly-descriptor>
(188)</ejb-jar>
(189)
```

Download des Beispiels

3.3 Java Data Objects (JDO)

Grundidee

Hintergrund des Ansatzes der *Java Data Objects* (JDO) ist es, die bestehenden Schnittstellenmechanismen dahingehend weiterzuentwickeln, daß die Persistenz von Objekten und Objektgraphen für den Programmierer vollständig transparent durch Komponenten der Laufzeitumgebung zur Verfügung gestellt werden.
Gleichzeitig etabliert JDO eine Abstraktion der verschiedenen Speicherungsmöglichkeiten und erlaubt es beispielsweise die dateibasierte Ablage innerhalb des Programmes identisch zur Objektspeicherung in einem Datenbankmanagementsystem zu handhaben. Auf dieser Basis läßt sich im Bedarfsfalle den Persistenzdienstleister auszutauschen ohne Änderungen am Programmcode zu erfordern.
Plakativ wird der Ansatz daher, in Anlehnung an die Zielsetzung der Programmiersprache Java des *write once -- run anywhere*, als *write once -- store anywhere* charakterisiert.

Technik

Um die weitestgehend transparente Handhabung der Objektpersistenz zu gewährleisten bedient sich JDO eines Ansatzes der über das alleinige Angebot einer Programmierschnittstelle hinausreicht. Die Zielsetzung der möglichst einfach handzuhabenden Interaktion mit den generischen Persistenzmechanismen läßt sich zwar durch das Angebot von durch den Programmierer zu implementierenden Schnittstellen und Persistenzklassen erreichen, jedoch ist der Einsatz signifikant komplexer als der bestehenden Persistenzschnittstellen. Darüberhinaus konterkariert der Zwang bei der Programmerstellung vorgegebene Schnittstellen zu berücksichtigen die Zielsetzung weitestgehender Transparenz der angebotenen Speichermechanismen.

Daher führt JDO die Technik der sog. *Bytecodeanreicherung* (engl. *bytecode enhancing*) ein. Hierbei wird durch eine Programmkomponente vorübersetzer Bytecode so abgeändert, daß die notwendigen Persistenzanweisungen in den bereits erzeugten ausführbaren Bytecode eingewoben werden.

Die benötigte Übersetzerkomponente wird durch die jeweilige JDO-Implementierung zur Verfügung gestellt und muß durch den Programmierer im Bedarfsfalle lediglich geeignet parametrisiert werden.

Im Falle der Referenzimplementierung müssen daher alle Klassen, die Objekte ausprägen, welche persistiert werden sollen, mit dem Werkzeug entsprechend nachbearbeitet werden. Der notwendige Aufruf hat folgende Struktur: `java com.sun.jdori.enhancer.Main -d enhanced de/jeckle/jdotest/Employee.class de/jeckle/jdotest/Employee.jdo`.
Dieser Aufruf reichert die bereits übersetzte Klasse `Emplyoee` innerhalb der Pakethierarchie `de.jeckle.jdotest` um Persistenzdaten an und legt das Ergebnis innerhalb des Dateisystemkatalogs `de/jeckle/jdotest` ab. Zur Anreicherung wird die Konfigurationsdatei `Employee.jdo` herangezogen, die im selben Pfad abgelegt ist wie die Quellcodedatei.

Alternativ zu diesem Ansatz steht auch die Möglichkeit zur Verfügung die benötigten Anweisungen bereits im Quellcode vorzusehen um so dasselbe Resultat zu erzielen, welches durch den Anreicherungsprozeß erzeugt wird. Diese Vorgehensweise hat jedoch wegen der damit verbundenen Aufwände kaum praktische Bedeutung erlangt und wird daher im folgenden nicht vertieft betrachtet.

Die Beispiele dieses Kapitels basieren auf der kostenfrei verfübaren JDO-Referenzimplementierung von SUN. Diese beschränkt zwar die unterstützten Persistenzmechanismen auf ausschließlich dateibasierte Speicherung und sieht keine Ablage in Datenbankmanagmenetsystemen vor.
Konzeptionell und programmierseitig ist die Interaktion mit dieser Implementierung jedoch identisch zu kommerziell verfügbaren Lösungen und können daher ohne weiteres auf diese und damit beliebige Persistenzdienstleister übertragen werden.

Konfiguration des Persistenzdienstleisters

Die Abbildung der in der Programmiersprache formulierten Interaktionen auf den konkreten physischen Persistenzdienstleister erfolgt sinnvollerweise an einer für alle JDO-nutzenden Applikationen zugänglichen Stelle im Rahmen einer Property-Datei.
Die Inhalte dieser Datei unterscheiden naturgemäß bei den verschiedenen JDO-Herstellen und inhärent mit dem gewählten Persistenztyp. So benötigt die dateibasierte Objektablage offenkundig andere Festlegungen als der Zugriff auf ein relationales Datenbankmanagementsystem.
Beispiel 76 zeigt die notwendigen Einstellung zur Konfiguration der dateibasierten Speicherung mit der SUN-Referenzimplementierung. Dort wird mit der `PersistenceManagerFactoryClass` diejenige Klasse innerhalb des JDO-Rahmenwerkes benannt, welche dem Programmierer die Persistenzdienste zur Verfügung stellt. `ConnectionURL` bildet das Bindeglied der Abbildung auf die physische Datei und benennt daher den Speicherort aller persistierten Objekte. Die zusätzlichen Angaben dienen der Authentisierung und Zugriffssteuerung beim Zugriff auf die erstellte Datei.

Beispiel 76: Konfiguration einer JDO-Implementierung

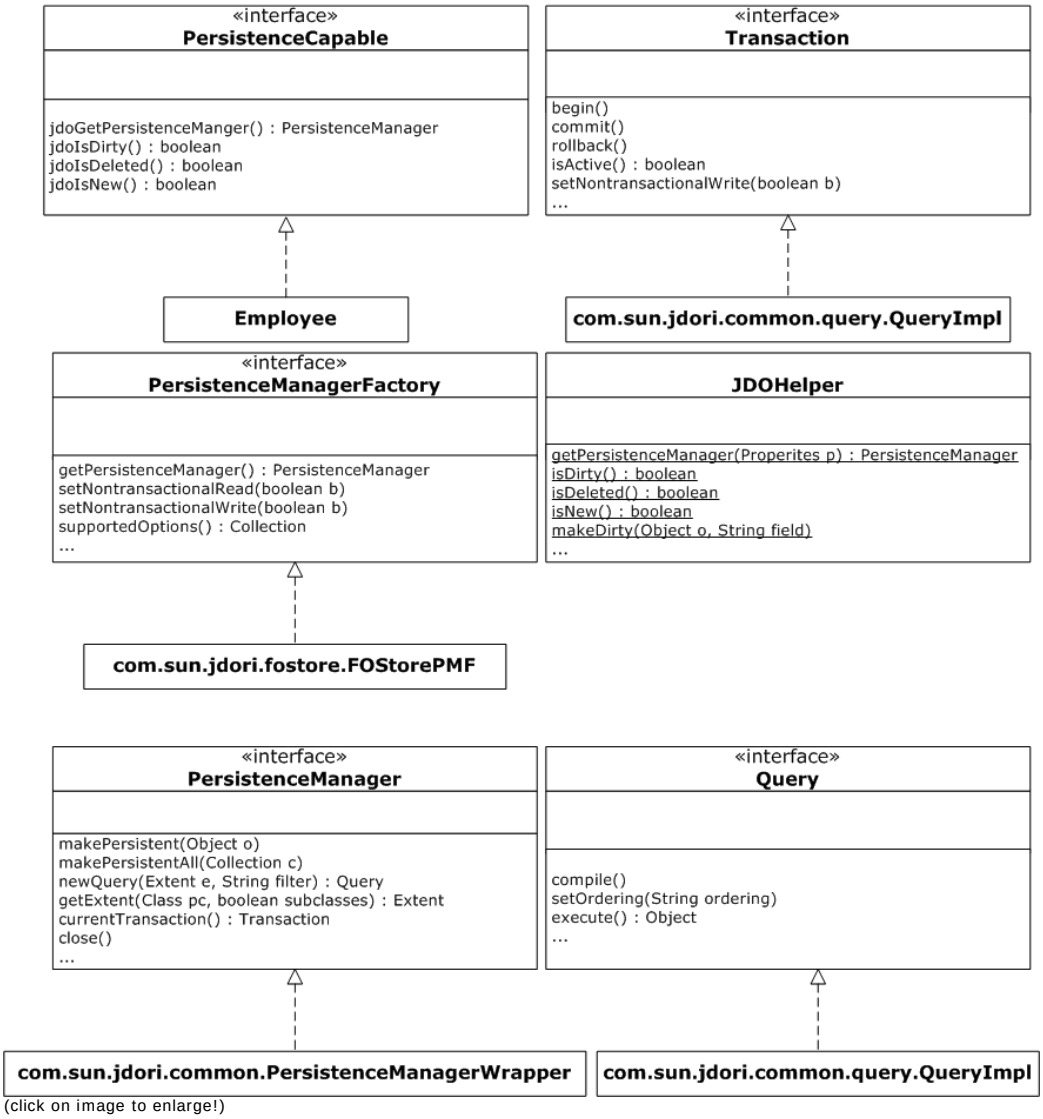
```
(1)javax.jdo.PersistenceManagerFactoryClass=com.sun.jdori.fostore.FOStorePMF
(2)javax.jdo.option.ConnectionURL=fostore:jdoriDB
(3)javax.jdo.option.ConnectionUserName=mario
(4)javax.jdo.option.ConnectionPassword=thePassword
```

Download des Beispiels

Struktur der JDO-API

Die JDO-API ist im Rahmen des Java Community Prozesses als Java-Schnittstellensammlung nebst zugehöriger Semantikdefinition spezifiziert. Die Implementierung der Schnittstellen erfolgt durch den Anbieter der jeweiligen JDO-Implementierung und erfolgt auf den jeweiligen Persistenztyp abgestimmt.
Abbildung 7 zeigt die grundlegenden Schnittstellen der JDO-API sowie die sie anbietenden Klassen der Referenzimplementierung.

Abbildung 7: *Grundlegende Struktur der JDO-API*



(click on image to enlarge!)

Die Schnittstelle `PersistenceCapable` bildet das Rückgrat der gesamten Persistenzbemühungen. Jede Klasse, deren Speicherung durch JDO verwaltet werden soll (in Beispiel die Klasse `Employee`) muß diese Schnittstelle zwingend implementieren.
Typischerweise erfolgt diese Implementierung jedoch nicht direkt durch den Applikationsprogrammierer, sondern wird im Rahmen der Bytecodeanreicherung nachträglich hinzugefügt.
Zur Interaktion mit Klassen, deren Implementierung der in `PersistenceCapable` deklarierten Methoden erst nach dem initialen Übersetzungsvorgang hinzugefügt werden kann der JDO-Anbieter die Hilfsklasse `JDOHelper` anbieten. Diese definiert verschiedene, ausschließlich als statisch deklarierte, Methoden um mit Objekten von Klassen zu operieren, als würden diese die Schnittstelle `PersistenceCapable` umsetzen, ohne deren Klassen zur tatsächlichen Schnittstellenimplementierung verpflichten.
Damit stellt `JDOHelper` die unabdingbare Voraussetzung zur Anwendungsentwicklung unter Verwendung der Bytecodeanreicherung dar, da diese erst nach dem Übersetzungsvorgang Implementierungen derjenigen Schnittstellen hinzufügt, die bereits im Code verwendeten wurden. Ferner bietet die Klasse die Möglichkeit den aktuellen Persistenzzustand eines JDO-verwalteten Objektes auszulesen.

Zur Erzeugung von Objekten, die später den Zugriff auf das physische Speichermedium regeln dienen die Umsetzungen der Schnittstelle `PersistenceManagerFactory`. Sie erlaubt die Parametrisierung und Verwaltung der Verbindung zum Persistenzmedium. Bereitgestellt wird die Implementierung, im Falle der Referenzimplementierung, durch die Klasse `com.sun.jdori.fostore.FOStorePMF`.
Die Verbindung zwischen Schnittstelle und tatsächlicher Implementierung wird im Rahmen der in Beispiel 76 gezeigten JDO-Konfiguration definiert. Zum Wechsel des Persistenzanbieters -- etwa von der durch die Referenzimplementierung angebotenen dateibasierten Speicherung auf eine datenbankgestützte Umsetzung -- genügt im die Abänderung dieses Eintrages in der Konfigurationsdatei.

Klassen, welche die Schnittstelle `PersistenceManagerFactory` implementieren, werden zur Erzeugung von sog. `PersistenceMangern` herangezogen. Umsetzungen dieser Schnittstelle (im Falle der Referenzimplementierung ist dies die Klasse `com.sun.jdori.common.PersistenceManagerWrapper`) dienen zur Interaktion mit der Persistenzveraltung innerhalb der JDO nutzenden Applikation. Alle Änderungen des Zustandes eines persistenten Objektes werden durch diese Klasse abgewickelt.

JDO wickelt sämtliche Zugriffe auf die persistenten Daten transaktionsgesichert ab. Dieser Mechanismus wird auf der abstrakten Ebene der API durch die Schnittstelle `Transaction` definiert und steht daher für alle Persistenzanbieter gleichermaßen zur Verfügung.

Die Schnittstelle definiert alle zur Transaktionssteuerung benötigten Operationen (darunter `begin`, `commit` und `rollback`) an.
Im Falle der Referenzimplementierung wird die Schnittstelle durch die Klasse `com.sun.jdori.common.query.QueryImpl` umgesetzt.

Zusätzlich sieht JDO eine abstrakte Möglichkeit zur Formulierung von Anfragen auf den verwalteten Datenbestand vor. Die notwendige Schnittstelle wird durch `Query` bereitgestellt.
Hierfür müssen die verschiedenen JDO-Implementierungen ebenfalls eigene Umsetzungen vorsehen.

Erzeugen eines persistenten Objektspeichers

Zur Erzeugung eines Objektspeichers ist bereits die Nutzung der Implementierungen der zentralen JDO-Schnittstellen sowie die der Transaktionssteuerung notwendig.
Das Beispiel zeigt die notwendigen Schritte zur Erzeugung eines persistenten Objektspeichers.
Zunächst lädt das Beispiel die Konfiguration aus der Eigenschaftsdatei des Beispiels 76. Anschließend wird durch die mit `true` belegte implementierungsspezifische Eigenschaft `com.sun.jdori.option.ConnectionCreate` festgelegt, daß im Rahmen des Verbindungsaufbaus auch notwendigenfalls der Objektspeicher neu erzeugt wird.
Die Interaktion mit JDO beginnt durch die Erzeugung eines `PersistenceManagerFactory` konformen Objektes durch den Aufruf `getPersistenceManagerFactory` unter Auswertung der zuvor geladenen und ergänzten Konfigurationseigenschaften.
Nach der Erzeugung des Factory-Objektes kann mittels diesem durch den Aufruf `getPersistenceManager` ein Objekt erzeugt werden, das die Interaktion mit dem Objektspeicher bereitstellt. Durch die Ermittlung des Persistenzmanagers wird gleichzeitig eine Verbindung zum Persistenzanbieter aufgebaut.
Ausgehend von diesem Verwaltungsobjekt kann durch Definition einer „leeren“ Transaktion -- d.h. einer Transaktion, die jenseits der Erzeugung des transaktionalen Kontexts und seines Abschlusses mit *committ*, keine Operationen definiert -- der Objektspeicher erzeugt werden.
Den Abschluß der Interaktion mit dem Objektspeicher bildet die Beendigung der Verbindung durch Ausführung der Methode `close` des Verbindungsobjektes.

Beispiel 77: Erzeugung eines persistenten Objektspeichers

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Properties;
(4)
(5)import javax.jdo.JDOHelper;
(6)import javax.jdo.PersistenceManager;
(7)import javax.jdo.PersistenceManagerFactory;
(8)import javax.jdo.Transaction;
(9)
(10)public class JDOCreateDB {
(11)    public static void main(String args[]) {
(12)            Properties props = new Properties();
(13)            try {
(14)                    InputStream is = ClassLoader.getSystemResourceAsStream("jdo.
properties");
(15)                    props.load(is);
(16)                    props.put("com.sun.jdori.option.ConnectionCreate","true");
(17)            } catch (IOException ioe) {
(18)                    System.out.println("Error loading properties");
(19)                    System.exit(1);
(20)            }
(21)            PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory
(props);
(22)            PersistenceManager pm = pmf.getPersistenceManager();
(23)
(24)            Transaction tx = pm.currentTransaction();
(25)            tx.begin();
(26)            tx.commit();
(27)            pm.close();
(28)    }
(29)}
```

Download des Beispiels

Parametrisierung der Persistenz

Grundsätzlich können Ausprägungen jeder beliebigen Javaklasse durch JDO persistiert werden, solange diese Klassen die Schnittstelle `PersistentCapable` explizit im Quellcode implementieren oder die benötigte Implementierung im Rahmen der Bytecodeanreicherung hinzugefügt wird.
Zur Steuerung des konkreten Persistenzverhaltens wird eine zusätzliche Konfigurationsdatei benötigt. Diese bedient sich der bekannten XML-Sytnax und definiert das Persistenzverhalten der durch JDO zu

verwaltenden Klasseninstanzen näher.
Beispiel 78 zeigt zunächst die zu persistierende Klasse Employee.

Beispiel 78: Zu persistierende Javaklasse

```
(1)package de.jeckle.jdotest;
(2)
(3)import java.util.HashSet;
(4)import java.util.Iterator;
(5)
(6)public class Employee {
(7)     private String name;
(8)     private String department;
(9)     private HashSet projects = new HashSet();
(10)
(11)    public String getName() {
(12)            return name;
(13)    }
(14)    public String getDepartment() {
(15)            return department;
(16)    }
(17)    public void setName(String name) {
(18)            this.name = name;
(19)    }
(20)    public void setDepartment(String department) {
(21)            this.department = department;
(22)    }
(23)    public void addProject(String project) {
(24)            projects.add(project);
(25)    }
(26)
(27)    public String toString() {
(28)            String result="Employee named "+name+" works in department "+department;
(29)            result+="\nworks in: ";
(30)            Iterator i = projects.iterator();
(31)            while (i.hasNext()) {
(32)                    result+=(String) i.next();
(33)                    result+=", ";
(34)            }
(35)            return (result);
(36)    }
(37)}
```

Download des Beispiels

Die Nutzung JDO-gestützter Objektpersistenz impliziert keinerlei Modifikationen oder Ergänzungen am Quellcode. Ebenso sind keinerlei Umsetzungskonventionen einzuhalten, die im Beispiel definierten get- und set-Methoden dienen lediglich der vereinfachten Interaktion.

Das Beispiel 79 illustriert eine Parameterdatei zur Definition des spezifischen Persistenzverhaltens von Objekten der Klasse Employee.

Beispiel 79: Parametrisierung der Objektpersistenz

```
(1)<?xml version="1.0"?>
(2)<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)<jdo>
(4)     <package name="de.jeckle.jdotest">
(5)             <class name="Employee">
(6)                     <field
(7)                             name="name"
(8)                             persistence-modifier="persistent"/>
(9)                     <field
(10)                            name="department"
(11)                            persistence-modifier="persistent"/>
(12)                    <field
(13)                            name="projects">
(14)                            <collection
(15)                                    element-type="java.lang.String"
(16)                                    embedded-element="true"/>
(17)                    </field>
(18)            </class>
(19)    </package>
(20)</jdo>
```

Download des Beispiels

Die XML-Datei definiert zunächst den Paket- und Klassennamen der zu persistierenden Klasse mittels des Attributs `name` der XML-Elemente `package` und `class`.
Innerhalb eines `class`-Elements kann für jedes Attribut der Javaklasse ein mit `field` benanntes Element zur näheren Charakterisierung des Speicherungsverhaltens angegeben werden.
Ein solches Element trägt zunächst im Attribut `name` den klassenweit eindeutigen Namen des Attributs und erlaubt die Festlegung des spezifischen Persistenzverhaltes mittels der Belegung des Attributs `persistence-modifier`. Ist dieses mit dem Wert `persistent` versehen, so wird ein so gekennzeichnetes Attribut durch JDO im Datenspeicher persistiert. Trägt das XML-Attribut den Wert none, so wird das Javaattribut bei der Abbildung in den JDO-Datenspeicher ignoriert.
Zusätzlich besteht die Möglichkeit durch die Belegung mit `transactional` die Zwischenspeicherung des Attributwertes während der Abarbeitung einer Transaktion zu erzwingen, um so eine spätere Wiederherstellung (nach einem Aufruf von `rollback`) zu gewährleisten. Jedoch werden Felder, die so gekennzeichnet sind, nicht persistent in den Datenspeicher übernommen, sondern stehen nur während der Programmlaufzeit zur Verfügung.
Fehlt diese Spezifikation zu einem Attribut in der XML-Datei, so wird vorgabegemäß die Belegung mit `persistent` angenommen, sofern es in der beherbergenden Javaklasse nicht als `static`, `transient` oder `final` ausgewiesen ist.

Attribute vom Typ einer Sammlungsklasse, wie sie durch die *Collection API* definiert werden müssen zusätzlich mit einem `collection`-Element, welches innerhalb des `field`-Elements plaziert ist, charakterisiert. Das `collection`-Element spezifiziert durch sein Attribut `element-type` den Typ der Elemente in der Sammlung festlegt. Zusätzlich kann durch das Boole'sche-Attribut `embedded-element` gesteuert werden, ob die Inhalte des Sammlungsobjektes zusammen mit dem die Sammlung referenzierenden Objekt persistiert werden sollen.

Das Beispiel legt für alle Attribute der Klasse `Employee` ihre persistente Speicherung fest (Belegung des XML-Attributs `persistence-modifier` für alle Attribute `persistent`); ebenso wird die in Objekten des Typs *Employee*, unter dem Namen `projects`, enthaltene Sammlungsinstanz einschließlich ihrer Inhaltsobjekte des Standard-API-Typs `String` dauerhaft abgespeichert.

Über diese Festlegungen hinaus gestattet das Parametrisierungsformat die Festlegung spezifischer Konsistenzsemantik in Gestalt der Auszeichnung eines *Primärschlüssels*. Dieses aus dem relationalen Modell bekannte Konstrukt fordert die Eindeutigkeit eines Attributs oder einer Kombination von Attributen über die gesamte Menge der Ausprägungen eines Typs.
Durch die Unterstützung als abstraktes JDO-Konstrukt steht dieses Konzept zur Konsistenzsicherung auch für Applikationen zur Verfügung, die sich nicht relationaler Datenbanken als Persistenzdienstleister bedienen.
Zur Realisierung des Primärschlüsselkonzeptes ist das als Schlüssel zu interpretierende Attribut in der XML-Beschreibung zusätzlich mit dem XML-Attribut `primary-key` zu versehen, welches den Wert `true` tragen muß. Zusätzlich ist innerhalb des Elements `class` diejenige Klasse anzugeben, welche das Attribut beherbergt, das als Schlüssel herangezogen werden soll.
80 zeigt die notwendigen Modifikationen an der Parameterdatei des Beispiels 79 um das Java-Attribut `name` als Primärschlüssel festzulegen. Die primärschlüsselanbietende Klasse ist in diesem Falle die Klasse `Employee` selbst, weshalb sich ihr Name auch im XML-Attribut `objectid-class` des `class`-Elements findet.

Beispiel 80: Parametrisierung der Objektpersistenz und Definition eines Primärschlüssels

```
(1)<?xml version="1.0"?>
(2)<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)<jdo>
(4)     <package name="de.jeckle.jdotest">
(5)             <class name="Employee"
(6)                        objectid-class="Employee">
(7)                     <field
(8)                             primary-key="true"
(9)                             name="name"
(10)                            persistence-modifier="persistent"/>
(11)                    <field
(12)                            name="department"
(13)                            persistence-modifier="persistent"/>
(14)                    <field
(15)                            name="projects">
(16)                            <collection
(17)                                    element-type="java.lang.String"
(18)                                    embedded-element="true"/>
(19)                    </field>
(20)            </class>
(21)    </package>
(22)</jdo>
```

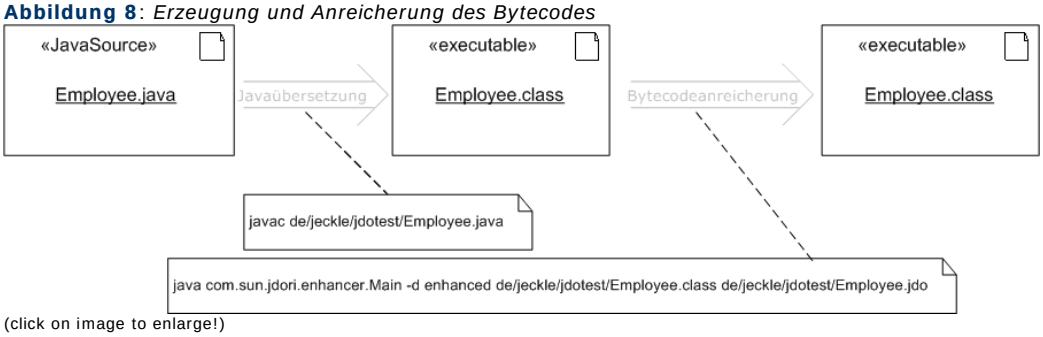
Download des Beispiels

Konsequenz der Einführung eines Primärschlüsselattributs ist die Überwachung der damit einhergehenden Konsistenzbedingungen durch das JDO-Laufzeitsystem. So führen Versuche zwei Objekte, die sich in der Belegung des als Primärschlüssel definierten Attributs nicht unterscheiden ebenso zu Fehlern wie schreibende Zugriffe auf dergestalt ausgezeichnete Attribute.

Anreicherung des Bytecodes

Voraussetzung der Persistenzverwaltung eines Objektes durch JDO ist die entsprechende Modifikation dieses Objektes, konkret die Implementierung der in `PersistenceCapable` festgelegten Operationen durch Methoden der objekterzeugenden Klasse.
Dies wird jedoch nur in Ausnahmefällen durch den Applikationsprogrammierer direkt vorgenommen. Häufigste Eionsatzform der JDO-API ist die Anwendung der Bytecodeanreicherung, welche die Implementierung der notwendigen Funktionalität automatisiert vornimmt und diese nach dem eigentlichen Übersetzungsvorgang in den erstellten Bytecode einbringt.
Abbildung 8 zeigt die daher notwendigen zwei Übersetzungsschritte.

Abbildung 8: *Erzeugung und Anreicherung des Bytecodes*



(click on image to enlarge!)

Die Illustration versammelt die zur Erzeugung und Anreicherung des Bytecodes der per JDO zu persistierenden Klasse `Employee` aus Beispiel 78. Zur Anreicherung des Bytecodes werden die in Beispiel 79 getroffenen Parametrisierungen herangezogen.
Zunächst wird der im Paket `de.jeckle.jdotest` abgelegte Quellcode `Employee.java` mit dem Javacompiler in (gewöhnlichen) Bytecode übersetzt.
Anschließend wird dieser vermöge des in der JDO-Referenzimplementierung vorhandenen Werkzeuges `Enhancer` um die Implementierung der in der Schnittstelle `PersistenceCapable` definierten Operationen angereichert. Hierzu wird dem Enhancer (bereitgestellt durch die Klasse `com.sun.jdori.enhancer.Main` zunächst das Zielverzeichnis des zu erzeugenden Bytecodes mittels des Parameters `d` übergeben. Naheliegernderweise kann der aus dem ursprünglichen Bytecode durch Erweiterung erzeugte nicht die Ausgangsdatei überschreiben, daher wird der angereicherte Bytecode im Verzeichnis `enhanced` gespeichert. Zusätzlich ist dem Enhancer der vollqualifizierte Name der anzureichernden Klasse sowie der vollqualifizierte Pfad der Parameterdatei (im Beispiel: `de/jeckle/jdotest/Employee.jdo`) zu übergeben. Diese muß im Falle des Einsatzes der Referenzimplementierung zwingend die Extension `jdo` besitzen.

Status JDO-verwalteter Objekte

Im Zusammenspiel zwischen transienter Objektverwaltung durch die Applikation im Hauptspeicher und persistenter Objektverwaltung durch JDO im Hintergrundspeicher werden verschiedene Status eines verwalteten Objekts unterschieden zwischen denen explizite Übergänge durch API-Aufrufe vorgegeben sind bzw. implizit durch Operationen auf den involvierten Objekten bestehen.
Im Detail werden folgende Status unterschieden:

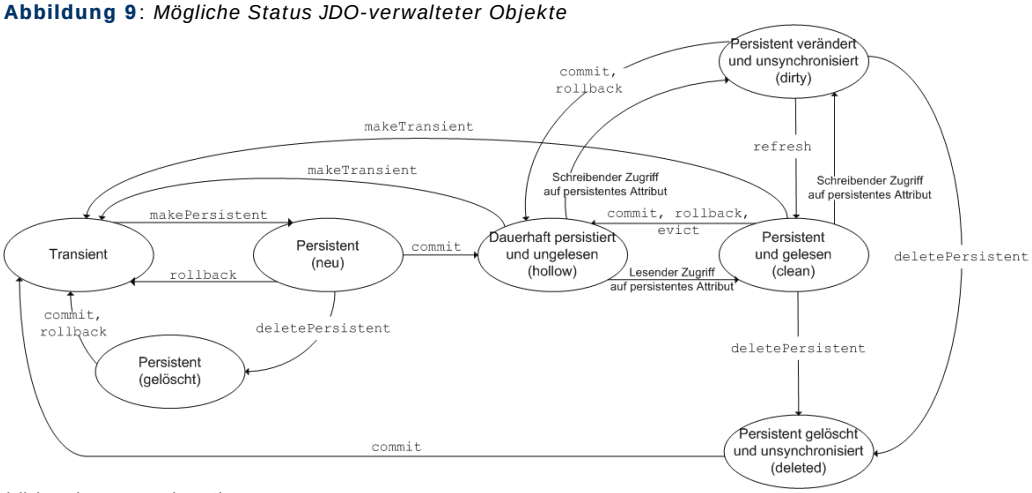
- **Transient**: Instantiierte Objekte im Hauptspeicher. Hierunter fallen alle noch nicht innerhalb von Transaktionen persistierten Objekte ebenso wie unangereicherte Javaobjekte und solche die ausschließlich über Attribut verfügen, die als in der XML-Parametrisierungsdatei als `transient` gekennzeichnet sind.
- **Persistent (neu)**: Objekte, die innerhalb einer laufenden (d.h. weder durch `commit` noch `rollback` abgeschlossenen) Transaktion erzeugt wurden.
  Endet eine Transaktion, etwa durch Programmabbruch, in diesem Zustand, so werden die Objekte mit diesem Status nicht dauerhaft gespeichert.
- **Persistent (gelöscht)**: Objekte, die in einer noch nicht abgeschossenen Transaktion persistiert und anschließend gelöscht wurden.
  Endet eine Transaktion, etwa durch Programmabbruch, in diesem Zustand, so werden die Objekte mit diesem Status nicht dauerhaft gespeichert, da sowohl der Persistierungs- als auch der anschließende Löschvorgang noch nicht durch `commit` bestätigt wurden.
- **Dauerhaft persistiert und ungelesen (hollow)**: Objekte, die durch Abschluß einer Transaktion mit `commit` dauerhaft gespeichert wurden und auf die noch kein Zugriff (weder lesend noch schreibend) erfolgte.
- **Persistent und gelesen (clean)**: Objekte, die persistent im Hintergrundspeicher abgelegt wurden und auf die bisher lediglich lesende Zugriffe erfolgten.
- **Persistent verändert und unsynchronisiert (dirty)**: Persistentes Objekt, dessen Inhalt im Rahmen einer noch nicht abgeschlossenen Transaktion verändert wurde.
  Wurden zwar Attributinhalte eines persistenten Objektes verändert, jedoch keine Wertänderungen

vorgenommen, d.h. in ein Attribut wird mit demselben Wert belegt, den es bereits enthält, dann steht es dem JDO-Implementierer frei diese Schreiboperation so zu implementieren, daß nicht der Zustand *dirty* eingenommen wird.
Zusätzlich kann jedes Objekt durch Aufruf der API-Methode `makeDirty` manuell in diesen Zustand versetzt werden.
Als Folge der Schreiboperation im Hauptspeicher differieren dessen Inhalte von denen des persistenten Objektspeichers.
Durch Aufruf der API-Methode `refresh` werden die Inhalte von Haupt- und Hintergrundspeicher synchronsiert, d.h. Inhalte des Hintergrundspeichers werden in den Hauptpeicher übernommen.

- **Persistent gelöscht und unsynchronisiert (deleted)**: Persistentes Objekt, das im Rahmen einer noch nicht abgeschlossenen Transaktion gelöscht wurde.
  Als Folge der Löschoperation im Hauptspeicher differieren dessen Inhalte von denen des persistenten Objektspeichers und alle Leseoperationen auf nicht-Primärschlüsselfelder führen zu Laufzeitfehlern.

Abbildung 9 zeigt die verschiedenen JDO-Status sowie die Ereignisse, die zu Zustandsübergängen führen, in der Übersicht.

Abbildung 9: *Mögliche Status JDO-verwalteter Objekte*



(click on image to enlarge!)

Speicherung von Objekten

Zur Speicherung von Objekten, deren Klassen durch den Bytecodeanreicherungsprozeß nachbearbeitet wurden, bietet die JDO-API die Aufrufe `makePersistent` und `makePersistentAll` an. Diese werden innerhalb eines Transaktionskontextes als Methoden eines `PersistenceManager`-Objekte ausgeführt.
Beispiel 81 zeigt die Speicherung von drei Objekten der Klasse `Emplyoee`, deren übersetzter Bytecode durch Anreicherung zur JDO-Kompatibilität modifiziert wurde.
Zunächst wird mit `empCol` vom Standardtyp `Vector` eine Sammlungsobjekt zur Aufnahme von Objektreferenzen definiert. Dieser Objektsammlung werden die Referenzen auf die erzeugten `Emplyoee`-Objekte (`emp1`, `emp2` und `emp3`) hinzugefügt.
Als Voraussetzung der Interaktion mit dem Objektspeicher muß zunächst eine Transaktion eröffnet werden. Hierzu muß zunächst durch Aufruf der Methode `currentTransaction` die der `PersistenceManager`-Instanz zugeordnete Transaktion ermittelt werden. Ausgehend vom gelieferten Ergebnisobjekt kann durch Ausführung der Methode `begin` eine neuer Transaktionskontext eröffnet werden.
Der Aufruf von `makePersistentAll` persistiert bei Übergabe der Objektsammlung alle in der Sammlung referenzierten Objekte. Alternativ können Einzelobjekte durch die Methode `makePersistent` in den Zustand dauerhafter Speicherung überführt werden.
Zur Übernahme in den Hintergrundspeicher muß der Transaktionskontext durch Aufruf von `commit` abgeschlossen werden. Der Aufruf von `rollback` würde stattdessen alle in der Transaktion vorgenommenen Änderungen verwerfen und auf den im Hintergrundspeicher verwalteten Datenzustand zurückgesetzt.

Beispiel 81: Speicherung von Objekten mit JDO

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Properties;
(4)import java.util.Vector;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDOStoreObj {
(13)    public static void main(String args[]) {
(14)            Properties props = new Properties();
```

```
(15)            try {
(16)                    InputStream is =
(17)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(18)                    props.load(is);
(19)            } catch (IOException ioe) {
(20)                    System.out.println("Error loading properties");
(21)                    System.exit(1);
(22)            }
(23)            PersistenceManagerFactory pmf =
(24)                    JDOHelper.getPersistenceManagerFactory(props);
(25)            PersistenceManager pm = pmf.getPersistenceManager();
(26)
(27)            Vector empCol = new Vector();
(28)
(29)            Employee emp1 = new Employee();
(30)            emp1.setName("Mario Jeckle");
(31)            emp1.setDepartment("D001");
(32)            emp1.addProject("P001");
(33)            emp1.addProject("P002");
(34)            empCol.add(emp1);
(35)
(36)            Employee emp2 = new Employee();
(37)            emp2.setName("John DoeX");
(38)            emp2.setDepartment("D003");
(39)            emp2.addProject("P001");
(40)            emp2.addProject("P042");
(41)            empCol.add(emp2);
(42)
(43)            Employee emp3 = new Employee();
(44)            emp3.setName("John Doe");
(45)            emp3.setDepartment("B042");
(46)            empCol.add(emp3);
(47)
(48)            Employee emp4 = new Employee();
(49)            emp4.setName("Barnie Bar");
(50)            emp4.setDepartment("B042");
(51)
(52)            pm.currentTransaction().begin();
(53)            pm.makePersistentAll(empCol);
(54)            pm.makePersistent(emp4);
(55)
(56)            pm.currentTransaction().commit();
(57)            pm.close();
(58)    }
(59)}
```

Download des Beispiels

Würde wie im Beispiel 80 gezeigt das Attribut `name` der Klasse `Employee` als Primärschlüssel definiert sein, so würde der Persistierungsversuch des durch `emp3` referenzierten Objektes einen Laufzeitfehler liefern, da mit `emp2` bereits ein Objekt mit derselben Belegung des Attributs `name` persistiert wurde.

Rücksetzen von Transaktionen

Treten während der Interaktion mit dem Persistenzspeicher, d.h. während eines noch nicht mit `commit` abgeschlossenen Transaktionskontextes Fehler auf, so können durch Aufruf der Methode `rollback` alle im aktuellen Kontext vorgenommen Änderungen auf den Stand vor Beginn der Transaktion zurückgesetzt werden.
Beispiel 82 zeigt das Verhalten der Methode `rollback` am Beispiel. Durch die Schreiboperation innerhalb der geöffneten Transaktion wird der Wert des Attributs `name` zwar verändert, jedoch durch Aufruf von `rollback` wieder auf den ursprünglichen Wert zurückgesetzt.

Beispiel 82: Transaktionen mit JDO

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDORollback {
(13)
(14)    public static void main(String args[]) {
(15)            Properties props = new Properties();
(16)            try {
(17)                    InputStream is =
(18)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(19)                    props.load(is);
(20)            } catch (IOException ioe) {
(21)                    System.out.println("Error loading properties");
(22)                    System.exit(1);
(23)            }
(24)            PersistenceManagerFactory pmf =
(25)                    JDOHelper.getPersistenceManagerFactory(props);
(26)            PersistenceManager pm = pmf.getPersistenceManager();
(27)
(28)            Employee e = new Employee();
(29)            e.setName("Marta Mayer");
(30)
(31)            pm.currentTransaction().begin();
(32)            pm.makePersistent(e);
(33)            pm.currentTransaction().commit();
(34)            displayPersistedObjects(pm);
(35)
(36)            System.out.println("Martha gets married and changes her name");
(37)
(38)            pm.currentTransaction().begin();
(39)            e.setName("Marta Smith");
(40)            pm.makePersistent(e);
(41)            displayPersistedObjects(pm);
(42)            System.out.println("Suppose and error happens now ...\nRolling back");
(43)            pm.currentTransaction().rollback();
(44)
(45)            displayPersistedObjects(pm);
(46)    }
(47)    private static void displayPersistedObjects(PersistenceManager pm) {
(48)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(49)            while (i.hasNext()) {
(50)                    System.out.println((Employee) i.next());
(51)            }
(52)    }
(53)}
```

Download des Beispiels

Die Ausführung des Beispiels liefert folgende Ausgabe:

```
Employee named Marta Mayer works in department null
works in projects:
Martha gets married and changes her name
Employee named Marta Smith works in department null
works in projects:
Suppose and error happens now ...
Rolling back
Employee named Marta Mayer works in department null
works in projects:
```

Schreiboperationen ohne Transaktionsschutz

Ist in bestimmten Anwendungsfällen die Arbeit ohne Transaktionsschutz -- und damit ohne die Möglichkeit der expliziten Rücksetzung von Änderungen mittels `rollback` oder der impliziten Rücksetzung nach einem

Systemausfall -- gewünscht, so kann dies durch Aktivierung der Schreibfunktionalität ohne Transaktionsschutz erreicht werden.
Hierzu muß de Methode `setNontransactionalWrite` mit dem Übergabeparameter `true` für eine Transaktion aufgerufen werden.
Das nachfolgende Beispiel zeigt als Modifikation von Beispiel 81 die persistente Übernahme einer Wertänderung ohne Transaktionsschutz.

Beispiel 83: Schreiboperation ohne Transaktionsschutz

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)public class JDONonTransact {
(12)    public static void main(String args[]) {
(13)            Properties props = new Properties();
(14)            try {
(15)                    InputStream is =
(16)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(17)                    props.load(is);
(18)            } catch (IOException ioe) {
(19)                    System.out.println("Error loading properties");
(20)                    System.exit(1);
(21)            }
(22)            PersistenceManagerFactory pmf =
(23)                    JDOHelper.getPersistenceManagerFactory(props);
(24)            PersistenceManager pm = pmf.getPersistenceManager();
(25)
(26)            Employee e = new Employee();
(27)            e.setName("Marta Mayer");
(28)            pm.currentTransaction().setNontransactionalWrite(true);
(29)
(30)            pm.currentTransaction().begin();
(31)            pm.makePersistent(e);
(32)            pm.currentTransaction().commit();
(33)            displayPersistedObjects(pm);
(34)
(35)            //martha gets married and changes her name
(36)
(37)            e.setName("Marta Smith");
(38)
(39)            displayPersistedObjects(pm);
(40)    }
(41)    private static void displayPersistedObjects(PersistenceManager pm) {
(42)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(43)            while (i.hasNext()) {
(44)                    System.out.println((Employee) i.next());
(45)            }
(46)    }
(47)}
```

Download des Beispiels

Traversierung des persistenten Objektbestandes

Zugriffe auf alle im Hintergrundspeicher verwalteten Objekte werden ebenfalls einheitlich durch Methoden der Implementierung der Schnittstelle `PersistenceManager` abgewickelt. Zur Traversierung des vollständigen Bestandes aller Instanzen einer Klasse bietet diese Schnittstelle die Operation `getExtent` an. Sie liefert alle Elemente der Extension (d.h. der Gesamtheit von Ausprägungen) einer gegebenen Klasse.
Beispiel 84 zeigt die Verwendung der Methode. Als Parameter wird diejenige Klasse übergeben, deren Ausprägungen zu ermitteln sind. Zusätzlich kann durch einen Boole'schen Schalter gesteuert werden, ob auch Subklassen der übergebenen Klasse retourniert werden sollen.
Der Aufruf liefert eine Sammlung von Objekten des Typs, welcher der Methode `getExtent` übergeben wurde.

Beispiel 84: Traversierung des Objektbestandes

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDOListObj {
(13)    public static void main(String args[]) {
(14)            Properties props = new Properties();
(15)            try {
(16)                    InputStream is =
(17)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(18)                    props.load(is);
(19)            } catch (IOException ioe) {
(20)                    System.out.println("Error loading properties");
(21)                    System.exit(1);
(22)            }
(23)            PersistenceManagerFactory pmf =
(24)                    JDOHelper.getPersistenceManagerFactory(props);
(25)            PersistenceManager pm = pmf.getPersistenceManager();
(26)
(27)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(28)            while (i.hasNext()) {
(29)                    System.out.println((Employee)i.next());
(30)            }
(31)    }
(32)}
```

Download des Beispiels

Anfragen an den persistenten Objektbestand

Als mächtige Alternative zur manuellen Traviersierung einer Objektextension spezifiziert JDO die Verwendung einer eigenen Anfragesprache auf Basis des Standards der *Object Query Language* (OQL) der Object Database Management Group (ODMG).
Diese -- als *JDO Object Query Language* (JDOQL) bezeichnete -- Anfragesprache ist direkt in die JDO-API integriert und wird über verschiedene Einzelmethoden genutzt. Aus diesem Grunde sind JDOQL-Anfragen nicht direkt mit den konsizsen SQL- oder OQL-Anfragen vergleichbar.
Beispiel 85 zeigt die Einbettung der Anfragesprache in die JDO-API.

Beispiel 85: Anfrage auf den persistenten Objektbestand mittels OQL

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Collection;
(4)import java.util.Iterator;
(5)import java.util.Properties;
(6)
(7)import javax.jdo.Extent;
(8)import javax.jdo.JDOHelper;
(9)import javax.jdo.PersistenceManager;
(10)import javax.jdo.PersistenceManagerFactory;
(11)import javax.jdo.Query;
(12)
(13)import de.jeckle.jdotest.Employee;
(14)public class JDOQuery {
(15)    public static void main(String args[]) {
(16)            Properties props = new Properties();
(17)            try {
(18)                    InputStream is =
(19)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(20)                    props.load(is);
(21)            } catch (IOException ioe) {
(22)                    System.out.println("Error loading properties");
(23)                    System.exit(1);
(24)            }
(25)            PersistenceManagerFactory pmf =
(26)                    JDOHelper.getPersistenceManagerFactory(props);
```

```
(27)            PersistenceManager pm = pmf.getPersistenceManager();
(28)
(29)            Extent ext = pm.getExtent(Employee.class, false);
(30)            String filter = "department == \"B042\"";
(31)            Query qry = pm.newQuery(ext, filter);
(32)            qry.setOrdering("name ascending");
(33)            qry.compile();
(34)            Collection c = (Collection) qry.execute();
(35)
(36)            Iterator i = c.iterator();
(37)            while (i.hasNext()) {
(38)                    System.out.println(i.next());
(39)            }
(40)    }
(41)
(42)}
```

Download des Beispiels

Das Beispiel illustriert eine Anfrage, die alle `Employee`-Objekte liefert, deren `department`-Attribut mit dem Wert `B042` belegt ist und liefert die nach dem Inhalt des Attributes `name` in aufsteigender Reihenfolge sortiert.
Hierzu wird zunächst die vollständige Extension der Klasse `Employee` ermittelt. Allerdings Extrahiert dieser Aufruf noch keine Werte aus dem persistenten Objektspeicher, sondern schafft nur die Grundlagen einer späteren manuellen Traversierung oder der Anfrage via JDOQL.
Zur Vorbereitung der tatsächlichen physischen Anfrage wird zunächst eine Zeichenkette geeignet belegt, um als Filterausdruck dienen zu können, der auf die vollständige Extension angewandt wird. Im Beispiel ist dieser Filterausdruck mit `department == \"B042\"` belegt. Aus Gründen der Zeichenkettenverarbeitung in Java muß hierzu der notwendige Einschluß des zu suchenden Wertes in Anführungszeichen geeignet maskiert werden.
Nach diesen Vorbereitungsschritten kann durch den Aufruf der durch das `PersistenceManager`-kompatible Objekt bereitgestellten Methode `newQuery` ein neues Anfrageobjekt (vom Typ `Query`) erzeugt werden.
Dieses Objekt erlaubt nach der gezeigten Festlegung des Anfrageumfanges die Parametrisierung der Anfrage. Das Beispiel illustriert dies am Aufruf der Methode `setOrdering`, die es erlaubt eine bestimmte Sortierreihenfolge der gelieferten Ergebnisse vorzugeben.
Zusätzlich kann durch die optionale Ausführung der Methode `compile` eine Prüfung der zusammengestellten Anfrage erfolgen, die zusätzlich auch interne implementierungsspezifische Optimierungen vornehmen kann.
Abschließend erfolgt die Ausführung der Anfrage durch Aufruf der Methode `execute`, welche die Anfrageergebnisse konform zur Standardschnittstelle `Collection` zurückliefert.

Löschen von Objekten

Zur Entfernung eines Objektes aus dem Objektspeicher stellt die Schnittstelle `PersistenceManager` die Methode `deletePersistent` zur Verfügung, welche ein einzelnes hauptspeicherresidentes Objekt aus dem persistenten Speicher löscht, bzw. mit `deletePersistentAll` eine Möglichkeit alle durch eine Sammlung referenzierten Objekte zu entfernen.
Da es sich hierbei um einen schreibenden Zugriff handelt, muß dieser in einen Transaktionskontext eingebettet werden oder explizit transaktionslos durchgeführt werden wie in Beispiel 83 gezeigt.
Beispiel 86 zeigt die Löschung unter Verwendung eines Transaktionskontextes.

Beispiel 86: Löschen eines Objektes aus dem persistenten Objektbestand

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Collection;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.Extent;
(7)import javax.jdo.JDOHelper;
(8)import javax.jdo.PersistenceManager;
(9)import javax.jdo.PersistenceManagerFactory;
(10)import javax.jdo.Query;
(11)
(12)import de.jeckle.jdotest.Employee;
(13)
(14)public class JDODeleteObj {
(15)    public static void main(String args[]) {
(16)            Properties props = new Properties();
(17)            try {
(18)                    InputStream is =
(19)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
```

```
(20)                    props.load(is);
(21)            } catch (IOException ioe) {
(22)                    System.out.println("Error loading properties");
(23)                    System.exit(1);
(24)            }
(25)            PersistenceManagerFactory pmf =
(26)                    JDOHelper.getPersistenceManagerFactory(props);
(27)            PersistenceManager pm = pmf.getPersistenceManager();
(28)
(29)            Extent ext = pm.getExtent(Employee.class, false);
(30)            String filter = "name == \"Marta Smith\"";
(31)            Query qry = pm.newQuery(ext, filter);
(32)            Collection c = (Collection) qry.execute();
(33)
(34)            pm.currentTransaction().begin();
(35)            pm.deletePersistentAll(c);
(36)            pm.currentTransaction().commit();
(37)            System.out.println("Object deleted");
(38)    }
(39)}
```

Download des Beispiels

Migration zu einem anderen Persistenzdienstleister

Der JDO-Ansatz tritt mit dem Versprechen auf vollständig sowohl unabhängig vom verwendeten Persistenzmedium (etwa: Datenbank, Dateisystem, etc.) als auch der eingesetzten JDO-Implementierung zu sein. Diese Zielsetzung wird nachfolgend auf Basis des im vorhergehenden diskutierten `Employee`-Beispiels untersucht. Hierzu wird die frei verfügbare JDO-Implementierung *TJDO* eingesetzt, welche verschiedene Datenbankmanagementsysteme zur Speicherung der Javaobjekte heranziehen kann. Im Beispiel wird das DBMS *MySQL* Persistierung der Applikationsobjekte genutzt.

Zur Portierung der bestehenden Applikation ist lediglich die Anpassung der JDO-Eigenschaften (Property-Datei) vorzunehmen, um den neuen Persistenzdienstleister sowie die verschiedenen DBMS-Spezifika zu berücksichtigen.
Beispiel 87 zeigt die neuen Inhalte.

Beispiel 87: Konfiguration der JDO-Implementierung TJDO

```
(1)javax.jdo.PersistenceManagerFactoryClass=com.triactive.jdo.PersistenceManagerFactoryImpl
(2)javax.jdo.option.ConnectionURL=jdbc:mysql://localhost/jdotest/
(3)javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
(4)javax.jdo.option.ConnectionUserName=mario
(5)javax.jdo.option.ConnectionPassword=thePassword
(6)com.triactive.jdo.autoCreateTables=true
```

Download des Beispiels

Zunächst werden die bereits in der Konfiguration der Referenzimplementierung durch Beispiel 76 genutzten Eigenschaften zur Identifikation derjenigen Klasse, welche die JDO-Schnittstelle `PersistenceManagerFactory` implementiert sowie zur Festlegung der Verbindungs-URL und des zu verwendenden Benutzernamens uns Passwortes an die neuen Gegebenheiten adaptiert. Konkret wird die durch TJDO bereitgestellte Klasse `com.triactive.jdo.PersistenceManagerFactoryImpl` als `PersistenceManagerFactory` konforme Implementierung sowie die Identifikation der zu verwendenden Datenbank nebst Benutzername und Anmeldekennwort bekanntgegeben.
Zusätzlich wird mit `com.triactive.jdo.autoCreateTables` eine implementierungsspezifische Eigenschaft mit `true` belegt, die TJDO veranlaßt im Bedarfsfalle benötigte Tabellenstrukturen automatisiert zu erzeugen.

Zusätzlich erfordert die verwendete JDO-Implementierung die Adaption der im Rahmen des Bytecodeanreicherungsprozesses herangezogenen Konfigurationsdatei (Beispiel 88). Auf diesem Wege wird dem Programmierer die Möglichkeit eröffnet die Abbildung auf relationale Tabellenstrukturen beeinflussen. In der Konsequenz erfordert der Wechsel der JDO-Implementierung die Wiederholung des Anreicherungslaufes für den Bytecode der zu persistierenden Klassen.

Beispiel 88: Parametrisierung der Objektpersistenz

```
(1)<?xml version="1.0"?>
(2)  <!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)  <jdo>
(4)      <package name="de.jeckle.jdotest">
(5)          <class name="Employee">
(6)              <field name="name">
(7)                  <extension vendor-name="triactive" key="length" value="max 32"/>
(8)              </field>
(9)              <field name="department">
(10)                  <extension vendor-name="triactive" key="length" value="max 32"/>
(11)              </field>
(12)          </class>
(13)      </package>
(14)  </jdo>
```

Download des Beispiels

Weitere Änderungen an den zu persistierenden Klassen oder den mit deren Objekten operierenden Applikationen ist nicht notwendig, alle Zugriffe werden nach den oben beschriebenen Änderungen transparent und ohne Neuübersetzung datenbankbasiert abgewickelt.

Vergleich der verschiedenen Persistenzansätze

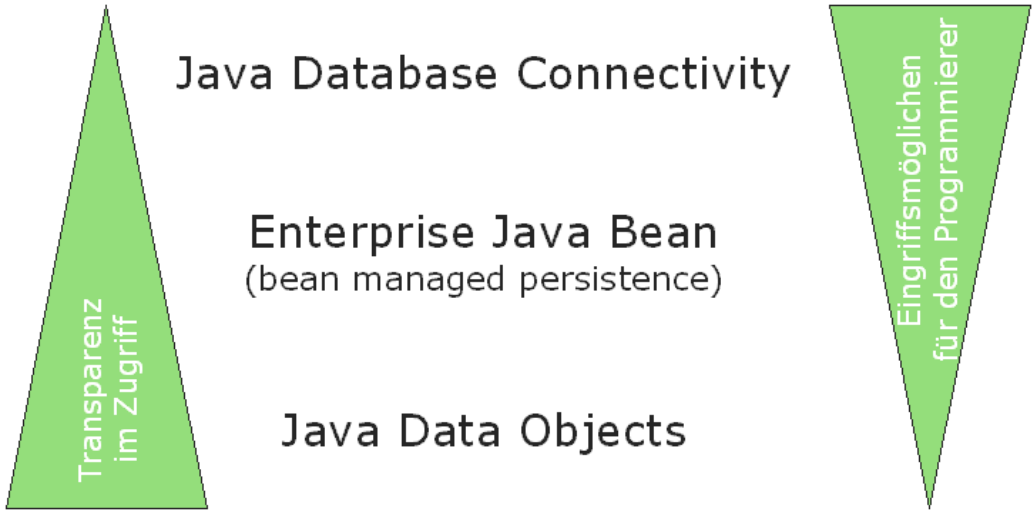
Abschließend seien die charakteristischen Eigenschaften der drei diskutierten Persistenzansätze JDBC, EJB und JDO kurz vergleichend nebeneinandergestellt.

Merkmal	JDBC	EJB	JDO
Transaktionsunterstützung	✓	✓	✓
Anfragemöglichkeit	✓	✓	✓
Standardisiertes API	✓	✓	✓
Standardanfragesprache	✓ SQL	✓ SQL/EJBQL	✓ JDOQL
Unterstützte Hintergrundspeicher	RDBMS	RDMBS Integrationsmiddleware	RDBMS, ORDBMS Integrationsmiddleware, Dateisystem, bel. andere
Transparenter Zugriff auf persistierte Daten	⊘	✓	✓
Berücksichtigung existierender relationaler Strukturen	✓	✓ bei bean managed persistence	✓ allerdings nicht im Standard vorgesehen

Die Tabelle zeigt klar, daß alle drei Persistenzmechanismen grundlegende Eigenschaften teilen, sich jedoch auch in zentralen Charakteristika unterscheiden.
Während sowohl JDBC als auch EJBs die direkte Verwendung von SQL-Anfragen gestatten bietet JDO mit JDOQL eine eigenständige Anfragesprache, die direkt in die Sprach-API eingebettet ist. Für EJBs existiert neben den in Kapitel 1.2 gezeigten Mechanismen auch die Möglichkeit der Verwendung der EJB-spezifischen Anfragesprache *EJBQL*, die jedoch hier nicht betrachtet wurde.
Hinsichtlich der jeweils unterstützten Hintergrundspeicherarchitekturen zur Realisierung der Persistenz treten jedoch deutliche Unterschiede zu Tage. So ist der Einsatz der JDBC-API auf relationale Datenquellen, bzw. Datenquellen die eine relationale Sicht anbieten, beschränkt. Innerhalb der EJB-Architektur können hingegen neben den -- hier diskutierten JDBC-basierten Mechanismen -- auch die Dienste einer Integrationsmiddleware zu Speicherung herangezogen werden und so eine gewisse Unabhängigkeit vom physischen Speichermedium erreicht werden. Einzig JDO bietet durch seine starke Abstraktion die Möglichkeit beliebige Persistenzdienstleister zu nutzen.
Zur effizienten Abwicklung dieses speicherformunabhängigen Zugriffs etabliert JDO notwendigerweise eine stark abstrahierte API, deren Funktionen keinerlei Rückschlüsse auf den verwendeten Persistenzmechanismus zulassen. Für EJB läßt sich dies prinzipiell auch realisieren, allerdings müssen für die Variante der bean managed persistence innerhalb der Entity Bean die Interaktionen mit dem Persistenzdienstleister expliziert werden, beispielsweise durch JDBC. Daher verhält sich dieser Ansatz intern ähnlich zur direkten Verwendung der JDBC-API, die inhärent jeden angebundenen

Persistenzmechanismus mit relationaler Zugriffssemantik belegt.
Aufgrund des vorherrschenden relationalen Speicherparadigmas kann die Einbindung bestehender Tabellenstrukturen in den API-Mechanismus gewünscht sein. Dies ist ausschließlich mit Ansätzen möglich, welche die anwenderdefinierte Strukturierung der Zugriffsausdrücke -- etwa durch die Verwendung von SQL -- gestatten. Dies ist ausschließlich für JDBC und EJB (sofern bean managed persistence verwendet wird) möglich; JDO sieht dies generell nicht vor.

Abbildung 10: *Vergleich zwischen den diskutierten Persistenztechniken*



(click on image to enlarge!)

Abschließend lassen sich die vorgestellten Schnittstellen hinsichtlich ihrer Möglichkeiten zur Bereitstellung eines transparenten Zugriffs auf den Hintergrundspeicher und der manuellen Eingriffsmöglichkeiten zur Kontrolle der Persistenz durch den Programmierer kategorisieren.
Prinzipiell läßt sich festhalten, daß diese Eigenschaftstypen konkurrierende Zielsetzungen darstellen. So bietet JDBC zweifelsohne die größten Möglichkeiten zum steuernden Eingriff durch den Programmierer, wobei dieser Ansatz in der Interaktion auch die größte Menge Wissen des Programmierers über die etablierten Speicherstrukturen erfordert. Daher realisiert JDBC generell die geringste Transparenz im Zugriff auf den Objektspeicher.
Auf der anderen Seite realisiert JDO die größtmögliche Transparenz im Objektzugriff, wobei dieser Freiheitsgrad zu generell zu Lasten der Eingriffsmöglichkeiten durch den Programmierer umgesetzt werden.

Web-Referenzen 10: Weiterführende Links
- TJDO -- eine freie JDO-Implementierung
- JDO @ SUN
- JDOCentral.com -- Die Anlaufstelle der JDO-Entwickler
- JDO-Spezifikation

---

4 Funktionsintegration

4.1 Java Message Service (JMS)

Einführung und Motivation

Der Austausch von Daten zwischen Rechnersystem ist so alt wie die Vernetzung zunächst separierter Rechnersysteme selbst. Während die Abwicklung des Austausches von Nachrichten zwischen den vernetzen Systemen anfänglich individuell für jede Plattform, d.h. spezifisch für jede Programmiersprache, das zugrundeliegende Betriebssystem und die zur physischen Datenübertragung eingesetzte Netzwerkinfrastruktur, gelöst werden mußte bildeten sich zur Steigerung der Interoperabilität und Portabilität bei gleichzeitiger Komplexitätsreduktion im Laufe der Zeit eigenständige generische Kommunikationskomonenten heraus, die zum Versand beliebiger Nachrichten eingesetzt werden können.

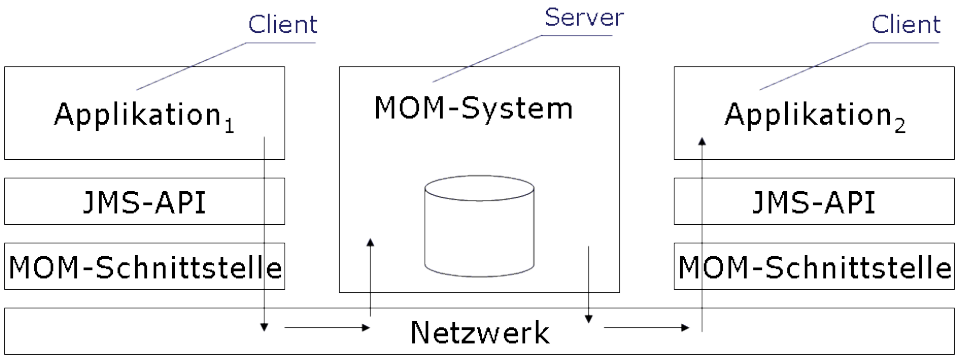
Ziel dieser -- als *message-oriented Middleware* bezeichneten -- Softwareinfrastruktur ist die deutliche Komplexitätsreduktion bei der Erstellung vernetzt kommunizierender Applikationen durch Einführung einer standardisierten, plattformübergreifend verfügbaren Lösung, welche dem Applikationsprogrammierer eine abstrahierte und vergleichsweise simple Programmierschnittstelle anbietet.

Zentrales Architekturmerkmal nachrichtenorientierter Middlewarekomponenten ist die gleichzeitige Betrachtung von ausschließlich zwei kommunizierenden Partnern, die als *client* und *server* bezeichnet werden, die in asynchroner Weise miteinander Nachrichten austauschen.

Im Rahmen der abgewickelten Interaktion versendet der Client dabei beliebige Daten an den Server, oder empängt diese von ihm. Der Server ist dabei die Message-orientierte Middleware, welche Sender und tatsächlichen Empfänger physische voneinander entkoppelt.
Die Begriffsbildung *asynchron* bezeichnet hierbei einen Kommunikationsstil, der die beteiligten Kommunikationspartner weder zeitlich noch prozedural koppelt.

Das Verhältnis zwischen Clients und Server, sowie das Zusammenspiel der verschiedenen physischen Architekturelemente ist in Abbildung 11 dargestellt.
Die Abbildung zeigt JMS-API, welche seitens der beteiligten Clients den Zugriff auf die systemspezifische Schnittstelle der verwendeten Middlware-Implementierung kapselt. Durch Pfeile ist der Ablauf eines Nachrichtenversandes von $\texttt{Applikation}_1$ an $\texttt{Applikation}_2$ dargestellt. Zunächst wird die Nachricht, durch Nutzung der involvierten Schnittstellen, an das MOM-System übertragen, welches es anschließend an den durch $\texttt{Applikation}_2$ vorgehaltenen Client ausliefert bzw. diesem zur Abholung bereitstellt.
Entlang des Pfades, welchen eine versandte Nachricht zurücklegt können auch mehrere MOM-System hintereinandergeschaltet auftreten. Diese agieren dann als Client bezüglich der in der Nachrichtenkette angrenzenden MOM-Systeme.

Abbildung 11: *Physische Architektur Nachrichten-orientierter Middleware*



(click on image to enlarge!)

Zusätzlich zeigt die Abbildung die typischerweise in MOM-Systemen präsente Datenbankkomponente, welche zur Sicherstellung der verläßlichen Nachrichtenübertragung dient. Im Normalfall wird dem versendenden Client erst die korrekte Übernahme der Nachricht in das MOM-System signalisiert, wenn sie persistent in der Datenbank abgelegt wurde. Im selben Sinne wird eine datenbankresidente Nachricht erst dann dauerhaft gelöscht, wenn sie dem Zielclient korrekt übermittelt wurde.
Durch diese Mimik entsteht, auch im Falle des Hintereinanderschaltens einzelner MOM-Systeme, eine verlässliche Nachrichtenstrecke, welche den Versand der Nachricht auch im Falle des Ausfalls einzelner Streckenabschnitte (d.h. MOM-Systeme) sicherstellen kann.

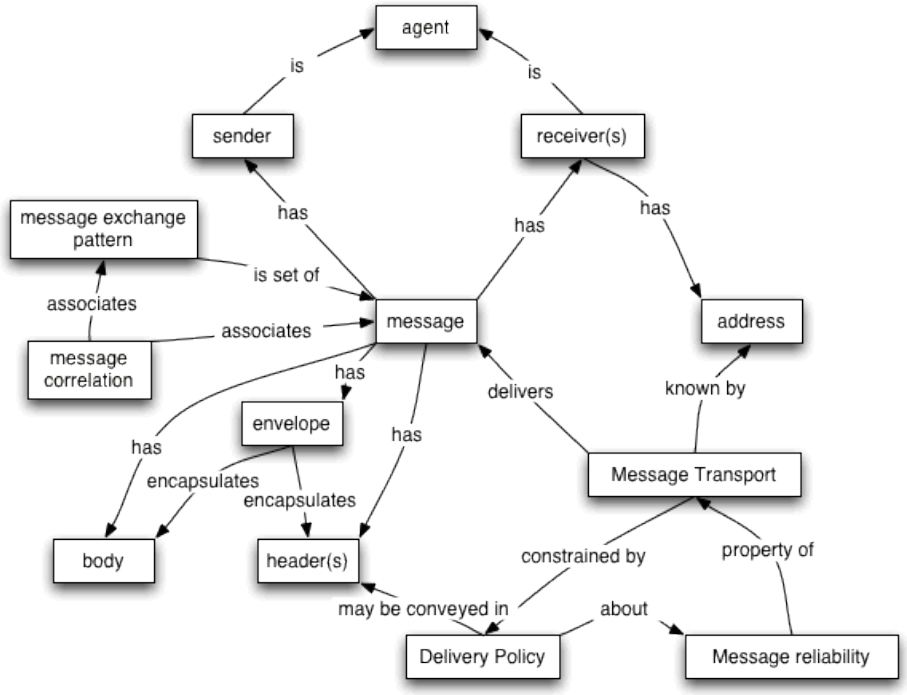
Abbildung Abbildung 12 zeigt, als Ausschnitt der Untersuchungen des W3C zu Web-Service-Architekturen, eine aktuelle Bestandsaufnahme der Komponenten einer Nachrichten-orientierten Architektur:

Tabelle 25: Elemente einer Nachrichten-orientierten-Middleware-Architektur

Architekturkomponente	Beschreibung
Address	Zielpunkt an den der Nachrichtentransportmechanismus die Nachricht ausliefert. Der syntaktische Aufbau einer Zieladresse kann vom verwendeten Transportprotokoll abhängen.
Delivery Policy	Art der Abwicklung des Nachrichtenversandes. Hierunter fallen qualitative Aspekte wie einzuhaltende Sicherheitsrestriktionen sowie die garantierte Auslieferung der Nachricht bis zu einem vorbestimmten Zeitpunkt.
Message	Logisches Konstrukt, das die auszuliefernde Nachricht repräsentiert. Sie besteht aus dem Nutzdateninhalt (Message Body) und optionalen beschreibenden Metadaten (Message Header).
Message Body	Der durch den Anwender festlegbare Nutzdateninhalt einer Nachricht.
Message Correlation	Einordnung einer Nachricht in einen Kontext. Korrelationsmechanismen stellen sicher, das zustandsbehaftete Protokolle -- die mehr als genau einer Kommunikation bedürfen -- über zustandslose Transportmechanismen korrekt abgewickelt werden können.
Message Envelope	Physische Implementierung der logischen Nachricht, daß Nachrichtenköpfe und Nutzdaten beinhaltet.

Message Header	Eine Datenstruktur, die deskriptive Metainformation über Teile einer Nachricht beinhaltet.
Message Exchange Pattern	Stil der Kommunikationsabwicklung hinsichtlich temoraler und logischer Abhängigkeiten. Synchrone und asynchrone Kommunikation sind zwei Beispiele konkreter Nachrichtenaustauschmuster. In gewissem Sinne stellen Nachrichtenaustauschmuster eine leichtgewichtige Korrelationsform dar.
Message Receiver	Agent, der die Nachricht entgegenimmt.
Message Reliability	Grad der Verläßlichkeit, daß eine Nachricht sicher zugestellt wird. Zwischen Sender und Empfänger muß diesbezüglich eine Übereinkunft hinsichtlich der Operationalität des Begriffs bestehen.
Message Sender	Agent, der in der Rolle als Client die Nachricht versendet.
Message Sequence	Geordnete Menge einzelner Nachrichten.
Message Transport	Technischer Mechanismus, der durch Agenten zum physischen Nachrichtenversand genutzt wird.

Abbildung 12: *Logische Architekturkomponenten einer Nachrichten-orientierter Middleware*



(click on image to enlarge!)

Innerhalb der Java-J2EE-API bietet die Programmierschnittstelle *Java Message Service* (JMS) einfachen Zugriff auf die Prinzipien Nachrichten-orientierter Kommunikation, die durch beliebige Implementierungen Nachrichten-orientierter-Middleware-Systeme zur Verfügung gestellt werden können.
Für die nachfolgenden Codebeispiele wurde die Implementierung der *Sun Java Message Queue*, welche als Bestandteil der J2EE-Referenzimplementierung ausgeliefert wird, genutzt. Die Beispiele sollten sich jedoch mit geringem Anpassungsaufwand auch auf andere Middlwaresysteme übertragen lassen.

Äquivalenz von synchroner und asynchroner Kommuniktion

TODO(Hier fehlt noch eine textuelle Erklärung)

Die JMS-API

Die JMS-API besteht aus einer Reihe von Java-Schnittstellenspezifikationen, die durch das verwendete Middlewaresystem implementiert werden.

Nachfolgend ist eine Übersicht der zentralen Schnittstellen und der durch sie angebotenen Funktionalität gegeben.

Tabelle 26: Schnittstellen der JMS-API

Schnittstelle	Beschreibung
Connection	Repräsentiert die Verbindung eines Clients zu einem Server über die Nachrichten versandt werden können.
ConnectionFactory	Kapselt verschiedene Konfigurationseinstellungen und wird clientseitig zur Erzeugung eines Connection-Objekts eingesetzt.
Destination	Dient der Repräsentation einer Transportmechanismen-spezifischen Adresse.
MessageProducer	Stellt dem Client die zum Nachrichtenversand nötigen Operationen zur Verfügung.
MessageConsumer	Stellt dem Client die zum Nachrichtenempfang benötigten Operationen zur Verfügung.>
Queue	Repräsentiert eine Nachrichtenstrecke.
Session	Liefert die logische Umgebung zum Versand und Empfang von Nachrichten.
TextMessage	Die zu versendene Nachricht. JMS gestatt spezifikationsgemäß ausschließlich die Übermittlung von Nachrichten, die durch Objekte der Standard-Klasse String repräsentiert werden.
Topic	Repräsentiert eine semantische Untermenge von Nachrichten in Form eines logischen Kanals.
TopicConnection	Repräsentiert einen logischen Kanal der durch Abonennten subscribiert werden und an alle abonnierenden Clients spontan Nachrichten versenden kann.
TopicConnectionFactory	Kapselt verschiedene Konfigurationseinstellungen und wird zur Erzeugung eines TopicConnection-Objekts eingesetzt.
Destination	Dient der Repräsentation einer Transportmechanismen-spezifischen Adresse.
MessageProducer	Stellt dem Client die zum Nachrichtenversand nötigen Operationen zur Verfügung.
MessageConsumer	Stellt dem Client die zum Nachrichtenempfang benötigten Operationen zur Verfügung.
TopicPublisher	Durch den Client eingesetzte Schnittstelle, die zum Versand von Nachrichten an abonnierte Kanäle dient.
TopicSubscriber	Durch den Client eingesetzte Schnittstelle, die zum Abruf von Nachrichten dient, welche an abonnierte Kanäle versandt wurden.
TopicSession	Schnittstelle, die Operationen zur Erzeugung von TopicPublisher-, TopicSubscriber- und TemporaryTopic-Objekten bereitstellt.

Synchroner Nachrichtenverkehr

Das Beispiel 89 zeigt die notwendigen Schritt zur Implementierung eines Clients zum synchronen Nachrichtenversand.
Zunächst ermitteln die Zeilen 29 bis 53 durch Nutzung des JNDI-Mechanismus die Referenz auf das zur Verbindungserzeugung zum Server benötige Objekt des Typs Connection Factory. Davon ausgehend wird in Zeile 56 ein Connection-Objekt erzeugt, welches die Verbindung zum Server verwaltet.
Der in Zeile 57 erfolgende Aufruf der API-Funktion createSession liefert eine Ausprägung des Typs Session, welches die logische Verbindung zum Server etabliert. Die in Zeile 57 des Beispiels genutzten Übergabeparameter erzeugen eine Session, die nicht transaktionsgestützt (erster Parameter) arbeitet und durch die als zweiten Übergabeparameter spezifierte Konstante AUTO_ACKNOWLEDGE dazu führt, daß die Verbindung (d.h. das Session-Objekt) den Nachrichtenempfang selbständig bestätigt, sobald der Client seine receive-Methode aufgerufen hat oder die Nachricht aus dem Nachrichtenspeicher entnommen wurde.
Ausgehend von der erzeugten Session erzeugt die Methode createProducer ein Objekt des Typs MessageProducer, welches zum Versand von Nachrichten an einen gegebene Adresse dient.
Zeile 60 erzeugt eine neue Nachricht des Typs TextMessage zum Versand. Dieser wird anschließend, durch die API-Funktion setText, textueller Nachrichteninhalt zugewiesen.
Der tatsächliche Sendevorgang findet (in Zeile 65) durch den Aufruf send statt. Die Implementierung des

Beispiels wiederholt diesen Sendevorgang (mit unterschiedlichen Nutzdateninhalten) so oft, wie per Aufrufparameter spezifiziert.
Zum Abschluß der Kommunikation versendet der Client eine Nachricht mit leerem Nutzdateninhalt (sie besteht nur aus verwaltender Headerinformation) an den Server. Diese dient als „Endemarkierung“ des kompletten Sendevorganges.

Beispiel 89: Versand einer Nachricht

```
(1)import java.util.Date;
(2)import javax.jms.Connection;
(3)import javax.jms.ConnectionFactory;
(4)import javax.jms.Destination;
(5)import javax.jms.JMSException;
(6)import javax.jms.MessageProducer;
(7)import javax.jms.Queue;
(8)import javax.jms.Session;
(9)import javax.jms.TextMessage;
(10)import javax.jms.Topic;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)public class SProducer {
(15)    public static void main(String[] args) {
(16)            final int NUM_MSGS;
(17)            if ((args.length < 2) || (args.length > 3)) {
(18)                    System.out.println("Program takes two or three arguments: "+
"<dest_name> <queue|topic> " + "[<number-of-messages>");
(19)                    System.exit(1);
(20)            }
(21)            String destName = new String(args[0]);
(22)            String destType = new String(args[1]);
(23)            System.out.println("Destination name is " + destName + ", type is " +
destType);
(24)            if (args.length == 3) {
(25)                    NUM_MSGS = (new Integer(args[2])).intValue();
(26)            } else {
(27)                    NUM_MSGS = 1;
(28)            }
(29)            Context jndiContext = null;
(30)            try {
(31)                    jndiContext = new InitialContext();
(32)            } catch (NamingException e) {
(33)                    System.out.println("Could not create JNDI " + "context: "
(34)                                    + e.toString());
(35)                    System.exit(1);
(36)            }
(37)            ConnectionFactory connectionFactory = null;
(38)            Destination dest = null;
(39)            try {
(40)                    connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/
QueueConnectionFactory");
(41)                    if (destType.equals("queue")) {
(42)                            dest = (Queue) jndiContext.lookup(destName);
(43)                    } else if (destType.equals("topic")) {
(44)                            dest = (Topic) jndiContext.lookup(destName);
(45)                    } else {
(46)                            throw new Exception("Invalid destination type" + "; must
be queue or topic");
(47)                    }
(48)            } catch (Exception e) {
(49)                    System.out.println("JNDI lookup failed: " + e.toString());
(50)                    e.printStackTrace();
(51)                    System.exit(1);
(52)            }
(53)            Connection connection = null;
(54)            MessageProducer producer = null;
(55)            try {
(56)                    connection = connectionFactory.createConnection();
(57)                    Session session = connection.createSession(false,
(58)                                    Session.AUTO_ACKNOWLEDGE);
(59)                    producer = session.createProducer(dest);
(60)                    TextMessage message = session.createTextMessage();
(61)                    for (int i = 0; i < NUM_MSGS; i++) {
(62)                            message.setText("This is message " + (i + 1) + " sent at "
(63)                                            + new Date());
(64)                            System.out.println("Sending message: " + message.getText
());
```

```
(65)                            producer.send(message);
(66)                    }
(67)                    producer.send(session.createMessage());
(68)            } catch (JMSException e) {
(69)                    System.out.println("Exception occurred: " + e.toString());
(70)            } finally {
(71)                    if (connection != null) {
(72)                            try {
(73)                                    connection.close();
(74)                            } catch (JMSException e) {
(75)                                    System.err.println("JMSException occured while
closing connection");
(76)                            }
(77)                    }
(78)            }
(79)    }
(80)}
```

Download des Beispiels
Download der Ergebnisdatei

Aufruf des Beispiels 89 mit: `j2ee14/bin/appclient -client SProducer.jar jms/Queue queue 5`

Beispiel 90 zeigt die Umsetzung eines nachrichtenempfangenden Clients.
Auch er ermittelt zunächst, ebenfalls unter Nutzung von JNDI, die relevanten Umgebungsdaten und erzeugt ausgehend von der `ConnectionFactory` eine Verbindung (Zeile 55) und eine Session zur Abwicklung des nachrichtenverkehrs.
Zur Kommunikation mit dem Client, welcher im vorhergehenden Beispiel zum Abschicken der Nachrichten diente, muß derjenige Client, der die Nachrichten entgegennimmt, an dieselbe Adresse gebunden werden, die bereits im Client als Nachrichtenendpunkt diente. Entsprechend wird dem Aufruf der JMS-API-Methode `createConsumer` dieselbe Endpunktidentifikation übergeben, die zuvor für den sendenden Client Verwendung fand. Der Aufruf der genannten Methode erzeugt ein Objekt des Typs `Consumer`, welches im Anschluß zum Empfang von Nachrichten, die an die übergebene Adresse versandt wurden, genutzt wird.
Die Empfangsbereitschaft seitens des Clients wird durch den Aufruf der Methode `start` (Zeile 58), die durch das `Connection`-Objekt bereitgestellt wird, hergestellt.
In Zeile 61 wird die Methode `receive` zum Empfang serverseitig vorliegender Nachrichten aufgerufen. Diese Methode ist synchron-blockierend, d.h. sie kehrt erst nach Empfang einer Nachricht zurück. Stellt die Middlewarekomponente aktuell keine solche zur Verfügung, so wird der methodenausführende Thread so lange blockiert, bis eine Nachricht vorliegt.
Konnte eine Nachricht empfangen werden, so wird in Zeile 64 durch `getText` der übermittelte textuelle Inhalt extrahiert.

Beispiel 90: Empfang einer Nachricht

```
(1)import javax.jms.Connection;
(2)import javax.jms.ConnectionFactory;
(3)import javax.jms.Destination;
(4)import javax.jms.JMSException;
(5)import javax.jms.Message;
(6)import javax.jms.MessageConsumer;
(7)import javax.jms.Queue;
(8)import javax.jms.Session;
(9)import javax.jms.TextMessage;
(10)import javax.jms.Topic;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)public class SConsumer {
(15)    public static void main(String[] args) {
(16)            String destName = null;
(17)            String destType = null;
(18)            Context jndiContext = null;
(19)            ConnectionFactory connectionFactory = null;
(20)            Connection connection = null;
(21)            Session session = null;
(22)            Destination dest = null;
(23)            MessageConsumer consumer = null;
(24)            TextMessage message = null;
(25)            if (args.length != 2) {
(26)                    System.out.println("Program takes two arguments: " + "<dest_name>
<queue|topic>");
(27)                    System.exit(1);
(28)            }
(29)            destName = new String(args[0]);
(30)            destType = new String(args[1]);
```

```
(31)            System.out.println("Destination name is " + destName + ", type is "
(32)                            + destType);
(33)            try {
(34)                    jndiContext = new InitialContext();
(35)            } catch (NamingException e) {
(36)                    System.out.println("Could not create JNDI " + "context: "
(37)                                    + e.toString());
(38)                    System.exit(1);
(39)            }
(40)            try {
(41)                    connectionFactory = (ConnectionFactory) jndiContext
(42)                                    .lookup("jms/QueueConnectionFactory");
(43)                    if (destType.equals("queue")) {
(44)                            dest = (Queue) jndiContext.lookup(destName);
(45)                    } else if (destType.equals("topic")) {
(46)                            dest = (Topic) jndiContext.lookup(destName);
(47)                    } else {
(48)                            throw new Exception("Invalid destination type" + "; must
be queue or topic");
(49)                    }
(50)            } catch (Exception e) {
(51)                    System.out.println("JNDI lookup failed: " + e.toString());
(52)                    System.exit(1);
(53)            }
(54)            try {
(55)                    connection = connectionFactory.createConnection();
(56)                    session = connection.createSession(false, Session.
AUTO_ACKNOWLEDGE);
(57)                    consumer = session.createConsumer(dest);
(58)                    connection.start();
(59)                    Message m;
(60)                    while (true) {
(61)                            m = consumer.receive();
(62)                            if (m instanceof TextMessage) {
(63)                                    message = (TextMessage) m;
(64)                                    System.out.println("Reading message: " + message.
getText());
(65)                            }
(66)                    }
(67)            } catch (JMSException e) {
(68)                    System.out.println("Exception occurred: " + e.toString());
(69)            } finally {
(70)                    if (connection != null) {
(71)                            try {
(72)                                    connection.close();
(73)                            } catch (JMSException e) {
(74)                                    System.err.println("JMSException occured while
closing connection");
(75)                            }
(76)                    }
(77)            }
(78)    }
(79)}
```

Download des Beispiels
Download der Ergebnisdatei

Aufruf des Beispiels 90 mit: `j2ee14/bin/appclient -client SProducer.jar jms/Queue queue 5`

Asynchroner Nachrichtenverkehr

TODO(Hier fehlt noch eine textuelle Erklärung)

Aufruf mit: `j2ee14/bin/appclient -client PSProducer.jar`

Beispiel 91: Versand einer Nachricht

```
(1)import javax.jms.DeliveryMode;
(2)import javax.jms.JMSException;
(3)import javax.jms.Message;
(4)import javax.jms.Session;
(5)import javax.jms.TextMessage;
(6)import javax.jms.Topic;
(7)import javax.jms.TopicConnection;
(8)import javax.jms.TopicConnectionFactory;
(9)import javax.jms.TopicPublisher;
(10)import javax.jms.TopicSession;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)
(15)public class PSProducer {
(16)    public static void main(String[] args) {
(17)            Topic t = null;
(18)            TopicConnectionFactory tcf = null;
(19)            TopicConnection tc = null;
(20)
(21)            try {
(22)                    Context jndiContext = new InitialContext();
(23)                    tcf = (TopicConnectionFactory) jndiContext.lookup("jms/
TopicConnectionFactory");
(24)            } catch (NamingException ne) {
(25)                    System.out.println(ne+ " does not exist");
(26)                    System.exit(1);
(27)            }
(28)
(29)            try {
(30)                    tc = tcf.createTopicConnection();
(31)                    TopicSession ts = tc.createTopicSession(false, Session.
AUTO_ACKNOWLEDGE);
(32)                    t = ts.createTopic("StockQuote");
(33)
(34)                    TopicPublisher tp = ts.createPublisher(t);
(35)
(36)                    TextMessage msg1 = ts.createTextMessage();
(37)                    TextMessage msg2 = ts.createTextMessage();
(38)
(39)                    msg1.setText("Quote");
(40)                    msg1.setStringProperty("Company","DCX");
(41)                    msg1.setDoubleProperty("USD",41.67);
(42)                    msg1.setDoubleProperty("EUR",34.99);
(43)                    msg1.setStringProperty("Date","2004-05-23");
(44)
(45)                    tp.publish(msg1, DeliveryMode.PERSISTENT, Message.
DEFAULT_PRIORITY, 7*24*3600*1000L);
(46)
(47)                    try {
(48)                            Thread.sleep(30000);
(49)                    } catch(InterruptedException ie) {
(50)                            System.err.println("InterruptedException caught");
(51)                    }
(52)
(53)                    msg2.setText("Quote");
(54)                    msg2.setStringProperty("Company","AZ");
(55)                    msg2.setDoubleProperty("USD",10.04);
(56)                    msg2.setDoubleProperty("EUR",9.30);
(57)                    msg2.setStringProperty("Date","2004-05-23");
(58)
(59)                    tp.publish(msg2, DeliveryMode.PERSISTENT, Message.
DEFAULT_PRIORITY, 7*24*3600*1000L);
(60)            } catch (JMSException e) {
(61)                    System.err.println("JMSException caught");
(62)                    e.printStackTrace();
(63)            }
(64)    }
(65)}
```

Download des Beispiels

TODO(Hier fehlt noch eine textuelle Erklärung)

Aufruf mit: `j2ee14/bin/appclient -client PSConsumer.jar`

Beispiel 92: Empfang einer Nachricht

```
(1)import java.util.Date;
(2)
(3)import javax.jms.JMSException;
(4)import javax.jms.Message;
(5)import javax.jms.MessageListener;
(6)import javax.jms.Session;
(7)import javax.jms.TextMessage;
(8)import javax.jms.Topic;
(9)import javax.jms.TopicConnection;
(10)import javax.jms.TopicConnectionFactory;
(11)import javax.jms.TopicSession;
(12)import javax.jms.TopicSubscriber;
(13)import javax.naming.Context;
(14)import javax.naming.InitialContext;
(15)import javax.naming.NamingException;
(16)public class PSConsumer {
(17)    public static void main(String[] args) {
(18)            Topic t = null;
(19)            TopicConnectionFactory tcf = null;
(20)            TopicConnection tc = null;
(21)
(22)            try {
(23)                    Context jndiContext = new InitialContext();
(24)                    tcf = (TopicConnectionFactory) jndiContext.lookup("jms/
TopicConnectionFactory");
(25)            } catch (NamingException ne) {
(26)                    System.err.println(ne+" does not exist");
(27)                    System.exit(1);
(28)            }
(29)
(30)            try {
(31)                    tc = tcf.createTopicConnection();
(32)                    tc.setClientID("MariosClient");
(33)                    TopicSession ts = tc.createTopicSession(false, Session.
AUTO_ACKNOWLEDGE);
(34)                    t = ts.createTopic("StockQuote");
(35)
(36)                    TopicSubscriber tSubDCX=ts.createDurableSubscriber(t,"mySub",
"Company LIKE '%DCX'",false);
(37)                    TopicSubscriber tSubAll=ts.createSubscriber(t);
(38)
(39)                    tSubDCX.setMessageListener( new MyTopicListener("DCX") );
(40)                    tSubAll.setMessageListener( new MyTopicListener("All"));
(41)
(42)                    tc.start();
(43)
(44)                    for (;;){
(45)                            System.out.println("("+new Date()+") waiting for
messages...");
(46)                            try {
(47)                                    Thread.sleep(1000);
(48)                            } catch(InterruptedException ie) {
(49)                                    System.err.println("InterruptedException caught");
(50)                            }
(51)                    }
(52)
(53)            } catch (JMSException e) {
(54)                    System.err.println("JMSException caught");
(55)                    e.printStackTrace();
(56)            }
(57)    }
(58)}
(59)
(60)class MyTopicListener implements MessageListener {
(61)    private String title;
(62)
(63)    public MyTopicListener(String title){
(64)            this.title = title;
(65)            System.out.println("Message listener "+title+" created");
(66)    }
(67)
(68)    public void onMessage(Message msg) {
(69)            try {
```

```
(70)            TextMessage tm = (TextMessage) msg;
(71)            System.out.println("("+this.title+") Message received: "+tm.getText());
(72)            System.out.println("Company="+tm.getStringProperty("Company"));
(73)            System.out.println("USD="+tm.getDoubleProperty("USD"));
(74)            System.out.println("EUR="+tm.getStringProperty("EUR"));
(75)            System.out.println("--------------------------");
(76)            } catch (JMSException je) {
(77)                    System.err.println("JMSException caught");
(78)                    je.printStackTrace();
(79)            }
(80)    }
(81)
(82)}
```

Download des Beispiels

Message-driven Beans

Die ursprünglich eigenständig entwickelte JMS-API ist inzwischen in die *Java 2 Enterprise Architecture* integriert worden. Implementierungen dieser Architektur können daher als MOM-System eingesetzt werden.

Darüber hinaus bildet die JMS-Funktionalität der Basis der *Message-driven Beans*. Diese Spielart der Entity Beans kann durch Empfang einer per JMS versandten Nachricht angesprochen werden. Voraussetzung hierzu ist die Implementierung der Operation `onMessage` der Schnittstelle `MessageListener` sowie die Implementierung der Schnittstelle `MessageDrivenBean`.

Das nachfolgende Beipiel zeigt die Reforumulierung des Beispiels 92 als Message-driven Bean.

Beispiel 93: Empfang einer Nachricht durch eine Message-driven Bean

```
(1)import javax.ejb.CreateException;
(2)import javax.ejb.MessageDrivenBean;
(3)import javax.ejb.MessageDrivenContext;
(4)import javax.jms.JMSException;
(5)import javax.jms.Message;
(6)import javax.jms.MessageListener;
(7)import javax.jms.TextMessage;
(8)
(9)public class MDBeanExample implements MessageDrivenBean, MessageListener{
(10)    private MessageDrivenContext ctx;
(11)    private TextMessage tm;
(12)
(13)    public void setMessageDrivenContext(MessageDrivenContext ctx) {
(14)            this.ctx = ctx;
(15)    }
(16)
(17)    public void ejbCreate() throws CreateException {
(18)            System.err.println("Message-driven Bean created");
(19)    }
(20)
(21)    public void ejbRemove() {
(22)            //does nothing
(23)    }
(24)
(25)    public void onMessage(Message msg)  {
(26)            if (msg instanceof TextMessage) {
(27)                    this.tm = (TextMessage) msg;
(28)                    try {
(29)                            System.out.println("Message received: "+this.tm.getText());
(30)                            System.out.println("Company="+this.tm.getStringProperty
("Company"));
(31)                            System.out.println("USD="+this.tm.getDoubleProperty
("USD"));
(32)                            System.out.println("EUR="+this.tm.getStringProperty
("EUR"));
(33)                            System.out.println("--------------------------");
(34)                    } catch (JMSException je) {
(35)                            System.err.println("JMSException caught");
(36)                            je.printStackTrace();
(37)                    }
(38)            }
(39)    }
(40)}
```

Download des Beispiels

Web-Referenzen 11: Weiterführende Links
- Introduction to Message-oriented Middleware
- JMS FAQ

4.2 Remote Method Invocation (RMI)

Grundidee

Der Java-spezifische Ansatz der *Remote Method Invocation* (RMI) zielt darauf einen lokationstransparenten Aufruf innerhalb von (verteilten) Java-Anwendungen zu ermöglichen.
RMI stellt hierbei die Einzelobjekte einer Applikation in den Vordergrund und gestattete den Aufruf von Methoden in derselben Art und Weise unabhängig davon ob sich das Objekt auf der aufrufenden Maschine selbst befindet oder an einer entfernten Lokation zur Verfügung steht.
Damit stellt sich RMI in die Tradition des objektorientierten Verteilungsmechanismus CORBA, ohne allerdings dessen programmiersprachenunabhängige Zielsetzung zu verfolgen. Gleichzeitig wurde das von DCE übernommene Serialierungsformat für hauptspeicherresidente Inhalte auf die Objektübertragung mittels eines Netzprotkolls ausgedehnt.
Die Grundprimitive des Ansatzes ist daher der entfernte Methodenaufruf (engl. *Remote Procedure Call*, RPC) welcher Kraft der lokalisationstransparenten Abstraktion RMIs dem Programmierer dieselbe Aufrufschnittstelle für entfernte Methoden bietet, die bereits für die Interaktion mit lokalen Objekten zum Einsatz kommt.
Zusätzlich erweitert RMI bestimmte zunächst lokal angebotene Dienstleistungen, wie etwa automatische Speicherverwaltung (*Garbage Collection*) für die verteilte Anwendung mit der Zielsetzung die Verwendung entfernter Objekte der lokal vorliegender gleichzustellen (Lokalisationstransparenz).
Zusammenfassend formuliert die RMI-Spezifikation daher folgende Ziele:

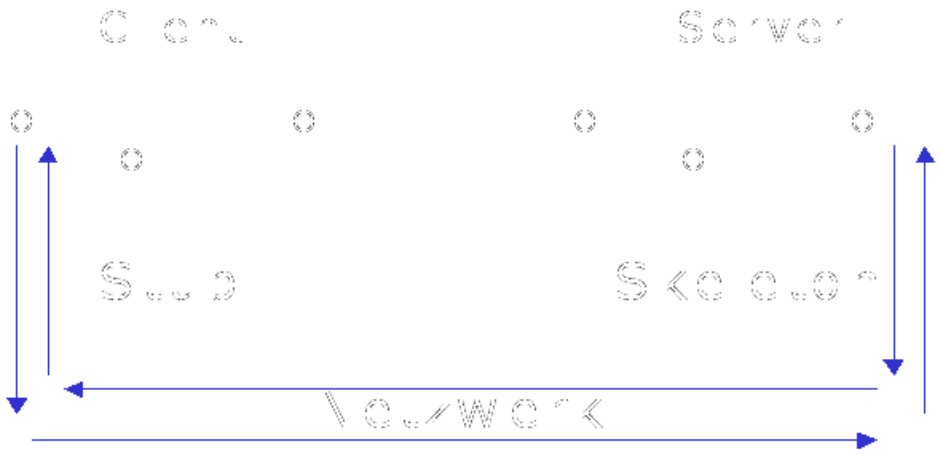
- Transparenter Methodenaufruf von Objekten die durch verschiedene virtuelle Maschinen verwaltet werden.
- Integration des Verteilungsmodells in die Programmiersprache unter weitestgehendem Erhalt der Semantik.
- Transparente Anlage des verteilten Objektmodells.
- Schaffung eines einfachstmöglichen Mechanismus zur Erstellung verteilter Applikationen.
- Erhalt der durch die Laufzeitumgebung geschaffenen Typsicherheit.
- Unterstützung verschiedener Referenzierungssemantiken.
- Erhalt der Java-Sicherheitsumgebung auch für verteilt ablaufende Applikationen.
- Unterstützung einer Call-back-Schnittstelle für Applets.

Technische Grundkonzepte

Integrale Bestandteile des RMI-Ansatzes und direkte Folgerungen der angestrebten Transparenz sind Architekturbestandteile *Stub* und *Skeleton*. Beide erfüllen symmetrische Aufgabenstellungen und werden mittels eines Dienstprogrammes automatisch aus dem übersetzten Bytecode des für den entfernten Zugriff anzubietenden Objekts erstellt.
Zur Ausführungszeit wirkt der Stub als Stellvertreter des entfernten Serverobjektes innerhalb der Adreßumgebung des Clients, analog fungiert der Skeleton als serverseitiger Stellvertreter des aufrufenden Objekts.

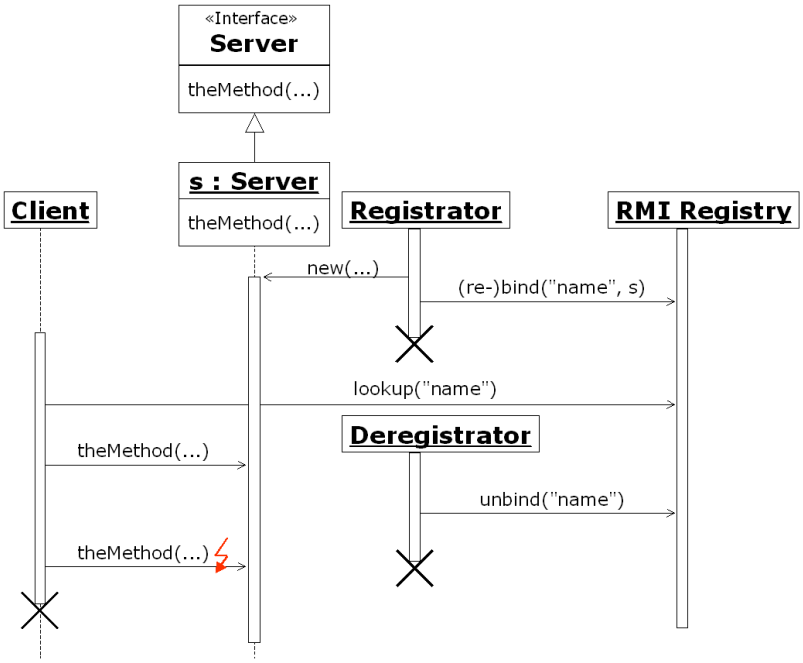
Abbildung 21 zeigt den Ablauf des Aufrufs eines entfernten Objekts mittels RMI. Hierbei spielen die Stub- und die Skeletonkomponente wie folgt zusammen:

1. Stub: Initiierung einer Verbindung mit derjenigen entfernt ablaufenden virtuellen Maschine welche das anzusprechende Objekt verwaltet.
2. Stub: Erzeugung der benötigten Leitungsdarstellung und Übertragung der Aufrufparameter an die entfernte virtuelle Maschine.
3. Stub: Warten auf Ausführungsende und evtl. erzeugten Rückgabeparameter.
4. Skeleton: Empfang und Decodierung der Leitungsdarstellung der Aufrufparameter.
5. Skeleton: Aufruf des (lokalen) Objekts und Entgegennahme etwaig erzeugter Rückgabewerte.
6. Skeleton: Erzeugung der benötigten Leitungsdarstellung und Übertragung der Rückgabewerte an die entferne virtuelle Maschine.
7. Stub: Empfang und Decodierung der Leitungsdarstellung des etwaig übergebenen Rückgabewertes.
8. Stub: Übergabe des Rückgabewertes (falls vorhanden) an (lokales) Aufruferobjekt.



Um das entfernte Objekt lokalisieren zu können, ohne daß dem Aufrufer der Server, auf dem dieses verwaltet wird, bekannt sein muß, wird als zusätzliche Architekturkomponente ein Verzeichnisdienst verwendet. Innerhalb dieses Dienstes wird jedes Objekt, auf das der entfernte Zugriff angeboten werden soll, ein eineindeutiger Name und die physische Objektlokation verwaltet.

Unter Berücksichtigung des Verzeichnisdiensts ergibt sich daher folgender Interaktionsablauf:



Das Bild zeigt das Objekt `s` als Instanz der Klasse `Server` sowie das auf die darin enthaltene Methode `theMethod` zugreifenden `Client`-Objekt. Zusätzlich ist mit `RMI Registry` der Verzeichnisdienst zur Verwaltung der zugreifbaren Objekte dargestellt. Die beiden Objekte `Registrator` bzw. `Deregistrator` dienen zur Ablage (Bindung) des `Server`-Objektes an einen im Verzeichnisdienst gespeicherten Namen bzw. der Aufhebung dieser Bindung.

Da die Bindung objektbezogen erfolgt, muß die registrierende Instanz (im Beispiel `Registrator`) eine Referenz auf das abzulegende Objekt besitzen. Die Beispielinteraktion unterstellt daher die Erzeugung der benötigten `Server`-Instanz durch den `Registrator` selbst. Dieser bindet das `Server`-Objekt unter dem Klarnamen `name` und legt diese Zuordnung durch Aufruf der Methode `bind` oder `rebind` in der Registry ab. Dort kann danach mittels der Methode `lookup`, unter Angabe des Namens eine Referenz auf das gebundene Objekt, erfragt werden. Die Interaktion zeigt diesen Aufruf durch den `Client`.
Ausgehend von dieser Referenz kann durch den `Client` die durch die `Server`-Schnittstelle publizierte Methode `theMethod` aufgerufen werden.
Ferner zeigt der Ablauf die Deregistrierung des `Server`-Objektes durch das `Deregistrator`-Objekt mittels der Methode `unbind`. Für diesen Aufruf ist keine Objektreferenz, sondern lediglich der Klarname des Objektes notwendig.

Vorgehen zur Entwicklung einer RMI-Applikation

Die Entwicklung einer RMI-basierten Applikation vollzieht sich immer im selben Ablauf der Erstellung der notwendigen Komponenten.
Dieser Ablauf ist nachfolgend anhand einer simplen Applikation veranschaulicht:

1. Definition der notwendigen Serverschnittstelle (HelloInterface)
2. Implementierung des Servers (HelloServer)
3. Implementierung des oder der Clientapplikationen (HelloClient)
4. Erzeugung der Stub- und Skeletonklassen (durch das Werkzeug rmic)
5. Start des RMI-Verzeichnisdiensts (rmiregistry)
6. Registrierung des Serverobjektes
7. Start des Servers
8. Ausführung des Clients

Die Schnittstelle HelloInterface:

Beispiel 94: Die Schnittstelle HelloInterface

```
(1)import java.rmi.Remote;
(2)import java.rmi.RemoteException;
(3)
(4)public interface HelloInterface extends Remote {
(5)     public String sayHello() throws RemoteException;
(6)}
```

Download des Beispiels

Die Schnittstelle enthält die Operationen aller Methoden, die ein Serverobjekt zur Verwendung zur Verfügung stellen soll.
Hierbei sind naturgemäß keine als private oder protected deklarierten Operationen zugelassen, da auf sie kein Zugriff möglich wäre.
Für jede Operation ist die möglicherweise auftretende Erzeugung einer Ausnahme des Typs RemoteException vorzusehen. Dieser Typ kapselt durch Kommunikationsfehler verursachte Ausnahmen.
Zusätzlich ist die gesamte Schnittstelle als Spezialisierung der Standardschnittstelle Remote zu definieren. Diese Schnittstelle definiert selbst keine Operationen, sondern dient lediglich zur Kennzeichnung von Schnittstellen, die entfernt aufrufbare Methoden versammeln.

Die Klasse HelloServer:

Beispiel 95: Die Klasse HelloServer

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.RemoteException;
(4)import java.rmi.server.UnicastRemoteObject;
(5)
(6)public class HelloServer
(7)     extends UnicastRemoteObject
(8)     implements HelloInterface {
(9)
(10)    protected HelloServer() throws RemoteException {
(11)            super();
(12)            try {
(13)                    Naming.rebind("rmi://localhost:1099/HelloService", this);
(14)            } catch (RemoteException re) {
(15)                    re.printStackTrace();
(16)            } catch (MalformedURLException mue) {
(17)                    mue.printStackTrace();
(18)            }
(19)    }
(20)
(21)    public static void main(String argv[]) throws RemoteException {
(22)            new HelloServer();
(23)    }
(24)
(25)    public String sayHello() throws RemoteException {
(26)            return (new String("Hello world!"));
(27)    }
(28)}
```

Download des Beispiels

Die Klasse enthält neben der Methode der durch die Schnittstelle exponierten Operation sayHello auch im Konstruktor die Registrierung innerhalb des RMI-Verzeichnisdienstes. Dieser ist jedoch, ebenso so wie alle anderen möglicherweise codierten Methoden, die nicht Bestandteil der explizierten Schnittstelle sind, nicht von einem entfernten Objekt aus zugreifbar.
Formal implementiert die Klasse mit HelloInterface die öffentlich zur Verfügung gestellten Operationen. Zusätzlich muß sie als Spezialisierung der API-Klasse UnicastRemoteObject definiert sein um durch die virtuelle Maschine zur Laufzeit als Objekt mit entfernten Zugriffen behandelt zu werden. Dies zieht von der

Behandlung des lokalen Falles abweichende Handhabungen der Parameterübergabe sowie der Freispeicherverwaltung und -rückgewinnung nach sich.
Die Implementierung der durch die Schnittstelle bekanntgegebenen und später durch entfernte Objekte nutzbaren Operation erfolgt identisch zur herkömmlichen Umsetzung im lokalen Falle. Einzige äußerlich erkennbare Änderung ist die Deklaration der möglicherweise bei der Ausführung auftretenden `RemoteException`. Diese wird jedoch üblicherweise nicht aktiv durch den anwenderdefinierten Code ausgelöst, sondern wird durch die Java-Standardbibliothek bzw. die -Laufzeitumgebung erzeugt. Die Deklaration dieser Ausnahme ist für Operationen, auf deren Implementierung entfernt zugegriffen werden soll, zwingend.
Zusätzlich nimmt die Applikation im Konstruktor die Registrierung des neu erzeugten Objektes unter dem frei gewählten Namen `HelloService` in der RMI-Registry vor. Hierbei wird die Methode `rebind` verwendet, um eine eventuell bereits existierende Objektbindung unter diesem Namen zu überschreiben. Auffallend ist die zur Bindung in der Registry verwendete URI `rmi://localhost:1099/HelloService`. Sie enthält nach dem idenzifizierenden URI-Schema (`rmi`) zwingend die Identifikation derjenigen Maschine, die das bereitzustellende Objekt verwaltet. Einschränkend sei hier jedoch angemerkt, daß RMI lediglich die Registrierung von Objekten vorsieht, die auf derselben physischen Maschine verwaltet werden wie der Registrierungsdienst selbst. Daher ist neben dem Pseudonamen `localhost` lediglich der Name der Maschine bzw. deren IP-Adresse oder die Pseudoadresse `127.0.0.1` zugelassen. Die optionale Portnummer benennt die logische TCP-Portadresse, über welche Kommunikationen mit der RMI-Registry abgewickelt werden. Daran im Anschluß folgt der anwenderdefinierte Namen des abzulegenden Objekts. Obwohl nach erfolgreicher Ausführung der Methode `rebind` die Kontrolle an das neu erzeugte Objekt der Klasse `HelloServer` zurückübergeben wird, terminiert das Programm nach Abarbeitung des Konstruktors (und damit Abschluß der Anweisungen in `main`) nicht, sondern wird durch die RMI-Registry wartend auf Aufrufe der Methode `sayHello` im Hauptspeicher gehalten.

Die Klasse `HelloClient`:

Beispiel 96: Die Klasse HelloClient

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.NotBoundException;
(4)import java.rmi.RemoteException;
(5)
(6)public class HelloClient {
(7)     public static void main(String[] args) {
(8)             HelloInterface obj = null;
(9)             try {
(10)                    obj = (HelloInterface) Naming.lookup("rmi://53.16.71.55:1099/
HelloService");
(11)            } catch (MalformedURLException mue) {
(12)                    mue.printStackTrace();
(13)            } catch (RemoteException re) {
(14)                    System.err.println("Lookup error");
(15)                    re.printStackTrace();
(16)            } catch (NotBoundException nbe) {
(17)                    System.out.println("Object not bount to registry");
(18)                    nbe.printStackTrace();
(19)            }
(20)            System.out.println("retrieved object's oid=" + obj);
(21)            try {
(22)                    System.out.println(obj.sayHello());
(23)            } catch (RemoteException re) {
(24)                    System.err.println("Error while invoking sayHello");
(25)                    re.printStackTrace();
(26)            }
(27)    }
(28)}
```

Download des Beispiels

Die Klasse des Beispiels 96 enthält den notwendigen Code um auf eine entfernt verwaltete und abgelegte Ausprägung der Schnittstelle `HelloInterface` zuzugreifen.
Zunächst wird vermöge der Methode `lookup` eine Referenz eines `HelloInterface`-konformen Objekts aus dem RMI-Verzeichnisdienst erfragt. Hierzu dient die bereits zur Ablage verwendete URI auch zu Ermittlung des Objekts durch das nutzungswillige Clientobjekt. Naturgemäß müssen sich Client- und Serverobjekt nicht auf derselben physischen Maschine befinden, sondern müssen lediglich durch eine bestehende Netzwerkverbindung miteinander kommunizieren können.
Nach Erhalt der Referenz kann das entfernte Objekt einem durch die lokale virtuelle Maschine verwaltetem gleichgestellt verwendet werden.
Da das Server-Objekt auch nach Abarbeitung seiner Methode resident im Speicher der entfernten virtuellen Maschine verbleibt ermittelt jeder Client bei jedem Aufruf der Methode `lookup` denselben Verweis auf dasselbe physische Objekt. Aus diesem Grunde ist bei der Programmierung darauf zu achten, daß sich nebenläufig stattfindende Zugriffe nicht gegenseitig behindern oder inkorrekte Ergebnisse liefern.

Die **Nutzung des Verzeichnisdienstes ist keine notwendige Voraussetzung** zur Ausführung entfernter Methodenaufrufe. Besitzt der Client bereits eine Referenz auf das entfernte Objekt (etwa aus einem Aufruf durch dieses welcher die Objektreferenz als Parameter enthielt), so kann der entfernte Methodenaufruf auch ohne vorherige Ermittlung der Referenz aus dem RMI-Verzeichnisdienst erfolgen.

Die **Parameterübergabe zwischen Client und entferntem Serverobjekt** geschieht anders als in der Java-üblichen Weise, d.h. für primitivwerte Parameter als Wert und Objektwertige als Referenz, durchgängig durch Wertübergabe.
Dies liegt darin begründet, daß der innerhalb der virtuellen Maschine verwandte Referenzierungsmechanismus auf Objekte als Inhalte des Heaps der virtuellen Maschine, nicht über Rechnergrenzen hinweg portabel gestaltet ist. Durch den notwendigen Aufwand der durch die Stub- und Skeletonkomponente zu codierenden bzw. zu decodierenden Daten steigt bei komplexen Objektstrukturen (z.B. Bäumen) sowohl die CPU-Last der Client- als auch der Servermaschine sowie die Netzwerklast.
Einzige Ausnahme von dieser Mimik bilden per RMI zugreifbare (entfernte) Objekte selbst. Sie werden nicht serialisiert und übertragen, sondern transparent durch ihre Stub-Repräsentation ersetzt, die an ihrer statt übertragen wird. Dies gilt insbesondere natürlich auch für Selbstreferenzen des Serverobjektes, die dieses als Rückgabewert liefert.

Naturgemäß ergeben sich auch für die **verteilte Freispeicherverwaltung** (distributed garbage collection) veränderte Rahmenbedinungen, wenngleich gemäß dem RMI-Designziel größtmöglicher Transparenz versucht wurde, eine ähnliche Handhabung wie bereits für den lokalen Fall zu realiseren.
Das in RMI gewählte Vorgehen orientiert sich an dem in der Programmiersprache Modula-3 realisierten: Jede virtuelle Maschine führt eine Tabelle mit einem Boole'schen Eintrag für jedes aufgrund der publizierten Schnittstellendefinition potentiell entfernt zugreifbare Objekt. Jeder Eintrag ist vorgabegemäß mit dem Wert *clean* belegt. Wird eine Referenz auf das Objekt erfragt, so wird dieses als *dirty* markiert. Erfolgt innerhalb einer gewissen Zeit (*lease Zeit*) keine erneute Anfrage nach einer weiteren Referenz oder ein Zugriff auf das Objekt, so wird die Markierung auf *clean* zurückgesetzt und das Objekt so zur Freigabe markiert.
Die Zeitspanne bis zur automatisierten Markierung als nicht mehr benötigtes Objekt kann durch die Java-Property `java.rmi.dgc.leaseValue` kontrolliert werden. Vorgabegemäß ist sie auf zehn Minuten gesetzt.

Die Activation-API

Neben der u.U. speicherplatz- und laufzeitaufwendigen „Dauerinstanziierung“ eines entfernt zugreifbaren Objektes, wie sie das vorherige Beispiel einführt, existiert die Möglichkeit, Objekte bei Bedarf dynamisch erzeugen zu lassen.

Das nachfolgende Beispiel zeigt dies. Der Ablauf in Erstellung und Ausführung ähnelt dem zuvor für einfache RMI-Applikationen gezeigten, jedoch unter Nutzung einer zusätzlichen Ausführungskomponente.

1. Definition der notwendigen Serverschnittstelle (`ActivatableHelloInterface`)
2. Implementierung des Servers (`ActivatableHelloServer`)
3. Implementierung einer Applikation zur Registrierung des Servers (`ActivatableSetup`)
4. Implementierung des oder der Clientapplikationen (`ActivatableHelloClient`)
5. Erzeugung der Stub- und Skeletonklassen (durch das Werkzeug `rmic`)
6. Start des RMI-Verzeichnisdiensts (`rmiregistry`)
7. Start des RMI-Daemons (`rmid`)
8. Registrierung des Serverobjektes
9. Ausführung des Clients

Im Vergleich zum vorhergehenden Ablauf wurde in diesem Beispiel die Registrierung des Servers in eine eigenständige Applikation ausgelagert, da ihr Ablauf im Konstruktor des Serverobjektes bei wiederholter Instanziierung durch die RMI-Umgebung redundant ausgeührt würde.
Ferner entfällt der explizite Start des Serverobjektes, da diese Aufgabe nunmehr automatisiert durch das Framework übernommen wird.
Die Komponente welcher die Kontrolle und Durchführung der Startvorgänge obliegt, der RMI-Daemon, muß dafür einmalig zusätzlich durch den Anwender gestartet werden.

Die Schnittstelle `ActivatableHelloInterface`:

Beispiel 97: Die Klasse ActivatableHelloInterface

```
(1)import java.rmi.Remote;
(2)import java.rmi.RemoteException;
(3)
(4)public interface ActivatableHelloInterface extends Remote {
(5)     public String sayHello() throws RemoteException;
(6)}
```

Download des Beispiels

Seitens der Schnittstellenbeschreibung der exponierten Operationen ergeben sich durch den Übergang auf

die servergesteuerte Instanziierung keine Änderungen.

Die Klasse ActivatableHelloServer:

Beispiel 98: Die Klasse ActivatableHelloServer

```
(1)import java.rmi.MarshalledObject;
(2)import java.rmi.RemoteException;
(3)import java.rmi.activation.Activatable;
(4)import java.rmi.activation.ActivationID;
(5)
(6)public class ActivatableHelloServer
(7)     extends Activatable
(8)     implements ActivatableHelloInterface {
(9)
(10)    public ActivatableHelloServer(ActivationID id, MarshalledObject data)
(11)            throws RemoteException {
(12)            super(id, 0);
(13)    }
(14)
(15)    public String sayHello() throws RemoteException {
(16)            return (new String("Hello world!"));
(17)    }
(18)}
```

Download des Beispiels

Seitens der Implementierung des für entfernte Aufrufe zur Verfügung gestellten Objekts ergeben sich durch die geänderte Instanziierungsmimik keine gravierenden Änderungen.
Lediglich die geänderte Spezialisierung, statt UnicastRemoteObject wird jetzt von der Klasse Remote abgeleitet, fällt zunächst ins Auge. Diese Änderung ist notwendig, um dem Framework die Möglichkeit der Instanziierung zu eröffnen, die bei einer völlig freien Formulierung --- insbesondere des Konstruktors --- nicht möglich wäre. Im Kern definiert die Klasse Activateable, neben einer Reihe von Hilfsmethoden, vier verschiedene Konstruktoren zur Anlage von Objekten. Mindestens einer davon ist zwingend durch die abgeleitete Klasse zu implementieren um dem eine automatisierte Objekterzeugung zu erreichen.
Hinsichtlich der Problemlogik, im Beispiel der Methode sayHello ergeben sich keine Änderungen.

Die Klasse ActivatableHelloClient:

Beispiel 99: Die Klasse ActivatableHelloClient

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.NotBoundException;
(4)import java.rmi.RemoteException;
(5)
(6)public class ActivatableHelloClient {
(7)     public static void main(String[] args) {
(8)             ActivatableHelloInterface obj = null;
(9)             try {
(10)                    obj = (ActivatableHelloInterface) Naming.lookup
("rmi://53.16.71.55:1099/HelloObj");
(11)            } catch (MalformedURLException mue) {
(12)                    mue.printStackTrace();
(13)            } catch (RemoteException re) {
(14)                    System.err.println("Lookup error");
(15)                    re.printStackTrace();
(16)            } catch (NotBoundException nbe) {
(17)                    System.out.println("Object not bound to registry");
(18)                    nbe.printStackTrace();
(19)            }
(20)            System.out.println("retrieved object's oid=" + obj);
(21)            try {
(22)                    System.out.println(obj.sayHello());
(23)            } catch (RemoteException re) {
(24)                    System.err.println("Error while invoking sayHello");
(25)                    re.printStackTrace();
(26)            }
(27)    }
(28)}
```

Download des Beispiels

Wie bereits beim Server gezeigt, ergeben auch Clientseitig durch die Veränderte Erzeugungssemantik keine Änderungen in der Ablauflogik, abgesehen von durch die geänderten Namen dieses Beispiels

bedingte Modifikationen.

Die Klasse `ActivatableSetup`:

Beispiel 100: Die Klasse ActivatableSetup

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.MarshalledObject;
(3)import java.rmi.Naming;
(4)import java.rmi.RMISecurityManager;
(5)import java.rmi.RemoteException;
(6)import java.rmi.activation.Activatable;
(7)import java.rmi.activation.ActivationDesc;
(8)import java.rmi.activation.ActivationException;
(9)import java.rmi.activation.ActivationGroup;
(10)import java.rmi.activation.ActivationGroupDesc;
(11)import java.rmi.activation.ActivationGroupID;
(12)import java.rmi.activation.UnknownGroupException;
(13)import java.util.Properties;
(14)
(15)public class ActivatableSetup {
(16)    public static void main(String argv[]) throws RemoteException {
(17)            System.setSecurityManager(new RMISecurityManager());
(18)            Properties props = new Properties();
(19)            props.put("java.security.policy", "policy");
(20)            ActivationGroupDesc.CommandEnvironment ace = null;
(21)            ActivationGroupDesc exampleGroup = new ActivationGroupDesc(props, ace);
(22)            ActivationGroupID agi = null;
(23)            try {
(24)                    agi = ActivationGroup.getSystem().registerGroup(exampleGroup);
(25)            } catch (RemoteException re) {
(26)                    re.printStackTrace();
(27)            } catch (ActivationException ae) {
(28)                    ae.printStackTrace();
(29)            }
(30)            String location = "file:/home/mario/RMITest/activation/";
(31)            MarshalledObject data = null;
(32)            ActivationDesc desc =
(33)                    new ActivationDesc(agi, "ActivatableHelloServer", location, data);
(34)
(35)            ActivatableHelloInterface stub = null;
(36)            try {
(37)                    stub = (ActivatableHelloInterface) Activatable.register(desc);
(38)            } catch (UnknownGroupException uge) {
(39)                    uge.printStackTrace();
(40)            } catch (RemoteException re) {
(41)                    re.printStackTrace();
(42)            } catch (ActivationException ae) {
(43)                    ae.printStackTrace();
(44)            }
(45)            try {
(46)                    Naming.rebind("rmi://localhost:1099/HelloObj", stub);
(47)            } catch (RemoteException re) {
(48)                    re.printStackTrace();
(49)            } catch (MalformedURLException mue) {
(50)                    mue.printStackTrace();
(51)            }
(52)
(53)    }
(54)}
```

Download des Beispiels

Die nachhaltigsten Veränderungen spiegelt die neu eingeführte Klasse zur Registrierung des Objekts für entfernten Zugriff wieder.
Sie parametrisiert und initialisiert zunächst den RMI-Daemon, der später im Bedarfsfalle die Erzeugung der benötigten Objekte übernimmt. Daran Anschließend wird ein konform zur publizierten Schnittstelle umgesetztes Objekt gemäß dem bisherigen Verfahren innerhalb des RMI-Verzeichnisdienstes an einen frei gewählten Namen gebunden.
Der automatische Aktivierungsprozeß erfordert zur Laufzeit des RMI-Daemons den Zugriff auf ein lokales oder erreichbares Dateisystem. Für alle Fälle des Zugriffs auf ein nichtlokales Dateisystem muß daher eine geeignete Sicherheitseinstellung der Javaausführungsumgebung gewählt werden. Im Standardumfang der API bereits enthalten ist die Klasse `RMISecurityManager`. Sie erlaubt es dynamisch geladenen Code innerhalb einer RMI-Umgebung auszuführen.
Zusätzlich kann über die Java-Property `java.security.policy` eine sog. *policy*-Datei referenziert werden,

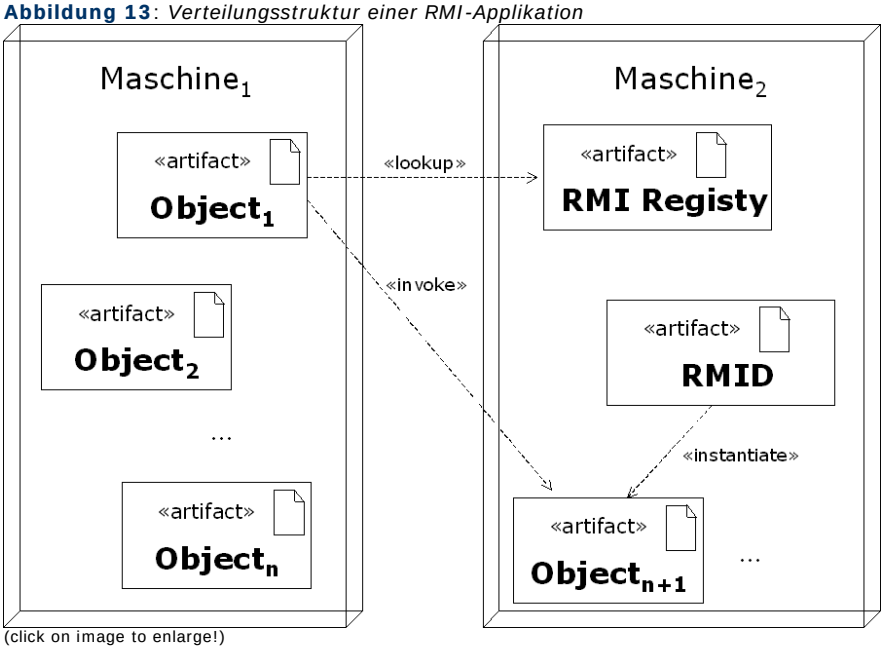
die nach Herkunft abgestufte Zugriffsrechte für signierte Inhalte definiert.
Für das wiedergegebene Beispiel wurde eine eine sehr lose Policy gewählt, die jeglichem Code den Zugriff auf alle Java-Funktionalitäten gestattet. (Policy-File) Mittels der Policyeigenschaften `com.sun.rmi.rmid.ExecPermission` und `com.sun.rmi.rmid.ExecOptionPermission` läßt sich eine noch feinere Zugriffssteuerung erreichen, wobei jedoch diese vorgabegemäß immer unabhängig von der Codeherkunft und einer evtl. existierenden Signatur zu definieren sind.
Die späteren Aktivierungsaufrufe werden innerhalb einer *Aktivierungsgruppe* gebündelt. Eine solche wird mittels einer Beschreibung (als Ausprägung der Klasse `ActivationGroupDesc`) hinsichtlich ihrer eineindeutigen Identität und ihrer Inialisierungsdaten beschrieben (im vorliegenden Beispiel ist sie mit `null` initialisiert).
Diese Aktivierungsgruppe wird mit den durch die Policy-Datei definierten Rechten ausgestattet erzeugt.
Die notwendige eineindeutige Identität wird durch das Aktivierungssystem mittels des Aufrufs der Methode `registerGroup` automatisch zur Verfügung gestellt. Hierbei gilt eine eindeutige Zuordnung zwischen Gruppenkennung (`ActivationGrouID`) und verwaltender virtueller Maschine. Soll daher ein Objekt oder eine ganze Gruppe in einer separaten virtuellen Maschine ablaufen, so genügt es, eine andere Gruppenkennung durch das System zu generieren und bei der nachfolgend diskutierten Registrierung zu übergeben.
Anschließend wird durch das Beschreibungsobjekt mit dem Klassennamen des automatisch zu instanziierenden Objektes (im Beispiel: `ActivatableHelloServer`) sowie der physischen Lokation der Klasse und etwaigen Initalisierungsdaten erzeugt und der Aktivierungsgruppe zugeordnet.
Als letzter Schritt erfolgt die Registrierung der durch die Aktivierungsbeschreibung beschriebenen Klasse von Objekten, um so ihre spätere Erzeugung zu gewährleisten. Dies geschieht durch Aufruf der statischen Methode `register` der als Parameter die erzeugte Aktivierungsbeschreibung übergeben wird.
Der Aufruf dieser Methode liefert den Stub des registrierten Objekts. Aufgrund der Stellvertreterrolle des Stubs (d.h. seiner Konformität zur Schnittstelle `Remote`) kann dieser zur Registrierung der Objekte in der RMI-Registry verwendet werden.

Der Aufruf der Klasse `ActivationSetup` erfolgt durch Übergabe von gültigen Belegungen der notwendigen Laufzeiteigenschaften an die virtuelle Javamaschine: `java -Djava.security.policy=./policy -Djava.rmi.codebase=file:/home/mario/RMITest/activation/ ActivationSetup`.

Objekte werden zwar durch den RMI-Daemon automatisiert aktiviert, sobald ein Zugriff auf sie erfolgt (*lazy activation*), jedoch nicht mehr deaktiviert. Das bedeutet, daß sie bis zur Terminierung der virtuellen Maschine im Hauptspeicher verbleiben. Abhilfe schafft hier der explizite Aufruf der Methode `inactive` der ein bereits bestehendes Objekt passiviert, jedoch nicht aus der Registrierung entfernt.

Ähnlich wie bereits für die RMI-Registry angemerkt, muß auch der RMI-Daemon auf derselben physischen Maschine zur Ausführung gelangen, die das für entfernte Zugriffe bereitzustellende Objekt enthält.
Es ergibt sich daher generell folgendes Verteilungsschema:

Abbildung 13: *Verteilungsstruktur einer RMI-Applikation*



(click on image to enlarge!)

Um die irreführenden Begriffe *Client* und *Server* zu vermeiden, sind die beiden Maschinen neutral durch $\texttt{Maschine}_1$ und $\texttt{Maschine}_2$ bezeichnet (Die Irreführung erklärt sich aus der mangelnden Trennschärfe der beiden Begriffe, sobald ein dienstanbieter Knoten auch gleichzeitig Dienste anderer Knoten nutzt, er damit gleichzeitig Server (seines Anfrages) und Client (bezüglich seiner Anfrage) ist).
Ein $\texttt{Object}_1$, das auf ein entferntes $\texttt{Object}_{n+1}$ zugreift (`invoke`) kann optional mittels der `RMI Registry`, welche auf demselben Rechner wie die virtuelle Maschine die das $\texttt{Object}_{n+1}$ verwaltet ablaufen muß, die notwendige Objektreferenz ermitteln (`lookup`). Sofern das zuzugreifende Objekt (noch) nicht existiert kann es über den Aktivierungsmechanismus gestartet werden. Die notwendige Objekterzeugung (`instantiate`) nimmt hierbei der RMI-Daemon (`RMID`) vor.

Sicherheitsgesichtspunkte

Generell werden die durch Stubs und Skelettons verarbeiteten Objekte oder Primitivwerte lediglich serialsiert, aber keine weiteren Maßnahmen zur Gewährleistung der Vertraulichkeit der übertragenen Daten ergriffen.
Durch die Integration des RMI-Mechanismus in die Java-API besteht die Möglichkeit innerhalb der Programmiersprache Möglichkeiten zur Übertragungssicherung zu ergreifen.
Seitens RMIs ist es hierbei vorgesehen die bestehende (einfache) TCP-Socketverbindung gegen beliebige selbsterstellte auszutauschen. Die Erzeugung der notwendigen Sockets muß mittels der Schnittstellen `RMIServerSocketFactory` bzw. `RMIClientSocket` abgewickelt werden. Auf diesem Wege stellt bleibt die Typsicherheit der Laufzeitumgebung, auch bei abgeänderten Transportprotokollen gewahrt.
Das nachfolgende Beispiel zeigt die Umsetzung einer einfachen gesicherten Verbindung mittels XOR-Bearbeitung aller versandten Inhalte.

Aus Gründen der Übersichtlichkeit sind die inhaltlich unverändert aus dem ersten Beispiel dieses Kapitels übernommenen Dateien `Hello.java` sowie `HelloClient.java` nicht wiedergegeben.

Die Klasse `HelloImpl`:

Beispiel 101: Die Klasse HelloImpl

```
(1)import java.rmi.Naming;
(2)import java.rmi.RMISecurityManager;
(3)import java.rmi.RemoteException;
(4)import java.rmi.server.UnicastRemoteObject;
(5)
(6)public class HelloImpl extends UnicastRemoteObject implements Hello {
(7)     public HelloImpl(String protocol, byte[] pattern) throws RemoteException {
(8)             super(
(9)                     0,
(10)                    new XorClientSocketFactory(protocol, pattern),
(11)                    new XorServerSocketFactory(protocol, pattern));
(12)    }
(13)
(14)    public String sayHello() throws RemoteException {
(15)            StringBuffer sb = new StringBuffer();
(16)            for (int i=0;i<100;i++)
(17)                    sb.append("X");
(18)            return sb.toString();
(19)    }
(20)
(21)    public static void main(String args[]) {
(22)
(23)            System.setSecurityManager(new RMISecurityManager());
(24)            byte[] aPattern = {(byte) 1011 };
(25)            try {
(26)                    HelloImpl obj = new HelloImpl("xor", aPattern);
(27)                    Naming.rebind("rmi://53.16.71.55:1099/HelloServer", obj);
(28)                    System.out.println("HelloServer bound in registry");
(29)            } catch (Exception e) {
(30)                    System.out.println("HelloImpl err: " + e.getMessage());
(31)                    e.printStackTrace();
(32)            }
(33)    }
(34)}
```

Download des Beispiels

Die Klasse zeigt als einzige offensichtliche Veränderung die Ergänzung des Konstruktors des entfernt zugreifbaren Objekts um die Erzeugung der alternativ verwendeten XOR-Sockets; alle weiteren Aufwände die veränderte Kommunikation abzuwickeln sind für den Programmierer transparent und werden durch das RMI-Framework sowie die Implementierung der XOR-Sockets erbracht.

Die Bestandteile der XOR-Socketimplementierung finden sich hier:

- `XorClientSocketFactory`
- `XorInputStream`
- `XorOutputStream`
- `XorServerSocket`
- `XorServerSocketFactory`
- `XorSocket`

Web-Referenzen 12: Weiterführende Links

- RMI @ SUN
- RMI Tutorial @ SUN
- Ninja RMI eine freie RMI-Implementierung
- JDBC FAQ @ JGuru.com
- G. Reese: *Database Programming with JDBC and Java*. O'Reilly, 1997.
- Verhältnis von X/Open CLI und ODBC

4.3 Representational State Transfer (REST)

Hinweis: Dieses Kapitel ist noch unvollständig

Servlets

Ergebnisse eines Anwendungssystems werden typischerweise zur Laufzeit auf der Basis von Anwendungslogik und in Abhängigkeit der Eingaben berechnet. Dies gilt auch für Web-basierte Anwendungssysteme. Jedoch stellt das HTML-basierte Web konzeptionell zunächst ein statisches Medium, welches vorgefertigte Hypertextseiten auf über das HTTP-Protokoll übermittelte Anforderung ausliefert, dar.

Zur Versöhnung dieses Widerspruchs haben sich zwei Ansätze herausgebildet: Zum einen die vollständige Verlagerung der Berechungslogik und Erbebnispräsentation in den anfragenden Client durch Übertragung eines vollständigen binären Programms. Zum anderen der Aufruf von serverseitig vorgehaltenen Applikationen, die nicht im Adresskontext des Webservers operieren.
Bekanntester Vertreter der ersten Kategorie sind die *Java Applets*, welche eine als *Java Bytecode* bezeichnete portable Binärrepräsentation an den Client übertragen, die dieser durch eine interpretativ arbeitende *virtuelle Maschine* ausführt. Die in der Praxis bedeutsamste Realisierung zum Zugriff auf extern vorliegende Programme stellt das *Common Gateway Interface* (CGI) dar.

Jedoch ergeben sich bei beiden Lösungsalternativen spezifische Probleme. So wird zur vollständigen Verlagerung der Ausführungslogik auf den Client, wie sie für Applets propagiert wird, die Möglichkeit zur Ausführung fremden Codes erzwungen. Dies läßt sich zwar durch geeignete Sicherheitsmechanismen absichern, jedoch bleibt die Notwendigkeit, den Code per Internet zu beziehen und lokal auszuführen.
Der CGI-Ansatz krankt an der mangelnden Ausführungsgeschwindigkeit, welche durch die notwendigen Prozeßkontextwechsel drastisch sinkt. Gleichzeitig unterliegt die auf diesem Wege angebundene Kontrollogik nicht der Verwaltungshoheit des Servers.

Java Servlets stellen konzeptionell eine konfluente Weiterentwicklung der beiden Ansätze dar, die es sich zum Ziel gesetzt hat die angeführten Nachteile zu mildern.
Hierbei handelt es sich primär um Server-seitig ausgeführten Java-Code, der über Internetprotokolle (zumeist HTTP) mit dem Benutzer interagiert. Technisch gesehen stellen Servlets Module eines Web-Servers dar, die dynamisch anstelle der Auslieferung statischer Inhalte (z.B. HTML-Seiten) ausgeführt werden. Zur Ausführung dient eine als *Servlet Container* bezeichnete Softwarekomponente, welche die durch die dort abgelegten Servlets bedienten Schnittstellen der Standard-API bedient.

Bis zur Freigabe der Java-2-Plattform war die Servlet-API Bestandteil des JDK. Mittlerweile sind neben der Servlet-Spezifikation von SUN auch einige Implementierungen in verbreiteten Web-Servern verfügbar. Daher wurde die gesamte Servlet-API aus dem `javax`-Paket entfernt, und steht nunmehr nur noch als Bestandteil diverser Web-Server-Produkte (u.a. J2EE-Servern) zur Verfügung.

Lebenszyklus eines Servlets:

1. Der Server lädt das Servlet
2. Clientanforderungen werden durch das Servlet bedient
3. Der Server entlädt das Servlet (evtl. erst während des Server-Shutdown-Vorganges)

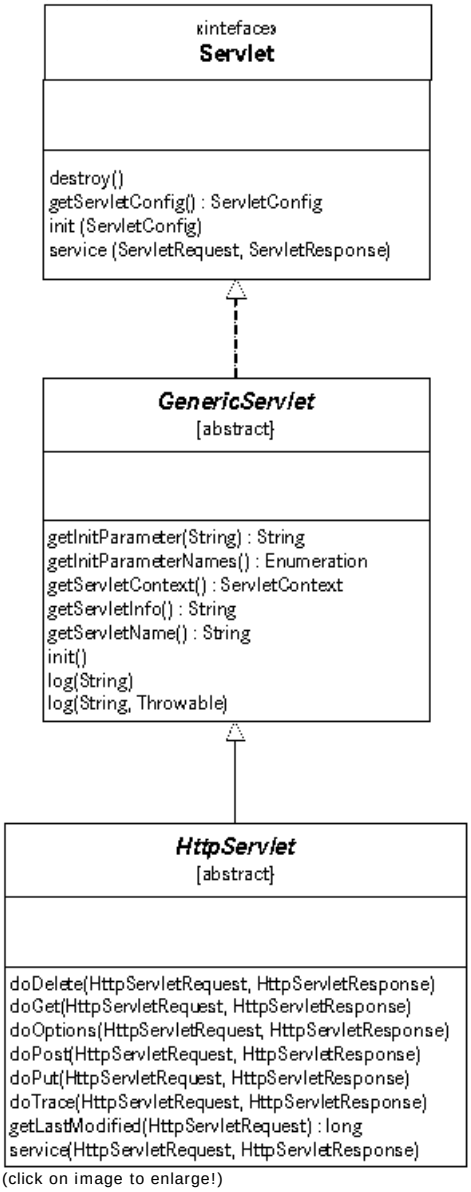
Im Detail korrespondieren diese drei Phasen mit Methoden der Servlet-Schnittstelle:

- `init(ServletConfig)`
  Initialisierung des Servlets nach dem Ladevorgang, und vor dessen Ausführung.
- `service(ServletRequest, ServletResponse)`
  Aufruf durch Servlet-Container (üblicherweise Web-Server) um Client-Anfrage zu bedienen.
  Pro Anfrage wird durch den Container vorgabegemäß ein neuer Thread gestartet. Beim Zugriff auf gemeinsam genutzte Ressourcen (Dateien, Datenbanken, etc.) ist daher zusätzlicher Synchronisationsaufwand notwendig.
- `destroy()`
  Wird vom Servlet-Container bei der Deaktivierung aufgerufen. Zuvor haben alle noch laufenden Threads Gelegenheit ihre Ausführung normal zu beenden. Diese Methode sollte beanspruchte Ressourcen freigeben und ggf. Zustände persistent sichern. Nach Ablauf von `destroy()` erfolgen keine Aufrufe an `service()` mehr durch den Servlet Container.

Ergänzend existieren noch die beiden Methoden `getServletConfig()` zum Zugriff auf die bei der Initialisierung übergebenen Parameter, und `getServletInfo()` zur Rückgabe einer beliebigen Zeichenkette (zumeist für Copyrightinformation o.ä. benutzt).

Die Servlet API gliedert sich in drei Stufen. Die Schnittstelle `Servlet` definiert alle notwendigen Verwaltungsoperationen, die innerhalb eines Servlets durch Methoden zu implementieren sind. Gleichzeitig liefert die diese Operationen umsetzende abstrakte Klasse `GenericServlet` eine erste einfache Basisimplementierung dieser Operationen. Die Spezifikation empfiehlt konkrete protkollspezifische Servletimplementierungen immer als Spezialisierung dieser Klasse zu realisieren. Für das populäre HTTP-Protokoll liefern die Standardimplementierungen mit `HttpServlet` eine solche Implementierung mit. Abbildung 1 stellt die Struktur der Basis API in der Übersicht zusammen.

Abbildung 14: *Struktur der Servlet-API*



(click on image to enlarge!)

Struktur eines Servlets:
Die Schnittstelle `Servlet` definiert alle Methoden, die ein Servlet zwingend implementieren muß. Dies kann durch eigenen Code geschehen, oder durch Erben von einer der vorgegebenen Klassen wie `GenericServlet` oder dem gebräuchlicheren `HttpServlet`.
Durch die Schnittstelle werden bereits Primitive für die beiden möglichen Kommunikationsrichtungen eingeführt: `ServletRequest` (Kommunikation vom Client zum Server) und `ServletResponse` (inverse Kommunikationsrichtung; Server an Client), als Parameter eines durch ein Servlet offerierten services. Beide Primitiven sind wiederum als Schnittstellen definiert, die durch konkrete Implementierungen (wie `HttpServletRequest` bzw. `HttpServletResponse`) umgesetzt werden müssen.

Die Schnittstelle **`ServletRequest`** ermöglicht es dem Servlet auf Parameternamen, Protkolldetails und Rechnernamen der sendenden Maschine zuzugreifen.
Ferner stellt die Schnittstelle den `ServerInputStream` und den `PrintReader` zur Verfügung, über den Daten des Clients (beispielsweise per HTTP POST oder PUT) empfangen werden können.

Die Schnittstelle **`ServletResponse`** erlaubt es dem Servlet Verwaltungsinformation, wie MIME-Typ und Content-Länge, der Antwort zu setzen.
Darüberhinaus stellt es mit dem `ServletOutputStream` einen Kommunikationskanal für binäre

Antwortdaten zur Verfügung. Unicode Text-artige Antworten können per `PrintWriter` übermittelt werden.

Die abstrakte **Klasse `HttpServlet`** stellt die notwendigen Methoden zur Bedienung von Client-Anforderungen bereit. Mindestens eine dieser Methoden ist zu überschreiben, um das gewünschte Verhalten des Servlets zu implementieren.

Die Implementierung des Beispiels 102 zeigt ein einfaches Servlet, welches nach einem Aufruf ein HTML-Dokument erzeugt und zurückliefert.

Beispiel 102: Ein einfaches Servlet

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)import java.util.Date;
(4)import javax.servlet.ServletException;
(5)import javax.servlet.http.HttpServlet;
(6)import javax.servlet.http.HttpServletRequest;
(7)import javax.servlet.http.HttpServletResponse;
(8)
(9)public class HelloWorld extends HttpServlet {
(10)    public void doGet(HttpServletRequest request, HttpServletResponse response)
(11)            throws IOException, ServletException {
(12)            response.setContentType("text/html");
(13)            PrintWriter out = response.getWriter();
(14)            out.println("<html>");
(15)            out.println("<head>");
(16)            out.println("<title>Hello World!</title>");
(17)            out.println("</head>");
(18)            out.println("<body>");
(19)            out.println("<h1>Hello World!</h1>");
(20)            out.println("<p>Current Date: "+new Date().toString()+"</p>");
(21)            out.println("</body>");
(22)            out.println("</html>");
(23)    }
(24)}
```

Download des Beispiels

Im Beispiel wird die `goGet`-Methode überschrieben, d.h. das Servlet reagiert auf HTTP GET-Anfragen. Als Antwort (ein wohlgeformtes HTML-Dokument) wird über den `PrintWriter` als Text (MIME-Typ `text/html`) versandt.

Der Servlet Container initialisiert und übergibt jedem Aufruf der Methode `doGet` ein Objekt des Typs `HttpServletRequest` und eines des Typs `HttpServletRespone` (welches im vorhergehenden Beispiel zur Übermittlung HTML-codierter Ergebnisse genutzt wurde.
Der Übergabeparameter des Typs `HttpServletRequest` ermöglicht den Zugriff auf die mit dem Aufruf des Servlets versandten Parameter. Beispiel 103 zeigt den Fall eines HTTP-basierten Servlet, welches die Parameter des `GET`-Aufrufs eingebettet in die URL erhält. Ein möglicher Aufruf enthält daher neben der Lokation des Dienstes auch die notwendigen Aufrufparameter: `http://www.example.com/addService?a=1&b=2`.

Beispiel 103: Ein einfacher Dienst

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)
(4)import javax.servlet.http.HttpServlet;
(5)import javax.servlet.http.HttpServletRequest;
(6)import javax.servlet.http.HttpServletResponse;
(7)
(8)public class SimpleService extends HttpServlet {
(9)     public void doGet(HttpServletRequest req, HttpServletResponse resp)
(10)            throws IOException {
(11)            resp.setContentType("text/xml");
(12)            PrintWriter out = resp.getWriter();
(13)            out.println(
(14)                    Integer.parseInt(
(15)                            req.getParameter("a")
(16)                                    + Integer.parseInt(req.getParameter("b"))));
(17)    }
(18)}
```

Download des Beispiels

Neben den bisher genutzten `GET`-Anfragen stehen im Rahmen der Standard-API auch Methoden zur

Behandlung der übrigen HTTP-Methoden zur Verfügung. Beispiel 104 zeigt die Funktionalität zum Zugriff auf Parameter im Rahmen eines DELETE-Aufrufs, auf HTTP-Headerdaten im Rahmen eines GET-Aufrufs sowie die Extraktion von Nutzdaten eines POST-Aufrufs.

Beispiel 104: Verschiedene Servletmethoden

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)import java.util.HashMap;
(4)
(5)import javax.servlet.ServletInputStream;
(6)import javax.servlet.http.HttpServlet;
(7)import javax.servlet.http.HttpServletRequest;
(8)import javax.servlet.http.HttpServletResponse;
(9)
(10)public class Methods extends HttpServlet {
(11)    private HashMap hm;
(12)    public void init() {
(13)            hm = new HashMap();
(14)    }
(15)
(16)    public void doDelete(HttpServletRequest req, HttpServletResponse resp) {
(17)            hm.remove(req.getParameter("key"));
(18)    }
(19)
(20)    public void doGet(HttpServletRequest req, HttpServletResponse resp)
(21)            throws IOException {
(22)            resp.setContentType("text/html");
(23)            PrintWriter out = resp.getWriter();
(24)            out.println(hm.get(req.getHeader("key")));
(25)    }
(26)
(27)    public void doPost(HttpServletRequest req, HttpServletResponse resp) {
(28)            byte line[] = new byte[100];
(29)            String key = null;
(30)            String value;
(31)            ServletInputStream sip = null;
(32)            try {
(33)                    sip = req.getInputStream();
(34)            } catch (IOException e) {
(35)                    e.printStackTrace();
(36)            }
(37)            try {
(38)                    sip.readLine(line, 0, 100);
(39)            } catch (IOException e1) {
(40)                    e1.printStackTrace();
(41)            }
(42)            String lineStr = new String(line, 0, line.length);
(43)            key = lineStr.substring(lineStr.indexOf(":") + 1, lineStr.indexOf("&"));
(44)            value =
(45)                    lineStr.substring(lineStr.lastIndexOf(":") + 1, lineStr.length());
(46)
(47)            hm.put(key, value);
(48)    }
(49)}
```

Download des Beispiels

Die Aufrufe erfolgen im HTTP-üblichen Stil:

- **GET**
  ```
  GET /HashMapApp HTTP/1.0
  key:name
  ```
- **POST**
  ```
  POST /HashMapApp HTTP/1.0

  key:name&value:mario
  ```
- **DELETE**
  ```
  DELETE /HashMapApp?key=name
  ```

Prinzipiell ist der Servletgedanke auch auf beliebige andere Protokolle und deren Operationen übertragbar. Jedoch wird er in der Praxis zumeist für HTTP eingesetzt. Die API ist jedoch generell so gestaltet, daß anwenderdefiniert jederzeit zusätzliche potokollspezifische Spezialisierungen der Klasse GenericServlet definiert werden können.

4.4 Web Services

Motivation und Hintergrund

Stand Datenaustausch in der Betrachtung der Metasprache XML bisher nahezu ausschließlich im Zeichen datei- oder strombasierter Integration, so eröffnet das Einsatzgebiet der *Web Services* den Einsatz XML-basierter Sprachen zur Abwicklung Internet-basierter online Kommunikation zwischen Anwendungssystemen.

Dem Begriff *Web Service* war in jüngerer Zeit eine bemerkenswerte Karriere beschieden, so daß es ihm gelang in der Praxis beachtliches Interesse auf sich zu ziehen. Allerdings fehlt bisher eine einheitliche Definition dieses Terminus hinsichtlich Zielsetzung und technischer Inhalte. Vielmehr versuchten und versuchen namhafte Herstelle durch eine Reihe verschiedener Definitionen, die am Markt offen zueinander in Konkurrenz treten, bisherige Techniken mit dem Ansatz der Web Services zu verschmelzen.
Jedoch führt(e) diese Vorgehensweise weder zu einer einheitlichen Begriffsübereinkunft und daher in der Folge auch zu keiner brauchbaren Abgrenzung gegenüber existierenden Techniken --- wie CORBA, RMI oder DCOM --- im Umfeld.
Nachfolgend wird daher eine an die Ergebnisse der Web Service Architecture-Arbeitsgruppe des World Wide Web Konsortiums angelehnte Definition zugrunde gelegt:

Definition 12: Web Service

Ein Web Service ist ein durch eine URI (gemäß RFC 2396) identifiziertes Softwaresystem, dessen öffentliche Schnittstellen und Protokollbindungen durch XML definiert und beschrieben sind. Diese Definitionen können durch andere Softwaresysteme ermittelt und bezogen werden. Auf der Basis der publizierten Schnittstellendefinitionen können diese Systeme dann mit dem Web Service in der durch die Schnittstelle festgelegten Weise interagieren. Diese Interaktion geschieht durch den Austausch XML-basierter Nachrichten, die mittels Internetprotokollen übertragen werden.

Der Begriff des *Service* ist hierbei durchaus in Anlehnung an den klassischen Dienstleistungsbegriff gemeint. Dieser beschreibt Handlungen, die im Auftrag eines Dritten vorgenommen werden, welche kein physisches Gut produzieren, sondern deren Charakter ausschließlich durch die Handlungserbringung selbst definiert ist.

Die Definition enthält ferner bereits Aussagen über die Komponenten einer Web Service basierten Architektur (einer sog. *serviceorientierten Architektur* (SOA)) und deren technischer Realisierung. Grundlegend Eigenschaften einer solchen Architektur ist neben der XML-Basiertheit die Verwendung von Universal Resource Identifkatoren zur eineindeutigen Benennung eines angebotenen Dienstes sowie die Nutzung der bestehenden Internetinfrastruktur zur Datenübertragung. Der Begriff *Internetinfrastruktur* soll hierbei nicht verengt auf die Techniken des WWW, d.h. HTTP auf Basis TCP/IP, verstanden werden, sondern durchaus im Sinne des der Internetprotokollhierarchie gebraucht werden.

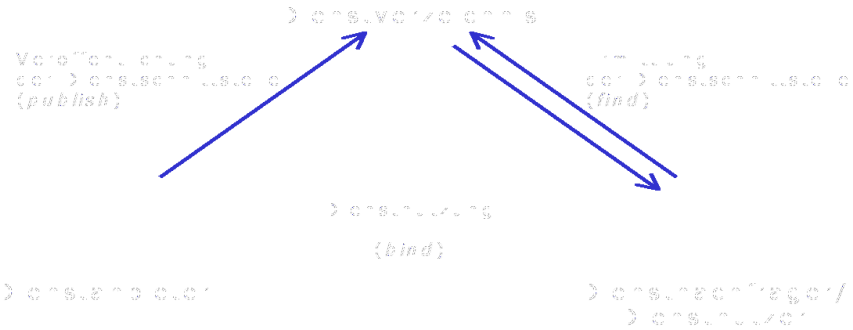
Gleichzeitig postuliert die Definition drei grundlegende Elemente einer serviceorientierten Architektur:

1. Ein XML-basierte Protokoll zur Darstellung und zum Austausch der Daten zwischen Diensten und ihren Nutzern.
2. Eine XML-basierte Beschreibungssprache zur Publikation von Dienstschnittstellen.
3. Einen Dienst zur Verwaltung der publizierten Schnittstellenbeschreibungen.

Diese drei aufeinander aufsetzenden Dienstschichten deuten auch bereits drei typische Rollen innerhalb einer serviceorientierten Architektur an:

1. Dienstanbieter.
   Er übernimmt die physische Bereitstellung des Dienstes, d.h. eine Implementierung und Publikation auf einem über das Internet zugänglichen Server.
   Zusätzlich kann er die Aufrufschnittstelle des Dienstes dokumentieren.
2. Dienstnachfrager/-nutzer.
   Er nutzt den angebotenen Dienst durch Übermittlung XML-codierter Nachrichten über Internetprotokolle.
   Zusätzlich kann er vor der Erstnutzung die Schnittstellenbeschreibung aus einem allgemein zugänglichen Dienstverzeichnis beziehen.
3. Dienstverzeichnis.
   Es verwaltet die durch die Dienstanbieter bereitgestellten kategorisierten Schnittstellenbeschreibungen.

Ausgehend von den Grundrollen und ihren Interaktionsbeziehungen ergeben sich die zwei in Abbildung 23 Abläufe für die Abwicklung einer Web Service-basierten Kommunikation.



Die Abbildung hebt die optionalen Schritte zur Ermittlung der Schnittstellenbeschreibung blau hervor. Liegt diese dem Aufrufwilligen bereits vor oder ist die Schnittstelle ihm bereits bekannt, so kann auf den Bezug der im Dienstverzeichnis abgelegten Beschreibungsdaten verzichtet werden und der Dienstaufruf direkt erfolgen.
Die Schnittstellenbeschreibung nimmt damit keine herausragende Rolle innerhalb der Architekturerstellung ein, wie dies beispielsweise für CORBA, RMI oder DCOM der Fall wäre (dort ist die Schnittstellenbeschreibung Teil des Entwicklungszyklus und muß zum Übersetzungzeitpunkt des Aufrufclients zwingend vorliegen).

Aus der Kombination der Architekturelemente und der Interaktionsszenarien haben sich drei Standards am Markt etabliert, deren Implementierungen die dargestellten Grundaufgaben innerhalb einer serviceorientierten Architektur erfüllen:

- SOAP --- Das Kommunikationsprotokoll zur Kommunikationsabwicklung zwischen Dienstanbieter und -nutzer.
- WSDL (Web Service Description Language) --- Die XML-basierte Schnittstellenbeschreibungssprache.
- UDDI (Universal Service Description, Discovery and Integration) --- Ein Verzeichnisdienst zur Verwaltung von Schnittstellenbeschreibungen, der selbst als Web Service realisiert ist.

Anmerkung: Ursprünglich wurde das Akronym *SOAP* zu *Simple Object Access Protocol* expandiert. Aufgrund der irreführenden Nähe zur objektorientierten Programmierung wurde jedoch im Verlauf des Standardisierungsprozesses beschlossen etablierte Akronym als Namen beizubehalten, jedoch von der weiteren Expansion abzusehen.

SOAP

SOAP, welches als Version 1.0 im Herbst 1999 als Ergebnis der Kooperation zwischen den Firmen *DevelopMentor*, *IBM*, *Microsoft*, *Lotus* und *UserLand Software* vorgestellt wurde markiert weniger den Beginn der Idee der Web Services, als vielmehr einen weiteren evolutionären Entwicklungsschritt.
Die Uridee die zur Abwicklung synchroner entfernter Funktionsaufrufe (*Remote Procedure Calls* (RPC)) zu übermittelnden Über- und Rückgabeparameter durch ein XML-Vokabular auszudrücken findet sich bereits im Protokoll *XML-RPC* verwirklicht.

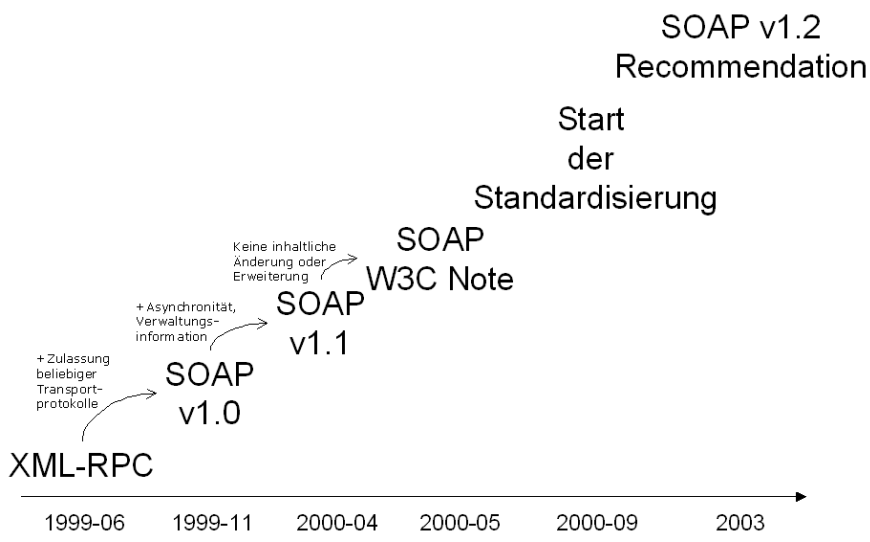
Ausgehend von diesem Ansatz definiert SOAP in der Version 1.0 ein vollständiges Protokoll, welches neben der Übertragung von Nutzdaten auch die Darstellung von dekriptiver Verwaltungsinformation vorsieht. Wie bereits für XML-RPC ist auch bei SOAP v1.0 ausschließlich die Verwendung von HTTP als Transportprotokoll definiert.

Diese Beschränkung wird durch die Anfang 2001 freigegebene SOAP Version 1.1 aufgehoben, welche SOAP als vollständig von der zu tatsächlichen Übertragung gewählten Protokollschicht entkoppelt und damit als eigenständige Protokollabstraktion etabliert. Gleichzeitig führt diese Version die Möglichkeit asynchroner Aufrufe ein.

Um den SOAP-Ansatz einer breiten Interessentenschicht zugänglich werden zu lassen und auch die Sensibilisierung für eine mögliche Standardisierung des Protokolls zu schaffen reichen die beteiligten Partner die Spezifikationsversion 1.1 unverändert als W3C Note ein.

Die im Herbst 2000 begonne Standardisierungsarbeit durch die W3C XML Protocol Arbeitsgruppe übernahm den eingereichten Spezifikationsvorschlag der Version 1.1 weitestgehend und führte, aus Gründen der Interoperabiltität behutsame Korrekturen sowie umfangreiche Erweiterungen ein. Mit der Verabschiedung des endgültigen W3C-Standards ist im Laufe des Jahres 2003 zu rechnen.

Die Abbildung 24 stellt nochmals die einzelnen SOAP-Versionen, ihre Unterschiede und den Vorläufer XML-RPC in der chronologischen Übersicht zusammen.



Aufbau einer SOAP-Nachricht

Jede SOAP-Nachricht, unabhängig davon ob es sich um eine synchron oder asynchron übermittelte Anfrage oder Antwort handelt besteht generell aus zwei Teilen:

1. Dem Rumpfelement (`Body`) zur Aufnahme der zu übermittelnden Nutzdaten.
2. Optional einem oder mehreren Kopfelementen (`Header`) zur Aufnahme von Metainformation.

Das Beispiel 105 zeigt einen vollständigen SOAP-Aufruf.
Er besteht aus einem *Umschlag* (dargestellt durch das Element `Envelope`), der das beschreibende Kopfelement `altertcontrol` sowie die im `Body`-Element zusammengefaßten Nutzdaten enthält.

Alle durch den Standard vorgegebenen Elemente und Attribute sind dem Namensraum `http://www.w3.org/2003/05/soap-envelope` zugeordnet und alle Anwenderdefinierten den entsprechenden eigenen Namensräumen.

Beispiel 105: Ein vollständiger SOAP-Aufruf

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2) <env:Header>
(3)  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
(4)   <n:priority>1</n:priority>
(5)   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
(6)  </n:alertcontrol>
(7) </env:Header>
(8) <env:Body>
(9)  <my:message xmlns:my="http://www.mycompany.com">
(10)   <my:text>Hello World!</my:text>
(11)  </my:message>
(12) </env:Body>
(13)</env:Envelope>
```

Die Aufgabe der *SOAP-Header* ist es Verwaltungsinformation aufzunehmen, die nicht ausschließlich durch den Empfänger des SOAP-Aufrufes zu verarbeiten ist und die nicht Teil der Nutzdaten ist.
Beispiele hierfür sind Daten über das Routingverhalten, gesetzte Sperren oder Transaktionen aber auch Zugriffsdaten wie Nutzername und Passwort (diese natürlich verschlüsselt!).

Die Verarbeitung eines Kopfelements ist vorgabegemäß optional, außer das zusätzliche gesetzte Attribut `mustUnderstand` weißt den Wert `1` oder `true` (die beiden durch XML Schema zugelassenen lexikalischen Repräsentationen für logisch *wahr*) auf. In diesem Falle muß ein *SOAP-Knoten* das Kopfelement verarbeiten. Kann die Verarbeitung nicht vorgenommen werden, so muß an den Sender der SOAP-Nachricht eine Fehlermeldung übermittelt werden.
Ein *SOAP-Knoten* ist hierbei jeder Rechner entlang des Nachrichtenpfades zwischen Sender und Empfänger, der das SOAP-Protokoll verarbeiten kann.

Soll die Verarbeitung der SOAP-Kopfelemente auf einen bestimmten Rechner (beispielsweise einen zwischenspeichernden Proxy-Knoten) beschränkt werden, so kann durch das im Standard vorhergesehene Attribut `role` eine eineindeutige Adressierung eines durch URI identifizierten (Zwischen-)Knotens erreicht werden.
Das Beispiel 106 zeigt die Nutzung des `role`- und des `mustUnderstand`-Attributs.

Beispiel 106: Nutzung der SOAP-Kopfelemente

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2)     <env:Header>
(3)     <abc:Extension1
(4)             xmlns:abc="http://example.org/2001/06/ext"
(5)             env:mustUnderstand="true"
(6)             role="http://www.example.com/xyz"/>
(7)     <def:Extension2 xmlns:def="http://example.com/stuff"
(8)             env:mustUnderstand="true"
(9)             env:role="http://www.w3.org/2003/05/soap-envelope/role/next"/>
(10)    </env:Header>
(11)    <env:Body>
(12)            ...
(13)    </env:Body>
(14)</env:Envelope>
```

Das Beispiel zeigt die Nutzung zweier Kopfelemente (`Extension1` und `Extension2`) in „eigenen“, d.h. nicht den im Standard vorgesehenen, Namensräumen.
Für beide Kopfelemente ist das Attribut `mustUnderstand` mit `true` belegt, was sie als zwingend zu prozessieren ausweist.
Zusätzlich ist für `Extension2` das Attribut `role` auf `http://www.example.com/xyz` gesetzt. Diese Belegung charakterisiert die durch `mustUnderstand` formulierte verpflichtende Verarbeitung näher. Im vorliegenden Falle muß sie ausschließlich durch den mit der URI `http://www.example.com/xyz` bezeichnenden Zwischenknoten erfolgen, für alle anderen Knoten des Nachrichtenpfades --- einschließlich des endgültigen Empfängers --- kann dieses Kopfelement ignoriert werden. Der dieses Kopfelement verarbeitende SOAP-Knoten darf es nach erfolgreicher Prozessierung aus der SOAP-Nachricht entfernen.
Bei der für das zweite Kopfelement (`Extension2`) gewählten Rolle (`http://www.w3.org/2003/05/soap-envelope/role/next`) handelt es sich um eine durch die SOAP-Spezifikation vordefinierte URI, welche alle Knoten entlang des Nachrichtenpfades einschließlich dem endgültigen Empfänger selbst bezeichnet. In diesem Fall darf das Kopfelement, selbst nach erfolgreicher Verarbeitung durch einen Knoten nicht entfernt werden, da weitere Verarbeitungsschritte durch andere Knoten folgen können. (Sofern es sich nicht um den endgültigen Empfänger handelt, müssen sogar weitere Verarbeitungsschritte folgen.)

Neben der dargestellten Sonderbelegung des Rollenattributes definiert der SOAP-Standard des W3C mit `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` eine Attributbelegung, die Kopfelemente kennzeichnet die ausschließlich durch den endgültigen Empfänger einer Nachricht verarbeitet werden dürfen und durch `http://www.w3.org/2003/05/soap-envelope/role/none` Kopfelemente, die durch einen SOAP-Knoten nicht verarbeitet werden dürfen.

Durch die Kopfelemente können Einzelknoten entlang des Vermittlungspfades einer SOAP-Nachricht pauschal oder gezielt zu einer anwenderdefinierten Verarbeitung angeregt werden. Die aktiven Zwischenknoten übernehmen hierbei nicht mehr nur reine vermittelnde Aufgaben auf den tieferliegenden (Transport-)Protokollschichten, sondern werden zu aktiven Elementen der SOAP-Nachrichtenkette. Aufgrund ihrer Stellung im Nachrichtenpfad zwischen Sender und endgültigem Empfänger werden sie auch mit dem Begriff *SOAP Intermediäre* belegt.

Die Übertragung der zwischen Dienstnutzer und Dienst ausgetauschten Daten (d.h. Aufruf- und Rückgabeparameter oder Einwegbotschaft) erfolgt durch das `Body`-Element der SOAP-Nachricht. Beispiel 107 zeigt den erzeugten SOAP-Datenstrom zum Aufruf eines Dienstes der die beiden `int`-Parameter *a* und *b* addiert und das Ergebnis zurückliefert.

Beispiel 107: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:env="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService>
(5)     <env:Body>
(6)             <b:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
(7)             <b:a>2</b:a>
(8)             <b:b>3</b:b>
(9)             </b:add>
(10)    </env:Body>
(11)</env:Envelope>
```

Innerhalb des `Body`-Elements werden die benannten Parameter durch jeweils ein Element, das den Namen des Parameters trägt, dargestellt. Jedes Parameterelement enthält die lexikalische Repräsentation des zu übertragenden Wertes. Der aufgerufene Dienst (im Beispiel: *add*) fungiert als gemeinsames Elternelement der einzelnen Parameterelemente.
Zur Unterscheidung der verschiedenen Vokabularelemente des SOAP-Aufrufes müssen alle anwenderdefinierten Teile des `Body`-Elements einem eigenen, vom SOAP-Vorgabenamensraum

abweichenden, Namensraum zugeordnet sein. Dies geschieht vor dem Hintergrund sowohl der gewünschten Abgrenzung von den durch den Standard vorgegebenen Elementen und Attributen, als auch mit der Zielsetzung zur Validierung der SOAP-Aufrufe eine XML-Schemainstanz einsetzen zu können. Hierzu sind die abweichenden Namensräume zwingend erforderlich, um mithilfe des any-Elements die Validierung der durch den SOAP-Standard strukturell nicht festlegbaren Body-Kindelemente zu ermöglichen.
Die Struktur dieser Kindelemente orientiert sich ausschließlich an der XML-Darstellung der für einen Service notwendigen Parameter und kann daher im allgemeinen Falle nicht durch den Standard vorgegeben werden. Aus diesem Grunde legt die SOAP-Spezifikation lediglich Encodierungsregeln zur Transformation hauptspeicherresidenter Objekte und Strukturen in eine XML-Darstellung fest.
Das Beispiel verwendet diese vordefinierten Encodierungsregeln, welche die Abbildung allgemeiner Objektgraphen in XML-Strukturen gestatten. Neben dieser, an der Belegung des Attributs encodingStyle mit `http://www.w3.org/2003/05/soap-encoding` erkennbaren, Serialisierungsform können durch den Anwender beliebige eigene Regeln zur Gewinnung der XML-Darstellung festgelegt werden. Einzig das im Standard definierte SOAP-Encoding muß jedoch zwingend von allen Implementierungen unterstützt werden.

In ähnlicher Darstellungsform der Anfrage findet sich auch die Übermittlung der durch den aufgerufenen Dienst berechneten Resultate codiert:

Beispiel 108: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService">
(5)     <env:Body>
(6)             <b:result env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">5</b:
result>
(7)     </env:Body>
(8)</env:Envelope>
```

Beispiel 108 zeigt die rückübermittelte Antwort des Addier-Dienstes.
Auch sie folgt den selben Struktur- und Erstellungsprinzipien. Daher weist auch Antwortnachricht die Trennung in (optionale und in diesem Beispiel nicht genutzte) Kopfelemente und nutzdatentragende Rumpfelemente als Kindelemente des Body-Elements auf.
Die Erstellung der XML-Darstellung erfolgt, wie bereits für den Aufruf, gemäß festgelegter bzw. anwenderdefiniert festlegbarer Richtlinien einer Encodierungsdarstellung. Auch in diesem Beispiel wurde die Standardencodierung gewählt, weshalb das encodingStyle-Attribut abermals mit dem Wert `http://www.w3.org/2003/05/soap-encoding` versehen ist.
Zur Abgrenzung von den durch den W3C-Standard vordefinierten Elementen muß auch zur Darstellung der Antwort eines Dienstaufrufs jedes Kindelement des Body-Knotens einen vom SOAP-Standard abweichenden Namensraum besitzen.
Die Organisation der Darstellung des Dienstergebnisses als XML-Dokument ist dem Diensterbringer überlassen. Im Beispiel wird ein Element `result` übertragen.

Quellcode des Web Service
Quellcode eines Web Serviceaufrufes

Fallstudie: Implementierung eines Web Services

Die nachfolgende Fallstudie betrachteten einen einfachen Web Service zur Realisierung der mathematischen Operationen auf dem Körper der komplexen Zahlen.
Die Diskussion wird an der frei verfügbaren SOAP-Implementierung Axis geführt, welche im Rahmen des Apache-Projekts gepflegt wird. Axis stellt eine vollständig in Java realisierte Bibliothek zur Umsetzung von Web Services die als Java-Klassen angeboten werden zur Verfügung.
Zur Ausführung wird eine beliebige Servlet-Umgebung benötigt. Für das Beispiel wurde die ebenfalls kostenfrei verfügbare Jakarta-Tomcat eingesetzt, welche als Ausführungsumgebung der Web Services dient.

Implementierung des Web Service:

Beispiel 109: Beispielwebdienst

```
(1)public class Complex {
(2)     private double re;
(3)     private double im;
(4)
(5)     public Complex(double re, double im) {
(6)             this.re = re;
(7)             this.im = im;
(8)     }
(9)     public Complex() {
(10)            this.re = 0;
(11)            this.im = 0;
(12)    }
(13)
(14)    public void setRe(double re) {
(15)            this.re = re;
(16)    }
(17)    public void setIm(double im) {
(18)            this.im = im;
(19)    }
(20)    public double getIm() {
(21)            return im;
(22)    }
(23)    public double getRe() {
(24)            return re;
(25)    }
(26)    public double modulus(Complex c1) {
(27)            return Math.sqrt(Math.pow(c1.getRe(), 2) + Math.pow(c1.getIm(), 2));
(28)    }
(29)    public Complex add(Complex c1, Complex c2) {
(30)            return new Complex(
(31)                    c1.getRe() + c2.getRe(),
(32)                    c1.getIm() + c2.getIm());
(33)    }
(34)    public Complex mult(Complex c1, Complex c2) {
(35)            return new Complex(
(36)                    c1.getRe() * c2.getRe() - c1.getIm() * c2.getIm(),
(37)                    c1.getRe() * c2.getIm() + c1.getIm() * c2.getRe());
(38)    }
(39)    public Complex div(Complex c1, Complex c2) {
(40)            double div = Math.pow(c2.getRe(), 2) + Math.pow(c2.getIm(), 2);
(41)            return new Complex(
(42)                    (c1.getRe() * c2.getRe() + c1.getIm() * c2.getIm()) / div,
(43)                    (c1.getIm() * c2.getRe() - c1.getRe() * c2.getIm()) / div);
(44)    }
(45)    public Complex pow(Complex c1, int n){
(46)            if (n==0) {
(47)                    return new Complex(1,0);
(48)            }
(49)            Complex result = c1;
(50)            for (int i=1;i<n;i++){
(51)                    result = result.mult(c1, c1);
(52)            }
(53)            return result;
(54)    }
(55)    public String toString() {
(56)            String result = new String();
(57)            result += re;
(58)            if (im < 0) {
(59)                    result += "-";
(60)            } else {
(61)                    result += "+";
(62)            }
(63)            result += "i" + Math.abs(im);
(64)            return result;
(65)    }
(66)}
```

Beispiel 109 zeigt die Implementierung der Klasse `Complex` deren Methoden als Web Service angeboten werden.
Augenscheinlich unterschiedet sich die Implementierung als Web Service nicht von der, die für die statische lokale Bereitstellung leistenden.
Bei der Umsetzung ist jedoch darauf zu achten, auf `this`-Verweise in Methodenrümpfen zu verzichten, da kein Objektkontext durch das SOAP-Protokoll mitversandt wird. Als pragmatisches Hilfsmittel zur

Einhaltung dieser Einschränkung bietet es sich an die als Dienst bereitzustellenden Methoden mit dem Schlüsselwort `static` zu versehen.
Zusätzlich ist auf die Definition eines parameterlosen Vorgabekonstruktors zu achten, da dieser serverseitig zur durch das Framework zur Objekterzeugung herangezogen wird.

Die serverseitige Bereitstellung des Dienstes kann im Rahmen des Axis-Frameworks auf zwei Arten geschehen. Zum einen durch Bereitstellung des Quellcodes aus Beispiel 109 in einem dafür gesondert vorgesehen Serververzeichnis (typischerweise `../tomcat/webapps/axis`). Zusätzlich ist die Datei mit der Extension `jws` (statt dem üblichen `java`) zu versehen. Zum Aufrufzeitpunkt wird der Quellcode durch das Axis-Framework für den Aufrufer transparent übersetzt und zur Ausführung gebracht.
Andererseits sieht das Framework die Möglichkeit der Ablage von vorübersetztem Java-Code vor. Dieser in der Axis-Terminologie als *Web Service Deployment* bezeichnete Vorgang gestattet dem Anwender weiterrechenden Einfluß auf Umstände der Dienstbereitstellung als die zuvor gezeigte Vorgehensweise.

Notwendige Voraussetzung des Deploymentvorganges ist die Bereitstellung einer separaten Konfigurationsdatei, dem sog. *Deployment-Deskriptor* unter zwingend vorgegebenen dem Dateinamen `deploy.wsdd`. Beispiel 110 zeigt eine solche Konfigurationsdatei für den Beispieldienst.

Beispiel 110: Deploymentdeskriptor des Beispieldienstes

```
(1)<deployment
(2)     xmlns="http://xml.apache.org/axis/wsdd/"
(3)     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
(4)   <service name="ComplexCalc" style="RPC">
(5)     <parameter name="className" value="Complex"/>
(6)      <parameter name="allowedMethods" value="add mult modulus div pow"/>
(7)      <beanMapping
(8)             qname="myNS:Complex"
(9)             xmlns:myNS="urn:Complex"
(10)            languageSpecificType="java:Complex"/>
(11)    </service>
(12)</deployment>
```

Innerhalb der Konfigurationsdatei werden Aussagen über das Interaktionsmuster, welches der Kommunikation mit dem Dienst zugrunde liegt. Im Beispiel wird durch die Belegung des Attributs `style` mit dem Wert `RPC` synchrone Kommunikation im Stile entfernter Methodenaufrufe gewählt.
Zusätzlich kann ein vom Namen des Compilats abweichender Dienstname frei festgelegt werden (im Beispiel `ComplexCalc` als Belegung des Attributs `name` des `service`-Elements).
Innerhalb der Kindelemente des Elements `service` wird die dienstimplementierende Klasse identifiziert (`parameter`-Element dessen `name`-Attribut den Wert `className` trägt, das zugehörige `value`-Attribut desselben Elements enthält dann den vollqualifizierten Klassennamen) sowie die per SOAP zugreifbaren Methoden mit separierenden Leerzeichen gemäß XML-Konvention aufgelistet (im Beispiel `add mult modulus div pow`).
Hierdurch eröffnet die Konfiguration des Webdienstes durch den Deploymentdeskriptor bereits einen Eingriffspunkt, der bei der reinen serverseitigen Ablage des Quellcodes, mit automatischer Freigabe aller öffentlich zugänglichen Methoden, nicht bestand.
Ferner können vermöge des Elements `beanMapping` anwenderdefinierte strukturierte Datentypen, die als Java-Bean-konforme Klasse umgesetzt sind definiert werden. Im Beispiel ist dies die Klasse zur Implementierung der komplexen Zahl.

Die Installation und Konfiguration (Deployment) des Web Service geschieht durch Aufruf des Administrationswerkeuges `java org.apache.axis.client.AdminClient deploy.wsdd`.
Zusätzlich ist die Dienstklasse im Verzeichnis `.../tomcat/webapps/axis/WEB-INF/classes` abzulegen.

Implementierung des Web Service Aufrufers:

Beispiel 111: Aufruf des Beispielwebdiensts

```
(1)import java.rmi.RemoteException;
(2)import javax.xml.namespace.QName;
(3)import javax.xml.rpc.ParameterMode;
(4)import javax.xml.rpc.encoding.XMLType;
(5)import org.apache.axis.client.Call;
(6)import org.apache.axis.client.Service;
(7)import org.apache.axis.encoding.ser.BeanSerializerFactory;
(8)import org.apache.axis.encoding.ser.BeanDeserializerFactory;
(9)
(10)public class callService {
(11)    public static void main(String[] args) {
(12)            Call call = new Call( new Service() );
(13)
(14)            QName qn = new QName("urn:Complex","Complex");
(15)
(16)            call.setTargetEndpointAddress("http://10.0.0.1:8080/axis/services/
```

```
ComplexCalc");
(17)            call.setOperationName("pow");
(18)            call.registerTypeMapping(Complex.class, qn,
(19)                                     new BeanSerializerFactory(Complex.class, qn),
(20)                                     new BeanDeserializerFactory(Complex.class, qn));
(21)
(22)            call.addParameter("c1", qn, ParameterMode.IN);
(23)            call.addParameter("n", XMLType.XSD_INT, ParameterMode.IN);
(24)            Object[] param = new Object[2];
(25)            param[0] = new Complex(2,3);
(26)            param[1] = new Integer(3);
(27)
(28)            call.setReturnType( qn );
(29)
(30)            try {
(31)                    Complex result = (Complex) call.invoke(param);
(32)                    System.out.println("result=" + result);
(33)            } catch (RemoteException re) {
(34)                    re.printStackTrace();
(35)            }
(36)    }
(37)}
```

Beispiel 111 zeigt die Implementierung eines Aufrufs des Beispielwebdienstes.
Zentrales Objekt des Aufrufs eines Web Service ist die Klasse `Call`. Sie kapselt die gesamten zur Kommunikation notwendigen Daten und übernimmt den synchronen Aufruf des entfernten Dienstes. Zur Verwaltung von Verbindungsdaten, die für verschieden Einzelaufrufe desselben Dienstes gleichermaßen Verwendung finden können diese zusätzlich durch einen `Service` repräsentiert werden. Im Beispiel wird hierauf jedoch aus Übersichtlichkeitsgründen verzichtet und alle alle Einstellungen direkt für das `Call`-Objekt vorgenommen.
Zu diesen Einstellungen zählt die dargestellte Angabe des *Service Endpunktes*, derjenigen Adresse, die konform zum gewählten Übertragungsprotkoll (im Beispiel ist dies HTTP) die physische Übergabestelle des SOAP-Aufrufs an den Web Service identifiziert.
Daher ist bei Änderung der Dienstbereitstellung, etwa durch Verlagerung auf einen anderen Server oder auch die sich aus dem oben gezeigten geänderten Deployment ergebende Endpunktadresse, lediglich der Parameter der Methode `setTargetEndpointAddress` geeignet anzupassen.

Zusätzlich zur Angabe der aufzurufenden Operation innerhalb des angesprochenen Dienstes, durch die Methode `setOperationName` erfolgt durch `addParameter` die Definition der Übergabeparameter des Dienstes.
Hierbei muß jeder Übergabeparameter in Name, Typ und Übergabeart spezifiziert werden. Im Beispiel sind dies die beiden Parameter `c1` und `n`, die als Eingabe (d.h. nur zum lesenden Zugriff) der Methode `pow` dienen.
Diese Methode implementiert die Potenzierung komplexer Zahlen durch fortwährende Multiplikation. Ihre Signatur erfordert daher die Übergabe der komplexen Basis sowie des ganzzahligen Exponenten.

Zur Übertragung der speicherresidenten Wertdarstellung ordnet das Axis-Framework den Java-Primtivdatentypen und einigen ausgewählten als Klasse realisierten Datentypen eindeutige Abbildungen in die in XML-Schema definierten Datentypen zu.
Die Reihenfolge der Aufrufe der `addParameter`-Methode muß der Auftrittsreihenfolge in der Signatur des aufzurufenden Dienstes entsprechen, um die Kompatibilität zur Programmiersprachen, die ausschließlich über Stellungsparameter verfügen zu wahren.
Tabelle 27 stellt die durch das Axis-Framework Typäquivalenzen zusammen:

Tabelle 27: Abbildung der Java-Datentypen auf XML-Schema

Java	XML-Schema	Bemerkung
byte[]	base64Binary hexBinary	Beide Darstellungsweisen sind gleichwertig möglich.
boolean	boolean	
byte	byte	
java.util.Calendar	dateTime	
java.math.BigDecimal	decimal	
double	double	
float	float	
int	int	
java.math.BigInteger	integer	
long	long	
javax.xml.namespace.QName	QName	Diese Javaklasse ist Bestandteil der API-Erweiterung für XML, die separat bezogen werden muß.

short	short
java.lang.String	string

Bei der zu übergebenden komplexen Zahl handelt es sich um eine anwenderdefinierte Klasse, die einen neuen Java-Typ etabliert für den (naturgemäß) keine Entsprechung in XML-Schema zur Verfügung steht. Aus diesem Grund muß auf Seiten des Aufrufers und des aufgerufenen Dienstes eine eigene Abbildungsvorschrift für die Erzeugung der SOAP-Darstellung bereitgestellt werden.

Durch die Axis-Serviceausführungsumgebung kann automatisiert eine solche Abbildung erzeugt werden, sofern der anwenderdefinierte Typ (d.h. die implementierende Klasse) gemäß der Java-Beans-Programmierkonvention umgesetzt ist. Diese ist eingehalten, sobald für jedes datenhaltende Attribut eine get- und set-Methode umgesetzt wird. Hintergrund dieses Vorgehens ist die Möglichkeit der Abfrage aller objektinternen Datenfelder durch Nutzung der Java-Reflection-API.
Die hierfür notwendige Zuordnung zwischen neuem Java- und XML-Typ geschieht im durch Beispiel 110 dargestellten Deploymentdeskriptor ab Zeile sieben. Dort gibt das Attribut `qname` den Namen des XML-Typs und `languageSpecificType` den Namen der implementierenden Java-Klasse (`Complex`) an.

Entsprechend vollzieht der Dienstaufrufer die Typabbildung im Code nach. Hierzu muß ihm der gewählte Name des neuen XML-Typen bekannt sein (im Beispiel: `Complex` im Namensraum `urn:Complex`). Durch den Aufruf der Methode `registerTypeMapping` gibt er dem Framework die Implementierungen der beiden Abbildungsrichtungen (Java-Hauptspeicherobjekte in SOAP bzw. SOAP zu Hauptspeicherobjekten) bekannt. Das Beispiel verwendet hierzu die im Axis-Framework mitgelieferten Klassen `BeanSerializerFactory` und `BeanDeserializerFactory` welche das Gewünschte auf Basis einer als Java-Bean codierten Klasse leisten.

Die Festlegung der tatsächlichen Übergabewerte erfolgt im Beispiel durch Erzeugung eines Arrays zur Aufnahme von `Object`-typisierten Elementen in die die zuvor hinsichtlich ihres Typs definierten Aktualparameter eingefügt werden.
Aufgrund der Einschränkung der Programmiersprache Java (bis zur Version 1.4), die Ausprägungen von Primitivtypen nicht als Objekte behandeln kann, erfolgt im Beispiel die Kapselung des `int`-Wertes für `n` durch ein Objekt der Klasse `Integer`.

Für die Definition des Typs des erwarteten Rückgabewertes gelten dieselben Gesetztmäßigkeiten hinsichtlich Bekanntmachung und Zugriffsabwicklung. Die Definition des erwarteten Rückgabetyps erfolgt durch Aufruf der Methode `setReturnType`.

Der synchrone Aufruf des entfernten Dienstes wird dann durch die Methode `invoke` des erzeugten und entsprechend parametrisierten `Call`-Objektes vollzogen.
Dieser Aufruf ist blockierend und läßt den Aufrufer bis zum Eintreffen des berechneten Ergebnisses warten.

Die Rückgabe wird, trotz der Definition des spezielleren Rückgabetyps `Complex` generell als `Object`-Ausprägung geliefert um eine strikt typprüfbare Signatur der Methode `invoke` zu erhalten. Daher muß das durch den Web Service gelieferte Resultat geeignet, d.h. konform zur durch `setReturnType` getroffenen Festlegung, typgewandelt werden.

Beispiel 112 zeigt den SOAP-Datenverkehr beim Aufruf des Dienstes zur Berechnung der Potenz einer komplexen Zahl. Aufgrund des derzeitigen Entwicklungsstandes des Axis-Frameworks weicht der Leitungsmittschnitt von der zu Eingang dieses Kapitels vorgestellten SOAP-Struktur des W3C-Standards ab. Gegenwärtig (d.h. zum Zeitpunkt der Verfügbarkeit der Version 1.1) unterstützt das Framework nur eine Untermenge des neuen Standards und codiert die XML-Nachrichten noch nach dem Stand der Vorgängerspezifikation.

Beispiel 112: SOAP-Nachricht zum Aufruf des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <pow soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
(8)             <c1 href="#id0"/>
(9)             <n xsi:type="xsd:int">3</n>
(10)            </pow>
(11)            <multiRef
(12)                    id="id0"
(13)                    soapenc:root="0"
(14)                    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(15)                    xsi:type="ns1:Complex"
(16)                    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(17)                    xmlns:ns1="urn:Complex">
(18)            <im xsi:type="xsd:double">3.0</im>
(19)            <re xsi:type="xsd:double">2.0</re>
```

```
(20)            </multiRef>
(21)    </soapenv:Body>
(22)</soapenv:Envelope>
```

Im Code des Beispiels gut zu sehen ist die Darstellung der beiden Übergabeparameter `c1` und `n` für die Basis der Potenzierung und den ganzzahligen Exponenten. Für `n` wird die in Tabelle 112 Abbildung in den äquivalenten Typen `int` aus XML-Schema durchgeführt. Zusätzlich überträgt die durch Axis gewählte Codierung die Typisierung explizit mit (Attribut `xsi:type="xsd:int"`) wodurch die Typabbildung offenbar wird.
Für den nicht direkt nach XML-Schema transformierbaren strukturierten (Java-)Datentypen `Complex` wird durch den Serialisierungsmechanismus ein eigenes mit `multiRef` bezeichnetes und durch das Attribut `id` identifiziertes Element erzeugt. Dieses Vorgehen entspricht der durch die SOAP-Spezifikation vorgesehenen Serialisierungs allgemeiner Graphen, welche die speicherresidenten Datenobjekte als Knoten interpretiert und hierfür wiederverwendbare (der Name des Elements deutet dies an) Elemente erzeugt. Die tatsächliche Wiederverwendung innerhalb des SOAP-Stromes findet durch Mehrfachnennung des im `id`-Attribut festgelegten identifizierenden Wertes im `href`-Attribut eines Verweiselementes statt.
Im Beispiel wird die Berechnungsbasis als ein solches `multiRef`-Element ausgedrückt, welches die Wertbelegungen der durch get-Methoden zugänglichen Javafelder (`im` und `re`) enthält.
Das den Übergabeparameter `c1` repräsentierende XML-Element enthält daher lediglich im `href`-Attribut einen Verweis auf die `multiRef`-Struktur.

Beispiel 113 zeigt den für die Übermittlung des Dienstergebnisses übertragenen SOAP-Datenverkehr. Auch er weicht, aufgrund der Konformität zur älteren SOAP-Spezifikationsversion, geringfügig vom aktuellen Standard ab.

Beispiel 113: SOAP-Nachricht zur Übermittlung des Berechnungsergebnisses des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <powResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
(8)     <powReturn href="#id0"/>
(9)     </powResponse>
(10)    <multiRef
(11)            id="id0"
(12)            soapenc:root="0"
(13)            soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(14)            xsi:type="ns1:Complex"
(15)            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(16)            xmlns:ns1="urn:Complex">
(17)    <im xsi:type="xsd:double">12.0</im>
(18)    <re xsi:type="xsd:double">-5.0</re>
(19)    </multiRef>
(20)    </soapenv:Body>
(21)</soapenv:Envelope>
```

Wie bereits beim Aufruf realisiert wird die XML-Darstellung des strukturierten Datentyps `Complex`, der hier als Rückgabetyp dient, durch Nutzung der allgemeinen Graphenserialisierung erzeugt.

WSDL

Die Grundidee der Web Services fordert nicht zwingend die Darstellung der Dienstschnittstellen für Dritte, wie dies beispielsweise Verteilungsarchitekturen wie CORBA und DCOM zwingend als Teil des Entwicklungszyklus vorgeben. Dennoch erweist es sich auch für Web Services sinnvoll die Schnittstellenbeschreibungen in einem maschinenlesbaren Format bereitzustellen um Werkzeugen die automatisierte Verarbeitung zu ermöglichen. Hierunter fallen beispielsweise Entwicklungsumgebungen, die aus Schnittstellenbeschreibungen erste Rohgerüste für die Abwicklung der Dienstaufrufe generieren können. Zusätzlich stellen formalisiert dokumentierte Schnittstellen einen wichtigen Bestandteil der Dokumentation eines Softwaresystems dar.
Zur Schnittstellendokumentation von Web Services etabliert sich gegenwärtig der Standard der *Web Service Description Language*, der durch eine Arbeitsgruppe des World Wide Web Konsortiums vorangetrieben wird.

Eine WSDL-Beschreibung selbst ist eine XML-Datei, WSDL mithin als XML-Schema-basiertes XML-Vokabular realisiert und zerfällt in sechs Teile:

1. **Service**: Zur allgemeinen Beschreibung des angebotenen Dienstes
2. **Operations**: Zur Beschreibung der angebotenen Operationen
3. **Messages**: Zur Beschreibung Signatur der einzelnen Nachrichten (Aufrufe und Antworten)
4. **PortTypes**: Zur Verknüpfung von *Operations* und *Messages*
5. **Bindings**: Zur Verknüpfung einer *Operation* mit einem Transportprotkoll
6. **Types**: Zur Definition anwenderdefinierter Übergabe- und Rückgabetypen

Die Angaben zur Struktur der anwenderdefinierten Typen ist optional und muß nur im Falle der Existenz solcher Typen erfolgen.

Das Beispiel 114 zeigt die vollständige WSDL-Dienstbeschreibung des aus der Fallstudie bekannten Dienstes:

Beispiel 114: WSDL-Beschreibung des Beispieldienstes

```
(1)<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.
apache.org/xml-soap" xmlns:impl="http://10.0.0.1:8080/axis/services/Complex" xmlns:
intf="http://10.0.0.1:8080/axis/services/Complex" xmlns:soapenc="http://schemas.xmlsoap.
org/soap/encoding/" xmlns:tns1="urn:Complex" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/
XMLSchema" targetNamespace="http://10.0.0.1:8080/axis/services/Complex">
(2)     <wsdl:types>
(3)             <schema targetNamespace="urn:Complex" xmlns="http://www.w3.org/2001/
XMLSchema">
(4)                     <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
(5)                     <complexType name="Complex">
(6)                             <sequence>
(7)                                     <element name="im" type="xsd:double"/>
(8)                                     <element name="re" type="xsd:double"/>
(9)                             </sequence>
(10)                    </complexType>
(11)            </schema>
(12)    </wsdl:types>
(13)    <wsdl:message name="addResponse">
(14)            <wsdl:part name="addReturn" type="tns1:Complex"/>
(15)    </wsdl:message>
(16)    <wsdl:message name="divRequest">
(17)            <wsdl:part name="c1" type="tns1:Complex"/>
(18)            <wsdl:part name="c2" type="tns1:Complex"/>
(19)    </wsdl:message>
(20)    <wsdl:message name="modulusRequest">
(21)            <wsdl:part name="c1" type="tns1:Complex"/>
(22)    </wsdl:message>
(23)    <wsdl:message name="divResponse">
(24)            <wsdl:part name="divReturn" type="tns1:Complex"/>
(25)    </wsdl:message>
(26)    <wsdl:message name="multResponse">
(27)            <wsdl:part name="multReturn" type="tns1:Complex"/>
(28)    </wsdl:message>
(29)    <wsdl:message name="powResponse">
(30)            <wsdl:part name="powReturn" type="tns1:Complex"/>
(31)    </wsdl:message>
(32)    <wsdl:message name="addRequest">
(33)            <wsdl:part name="c1" type="tns1:Complex"/>
(34)            <wsdl:part name="c2" type="tns1:Complex"/>
(35)    </wsdl:message>
(36)    <wsdl:message name="modulusResponse">
(37)            <wsdl:part name="modulusReturn" type="xsd:double"/>
(38)    </wsdl:message>
(39)    <wsdl:message name="powRequest">
(40)            <wsdl:part name="c1" type="tns1:Complex"/>
(41)            <wsdl:part name="n" type="xsd:int"/>
(42)    </wsdl:message>
(43)    <wsdl:message name="multRequest">
(44)            <wsdl:part name="c1" type="tns1:Complex"/>
(45)            <wsdl:part name="c2" type="tns1:Complex"/>
(46)    </wsdl:message>
(47)    <wsdl:portType name="Complex">
(48)            <wsdl:operation name="pow" parameterOrder="c1 n">
(49)                    <wsdl:input name="powRequest" message="impl:powRequest"/>
(50)                    <wsdl:output name="powResponse" message="impl:powResponse"/>
(51)            </wsdl:operation>
(52)            <wsdl:operation name="add" parameterOrder="c1 c2">
(53)                    <wsdl:input name="addRequest" message="impl:addRequest"/>
(54)                    <wsdl:output name="addResponse" message="impl:addResponse"/>
(55)            </wsdl:operation>
(56)            <wsdl:operation name="mult" parameterOrder="c1 c2">
```

```
(57)                    <wsdl:input name="multRequest" message="impl:multRequest"/>
(58)                    <wsdl:output name="multResponse" message="impl:multResponse"/>
(59)            </wsdl:operation>
(60)            <wsdl:operation name="modulus" parameterOrder="c1">
(61)                    <wsdl:input name="modulusRequest" message="impl:modulusRequest"/>
(62)                    <wsdl:output name="modulusResponse" message="impl:modulusResponse"/
>
(63)            </wsdl:operation>
(64)            <wsdl:operation name="div" parameterOrder="c1 c2">
(65)                    <wsdl:input name="divRequest" message="impl:divRequest"/>
(66)                    <wsdl:output name="divResponse" message="impl:divResponse"/>
(67)            </wsdl:operation>
(68)    </wsdl:portType>
(69)    <wsdl:binding name="ComplexSoapBinding" type="impl:Complex">
(70)            <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
http"/>
(71)            <wsdl:operation name="pow">
(72)                    <wsdlsoap:operation/>
(73)                    <wsdl:input>
(74)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(75)                    </wsdl:input>
(76)                    <wsdl:output>
(77)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(78)                    </wsdl:output>
(79)            </wsdl:operation>
(80)            <wsdl:operation name="add">
(81)                    <wsdlsoap:operation/>
(82)                    <wsdl:input>
(83)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(84)                    </wsdl:input>
(85)                    <wsdl:output>
(86)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(87)                    </wsdl:output>
(88)            </wsdl:operation>
(89)            <wsdl:operation name="mult">
(90)                    <wsdlsoap:operation/>
(91)                    <wsdl:input>
(92)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(93)                    </wsdl:input>
(94)                    <wsdl:output>
(95)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(96)                    </wsdl:output>
(97)            </wsdl:operation>
(98)            <wsdl:operation name="modulus">
(99)                    <wsdlsoap:operation/>
(100)                   <wsdl:input>
(101)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(102)                   </wsdl:input>
(103)                   <wsdl:output>
(104)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(105)                   </wsdl:output>
(106)           </wsdl:operation>
(107)           <wsdl:operation name="div">
(108)                   <wsdlsoap:operation/>
(109)                   <wsdl:input>
(110)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(111)                   </wsdl:input>
(112)                   <wsdl:output>
(113)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(114)                   </wsdl:output>
(115)           </wsdl:operation>
(116)   </wsdl:binding>
(117)   <wsdl:service name="ComplexService">
(118)           <wsdl:port name="Complex" binding="impl:ComplexSoapBinding">
(119)                   <wsdlsoap:address location="http://10.0.0.1:8080/axis/services/
Complex"/>
```

```
(120)          </wsdl:port>
(121)   </wsdl:service>
(122)</wsdl:definitions>
```

Das Beispiel spiegelt im **Service**-Element die bereits durch den Deploymentdeskriptor getroffenen Festlegungen wieder und macht sie so dem (potentiellen) Dienstnutzer zugänglich. So enthält das Attribut `name` mit dem Wert `ComplexService` den gewählten Namen des Dienstes. Zusätzlich der als Kindelement realisierte `address`-Eintrag unter `location` die Adresse des SOAP-Endpunktes wieder welche SOAP-Botschaften zum Dienstaufruf entgegennimmt.
Da derselbe Dienst durch verschiedene Endpunkte zur Verfügung gestellt werden kann, ist das mehrfache Auftreten von `address`-Element zugelassen. Aus Gründen der Eindeutigkeit und Zuordnung des Endpunktes zum jeweils unterstützten Transportprotokoll ist jedes `address`-Element durch ein `port`-Element umschlossen welches auf das definierte Protokoll verweist.

Jedes **Operation**-Element definiert die zum Aufruf zu übermittelnden Parameter hinsichtlich Reihenfolge und referenziert die eine Interaktion konstituierenden Interaktionsschritte.
Im Beispiel legt die Operation `pow` (die Benennung wird durch das `name`-Attribut festgelegt) fest, daß die beiden Parameter `c1` und `n` benötigt werden. Zusätzlich erfolgt durch das Element `input` eine Referenz auf die Eingabenachricht bzw. `output` auf die Ausgabenachricht die zur Initiierung der Operation bzw. nach deren Ende versandt werden.

Innerhalb des **Message**-Elements werden die Übergabe Parameter inhaltlich hinsichtlich Typ und Name ausdefiniert.
Für die Nachricht `powRequest` (Aufruf der Methode `pow`) werden daher `c1` vom Typ eigendefinierten Typ `Complex` sowie `n` vom Standardtyp `int` genannt.

Durch das WSDL-Element `PortType` erfolgt die Aggregation der zuvor definierten Operationen, deren Nachrichten, an den in der Service-Definition genannten Endpunkt.

Die technischen Details wie gewähltes Encodierungsschema und genutzter Namensraum werden im Rahmen des **Binding**-Elements für jeden Nichtstandardtypen --- im Beispiel ist dies `Complex` --- definiert.

Die zur Erzeugung der über die Netzwerkleitung zu transportierenden XML-Darstellung dieser Nichtstandardtypen wird durch ein XML-Schemafragment beschrieben, welches als Kindelement des WSDL-Elements **Types** abgelegt ist.

UDDI

Zwar bietet WSDL eine wertvolle Möglichkeit zur Beschreibung der technischen Schnittstellencharakteristika eines Web Service, jedoch bleiben andere Fragestellungen --- etwa die nach der Natur und menschenlesbaren Beschreibung eines Dienstes oder seinen Nutzungs- und Abrechnungsbedingungen --- durch diesen Standard vollkommen offen. Überdies enthält das offizielle Spezifikationsdokument keinerlei Aussagen über einen präferierten oder auch nur sinnvollen Bereitstellungsort der WSDL-Beschreibungen.
Einen Ansatz zur Antwort auf diese Fragestellungen versucht der im Rahmen des OASIS-Standardisierungsprozeß vorangetriebene Verzeichnisdienst *Universal Service Description, Discovery and Integration* zu liefern.

Dieses, selbst als SOAP-basierter Web-Dienst angebotene, Verzeichnisdienst stellt eine Verwaltungsstruktur zur Ablage von WSDL-Beschreibungen und anderer dienstbezogener deskriptiver Metadaten bereit die durch eine definierte Schnittstelle abgefragt werden kann.

Die Grundprimitive des UDDI-Dienstes sind:

- **Business Entity**: Der Anbieter eines Web Service
- **Business Service**: Der angebotene Dienst
- **tModel**: Die Beschreibung des Dienstes
- **BindingTemplate**: Verknüpfung von tModel und Business Service

Ziel der **Business Entity**-Struktur ist es telephonbuchartig beschreibende Metadaten zum Dienstanbieter, wie Firmenname und Erreichbarkeitsdaten in einer alphabetisch sortierten Struktur anzubieten.

Jedem Business Entity-Eintrag können mehrere **Business Services** zugeordnet sein, welche die angebotenen Web Services repräsentieren.

Jedes **tModel** (Abkürzung für *technical model*) kann eine WSDL-Beschreibung oder beliebige durch den Anwender festlegbare dienstbezogene deskriptive Inhalte aufnehmen.
Insbesondere kann ein tModel auch Kategorisierungen eines Dienstes, etwa die Zuordnung zu einer Dienstfamilie, die Herstellung eines bestimmten Typ-Kontexts wie Erbringungsort des Dienstes, aufnehmen.

Alle tModels eines Dienstes werden mit diesem durch **BindingTemplate**-Elemente verbunden.

Die Interaktion mit einem UDDI-Verzeichnisdienst kann SOAP-basiert oder durch manuelle Erfassung der abzulegenden Daten über eine Webseite erfolgen.
Ziel dieser Interaktion ist zumeist einer der global angebotenen UDDI-Verzeichnisdienste, die gegenwärtig innerhalb eines Tages für die vollständige Inhaltsreplizierung sorgen.

Aufgrund immernoch offener Sicherheitsfragestellungen und der als ungelöst anzusehenden Abrechnungsproblematik liegen jedoch in den aktuell zugänglichen UDDI-Diensten kaum kommerzielle Dienstangebote vor, sondern überwiegend Testeinträge.

Web-Referenzen 13: Weiterführende Links

- SOAP-Standard des W3C
  - Primer
  - Messaging Framework
  - Adjuncts
- Spezifikation der Web Service Description Language
- UDDI-Spezifikation
- Axis-Framework Eine Open-Source SOAP-Implementierung
- Aktuelle Informationen rund um Web Services

---

5 Präsentationsaspekte

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.1 XHTML und XForms

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.2 Java Server Pages (JSP)

Beim Einsatz von Servlets ist oftmals das Verhältnis zwischen berechneten Anteilen und statisch produzierten Ausgaben deutlich zu Gunsten der statischen Anteile verschoben. Gleichzeitig ist jedoch die Ausgabe statischen HTML- oder XML-Codes vergleichsweise mühsam und schreibaufwendig, wie Beispiel 102 zeigt.
Zur Abhilfe dies vermeidbaren Aufwende und damit zur Unterstützung von Applikationstypen in denen die dynamisch festzulegenden Anteile einer Ausgabe deutlich gegenüber den statischen zurückstehen sieht Sammlung der J2EE-Applikationstypen die *Java Server Pages* (JSP) vor.

Konzeptionell stellt eine Server Page eine --- gegenüber klassischem HTML/XML nahezu unveränderte --- Ausgabeseite dar, in die Anweisungen eingestreut werden können, welche zur Auslieferungszeit an den Client durch ihr Berechnungsergebnis ersetzt werden.
Technisch ist der Ansatz auf Basis der existierenden Servlet-API realisiert und wird serverseitig auch transparent in ein solches übersetzt. Als einzige Einschränkung gegenüber der Mächtigkeit des Servletansatzes ist jedoch die Beschränkung auf die Verarbeitung von HTTP-POST-Anfragen zu sehen.

Beispiel 115 zeigt die Reformulierung der Servlet-basierten Umsetzung aus Beispiel 102 als JSP:

Beispiel 115: Einfache JSP

```
(1)<%@page import="java.util.Date"%>
(2)
(3)<html>
(4)     <head>
(5)             <title>Hello World!</title>
(6)     </head>
(7)     <body>
(8)             <h1>Hello World!</h1>
(9)             <p>Current date: <%= (new Date()).toString() %></p>
(10)    </body>
(11)</html>
```

Download des Beispiels

Augenfälligste Änderung gegenüber der Fassung als Servlet ist die direkte Darstellung der HTML-Ausgabeelemente. Die aktiven Inhalte werden durch Elemente, die mit dem Zeichen „%=“ beginnen und

mit „%“ enden eingeschlossen. Dazwischen befindet sich übersetzbarer Java-Quellcode.
Etwaige benötige Bibliotheken werden im Java-üblichen import-Stil angegeben und zu Beginn der JSP-Seite eingeführt.

Zu Beginn einer JSP-Seite können im Rahmen der page-Deklaration verschiedene seitenglobale Vereinbarungen getroffen werden:

- `language`
  Wurde ursprünglich für eine (immernoch ausstehende) Unterstützung des JSP-Mechanismus durch verschiedene Programmiersprachen eingeführt. Vorgabegemäß hat das Attribut den Wert „Java“. Andere Werte sind gegenwärtig nicht definiert.
- `extends`
  Erlaubt es die aus der JSP-Seite entstehende Servletklasse explizit als Spezialisierung der angegebenen Klasse zu definieren.
- `import`
  Kommaseparierte Liste der verwendeten Bibliotheken.
- `session`
  Legt fest, ob die JSP-Seite im Rahmen des HTTP-Sessionhandlings zustandsbehaftet verarbeitet wird.
- `buffer`
  Erlaubt die Festlegung der innerhalb des übersetzten Servelts verwendeten Ausgabemechanismus. Vorgabegemäß und bei Angabe von none ist dies `PrintWriter`.
- `autoFlush`
  Legt fest, daß gefüllte Ausgabepuffer automatisch an den anfragenden Client übermittelt werden.
- `isThreadSafe`
  Weist dieses Attribut den Wert `true` auf, dann werden durch die Ausführungsumgebungen mehrere Anfragen zur selben Zeit an die JSP (bzw. das entstehende Servlet) übergeben.
- `info`
  Möglichkeit zur Ablage informaler deskriptiver Information.
- `errorPage`
  Referenz auf eine andere JSP, die zur Verarbeitung von Ausnahmeereignissen herangezogen wird.
- `isErrorPage`
  Kennzeichent ob eine JSP-Seite zur Fehlerbehandlung einer anderen Seite herangezogen werden kann und soll.

Die nähere Analyse des Quellecodes von Beispiel 115 sowie die Semantik der vorgestellten Attribute offenbart zwei unterschiedliche Codierungsformen für JSPs. Zunächst die durch <@ eingeleiteten vordefinierten **Direktiven**, welche im Rahmen des Serveltübersetzungsprozesses in Java-Code transformiert werden. Diese Codeabbildung wird durch eine sog. *Tag Library*, einer Sammlung von Javacodefragementen und -transformationsvorschriften zur dynamischen Erzeugung der Ausgabe, gesteuert.
Zum zweiten gestattet der JSP-Mechanismus die direkte Darstellung von Java-Quellcode, der zur Laufzeit direkt in ausführbaren Code umgesetzt wird im Rahmen der sog. **Scriptlets**. Diese durch <% eingeführten Codeanteile werden unverädert in das entstehende Servlet einkopiert.
Zusätzlich stellt das Beispiel, durch <= eingeleitete, Ausdrücke (**Expressions**) dar, deren Resultatwert durch die Laufzeitumgebung in die an den Client übermittelte Seite eingefügt wird.
Im Beispiel nicht dargestellt sind Variablen- oder Konstantendeklarationen, welche auf die Einleitungssequenz <%! folgen.

Tag Libraries

Zur Strukturierung des Entwurfs und zur Wiederverwendung bereits einmal eingesetzter Codesequenzen für JSPs existiert mit dem Mechanismus der Tag Libraries die Möglichkeit anwenderdefiniert den Sprachumfang der Server Pages zu erweitern.
Einige häufig gebrauchte Aktionen sind bereits in der im Standardumfang enthaltenen Bibliothek versammelt:

- `useBean`
  Zugriff auf eine Enterprise Java Bean.
- `setProperty`
  Setzen einer Java-Property. Wird zumeist in Verbindung mit useBean verwendet um Bean-Eigenschaften festzulegen.
- `getProperty`
  Auslesen einer Java-Property.
- `param`
  Dient zur Parameterübergabe an andere Direktiven.
- `include`
  Inklusion anderer Datenquellen.
- `forward`
  Weiterleitung des Aufrufes der die JSP-Seite erreichte an andere Ressource.
- `plugin`
  Einschluß eines Java-Applets in die erzeugte Ausgabe.

Beispiel 116 zeigt mit der `include`-Direktive die vordefinierte JSP-Aktion zur serverseitigen Inklusion

existierender Dateien.

Beispiel 116: Importmechanismus für Server Pages

```
(1)<html>
(2)     <head>
(3)             <title>J. W. von Goethe: Faust I</title>
(4)     </head>
(5)     <body>
(6)             <h1>FAUST: EINE TRAGÖDIE</h1>
(7)             <h2>Zueignung.</h2>
(8)             <%@ include file="Zueignung.html" %>
(9)
(10)            <h2>Vorspiel auf dem Theater.</h2>
(11)            <%@ include file="Vorspiel.html" %>
(12)
(13)            <h2>Prolog im Himmel.</h2>
(14)            <%@ include file="Prolog.html" %>
(15)    </body>
(16)</html>
```

Download des Beispiels

Die Erzeugung eigener Tag Libraries geschieht durch Erstellung einer Java-Klasse, welche die Schnittstelle `javax.servlet.jsp.tagext.Tag` implementiert. Zumeist wird jedoch die Klasse `TagSupport` spezialisiert, die einige zusätzliche Methoden definiert, welche die Erstellung vereinfachen.

Beispiel 117 zeigt die Umsetzung des anwenderdefinierten Tags `hello` und Beispiel 118 seine Anwendung.

Beispiel 117: Erstellung einer Tag Library

```
(1)import java.io.IOException;
(2)import java.util.Date;
(3)
(4)import javax.servlet.jsp.JspTagException;
(5)import javax.servlet.jsp.tagext.TagSupport;
(6)
(7)public class HelloTag extends TagSupport {
(8)     public int doStartTag() throws JspTagException {
(9)             return EVAL_BODY_INCLUDE;
(10)    }
(11)
(12)    public int doEndTag() throws JspTagException {
(13)            try {
(14)                    pageContext.getOut().write("Hello World!");
(15)                    pageContext.getOut().write(
(16)                            "current date: " + new Date().toString());
(17)            } catch (IOException ioe) {
(18)                    ioe.printStackTrace();
(19)            }
(20)            return EVAL_PAGE;
(21)    }
(22)
(23)}
```

Download des Beispiels

Das Beispiel definiert Methoden für die beiden durch die Schnittstelle vorgegebenen Operationen `doStartTag` und `doEndTag`, die bei der Verarbeitung der entsprechenden Tag-Anteile aufgerufen werden. Der Rückgabewert der Methoden legt fest, daß die Inhalte des anwenderdefinierten Elements in der JSP-Seite in den Ausgabestrom übernommen werden (`EVAL_BODY_INCLUDE`) bzw. das mit der Verarbeitung der Seite fortgefahren werden soll (`EVAL_PAGE`).

Beispiel 118: Nutzung eines anwenderdefinierten Tags

```
(1)<%@ taglib uri="hello" prefix="example" %>
(2)<html>
(3)     <head>
(4)             <title>Hello World!</title>
(5)     </head>
(6)     <body>
(7)             <p>Statische Ausgabe ...</p>
(8)             <example:hello>
(9)             </example:hello>
(10)    </body>
(11)</html>
```

Download des Beispiels

Abschließende Bemerkung:
Der JSP-Mechanismus eignet sich sehr gut zur schnellen Erstellung leistungsfähiger Anwendungen einer gewissen Komplexität jedoch wirkt sich die physische Kopplung des Kontrollflusses an die erzeugte Ausgabe durch die Ablage in derselben Datei negativ auf die Skalierbarkeit hinsichtlich der betrachteten Problemgröße aus. Durch die mangelnde zentralisierte Verwaltbarkeit lassen sich komplexe Anwendungen oft nur noch schwer überschauen und testen.
Zusätzlich wirkt sich die erst zum Aufrufzeitpunkt serverseitig vorgenommene Übersetzung in ein Servlet nicht nur negativ auf die Laufzeit aus, sondern erschwert zusätzlich die Fehlersuche, da Syntaxfehler erst beim Aufruf entdeckt werden können.

5.3 Java Server Faces (JSF)

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.4 XSL Transformations (XSLT)

Die prominenteste Verwendung der XPath-Pfadausdrücke und eine der in jüngerer Zeit am weitesten beachteten XML-Vokabulare dürfte XSLT -- die Sprache der *XSL Transformations* -- sein. Sie wurde im Verlauf der Standardisierung der Stylesheetsprache XSL von dieser abgetrennt und seither durch eine eigene W3C-Arbeitsgruppe vorangetrieben.
In einem Satz zusammengefaßt stellt sie eine Turing-vollständige funktionale Programmiersprache zur Transformation wohlgeformter XML-Dokumente in beliebige Unicode-Streams -- und damit im speziellen wiederum in XML-Dokumente -- dar.
Der Begriff *Transformationen* bezeichnet hierbei die Selektion einzelner Bestandteile des Quelldokuments, deren Umordnung sowie die Ableitung neuer Inhalte aus den bereits bestehenden.
Derzeit aktuell ist die Recommendation zur Version 1.0 von Herbst 1999. Intern arbeitet das W3C bereits an der Nachfolgerversion XSLT v2.0, welche auch die absehbaren Entwicklungen des Umfeldes, wie XPath v2.0 oder XML-Schema, berücksichtigen wird.

Jedes XSLT-Stylesheet ist ein gültiges XML-Dokument, in dem alle Elemente der Sprache XSLT im Namensraum `http://www.w3.org/1999/XSL/Transform` plaziert sind.
Üblicherweise wird der Namensraum an das Präfix `xsl` gebunden, welches allen XSLT-Sprachelementen explizit vorangestellt wird.
Das Wuzelelement eines XSLT-Dokuments bildet der Knoten `stylesheet`. Alternativ kann auch `transform` angegeben werden. Zusätzlich verfügt jedes Stylesheet über ein Versionsattribut zur Bezeichnung der verwendeten XSLT-Version.
Das Beispiel zeigt ein minimales Stylesheet

Beispiel 119: Ein minimales Stylesheet

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

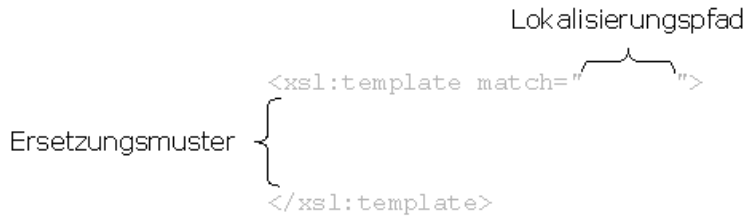
Download des Beispiels

Angewendet auf das Beispieldokument der Projektverwaltung liefert es folgende Ausgabe:

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)Hans Hinterhuber
(3)Franz Xaver Obermüller
(4)IT-Kompetenz verschiedene Betriebssysteme und professionelle Programmierung
verschiedener Programmiersprachen
(5)Entwickler von 1988-1990 Projektleiterfunktion von 1990-93 im X42-Projekt in Abteilung
AB&amp;amp;C
(6)Fritz Meier
```

Dies mag zunächst verwundern, definiert doch das Beispiel-Stylesheet keinerlei Transformationsregeln oder Ausgabefunktionen.
Das Ergebnis des minimal-Beispiels wird durch die eingebauten Vorgaberegeln erzeugt. Diese geben, sofern nicht anders angegeben alle Character Information Items unverändert aus.
Durch Überschreiben dieser Vorgaben und die Definition eigener Regeln lassen sich komplexe Transformationen auf dem Eingabedokument verwirklichen.

Transformationsschablonen



Die Graphik zeigt den Aufbau einer Transformationsschablone. Ihre beiden Hauptbestandteile sind der *Lokalisierungspfad* und das *Ersetzungsmuster*.
Der Lokalisierungspfad (in der Spezifikation als *pattern* bezeichnet) liefert eine Knotenmenge. Als Syntax wird eine eingeschränkte Variante der Lokatorsprache XPath verwendet.
Augenfälligster Unterschied zur bisherigen Notation ist die Optionalität der descendant-Achse (zumeist zu // verkürzt) zur hierarchiebenenunabhängigen Lokalisierung eines Knotens.
Im Rumpf des Musters legt das Ersetzungsmuster diejenige Zeichenfolge fest, die statt jedem Element der lokalisierten Knotenmenge ausgegeben werden soll.
Das nachfolgende Transformationssheet liefert angewandt auf das Projektverwaltungsbeispiel dreimal die Ausgabe `Person gefunden!`; für jedes Auftreten des Knotens `Person` in der Eingabe.

Beispiel 120: Eine einfache Transformation

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)     <xsl:template match="Person">
(6)             <xsl:text>Person gefunden!</xsl:text>
(7)     </xsl:template>
(8)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Innerhalb des Ersetzungsmusters können im Allgemeinen beliebige Textsequenzen angegeben werden. Insbesondere ist die Verwendung wohlgeformter XML-Fragmente zugelassen.
Textsequenzen werden hierbei durch direktes Anschreiben, oder umschlossen durch das Element `xsl:text` definiert.
Anmerkung: Die Anforderungen an die Wohlgeformtheit sind hierbei auf die korrekte Terminierung der Elemente (damit einhergehend ihre korrekte Schachtelung), sowie die Quotierung der Attribute beschränkt.
Im folgenden Beispiel wird jedes Element des Typs `Person` durch ein leeres `Mitarbeiter`-Element ersetzt.
Das Beispiel erklärt auch die übliche Anwendungspraxis, alle XSLT-Elemente durch Namensraumpräfix zu qualifizieren, statt der -- weniger schreibaufwendigen -- Überschreibung des Vorgabenamensraumes.
Würde der Vorgabenamensraum mit der Namensraum-URI der XSL-Transformations belegt, so befände sich auch jedes XML-Element und -Attribut innerhalb des Ersetzungsmusters in diesem Namensraum. Als Konsequenz würde der XSLT-Prozessor die Transformation wegen des Auftretens ungültiger (d.h. nicht in der XSLT-Sprache enthaltener) Elemente ablehnen. Bei Redefinition des Vorgabenamensraumes müßte daher für alle Elemente, die nicht Bestandteil von XSLT sind, eine explizite Namensraumdefinition im Element erfolgen, wodurch die Lesbarkeit stark herabgesetzt würde.

Beispiel 121: Erzeugung einer XML-Ausgabe

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter/>
(7)</xsl:template>
(8)
(9)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Erwartungsgemäß liefert das Beispiel dreifach das leere Element `Mitarbeiter` anstelle der `Personen`-Elemente der Eingabe.
Die bisher genutzte Form der Ersetzung ist jedoch für die praktische Anwendung in nur äußerst wenigen Fällen geeignet, da sie keine Übernahme von Daten der Eingabedatei in die Ausgabe erlaubt.
Dieses Manko wird durch das XSLT-Sprachelement `value-of` behoben.

Das Element `value-of` übernimmt als Teil des Ersetzungsmusters frei selektierbare Text-artige Informationsknoten aus dem Quelldokument. Die Lokalisierung der zu übernehmenden Knoten erfolgt

durch XPath-Syntax innerhalb des select-Attributs.
Im folgenden Beispiel wird der Inhalt der Personen-Knoten in den neuen Knotentyp Mitarbeiter übernommen.

Beispiel 122: Übernahme bestehender Information aus dem Quelldokument

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:value-of select="self::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Das Resultat liefert jedoch nicht das Quelldokument, unter Umbenennung der Personen-Knoten in Mitarbeiter, sondern die dargestellte Textsequenz.

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)<Mitarbeiter>Hans Hinterhuber</Mitarbeiter>
(3)<Mitarbeiter>Franz Xaver Obermüller IT-Kompetenzverschiedene Betriebssysteme und
professionelle
(4)Programmierung verschiedener Programmiersprachen Entwickler von 1988-1990
Projektleiterfunktion
(5)von 1990-93 im X42-Projekt in Abteilung AB&amp;amp;C</Mitarbeiter>
(6)<Mitarbeiter>Fritz Meier</Mitarbeiter>
```

Die Lösung liegt in der Definition des value-of-Elements. Es konvertiert alle durch den im select-Attribut bezeichneten XPath lokalisierten Knoten in ihre Textrepräsentation. Im vorliegenden Beispiel ist dies der Inhalt der Knoten des Typs Person, der durch den Elementinhalt gebildet wird. Der Elementinhalt wird hierbei durch alle Kindknoten und deren Attribute gebildet.

Zur unveränderten Übernahme eines vollständigen Elements einschließlich der Auszeichnungssymbole wird das XSLT-Element copy-of angeboten.
Das nachfolgende Beispiel modifiziert das vorhergend diskutierte Stylesheet dergestalt, daß für alle Personen-Knoten zunächst ein öffnender Mitarbeiter-Tag gesetzt wird. Im Rumpf des so begonnenen Elements werden durch das copy-of-Element alle Kindknoten der aktuellen Knotenmenge unverändert kopiert.
Als zusätzliche Veränderung gegenüber dem vorigen Beispiel fällt die Nutzung der child-Achse im select-Attribut des copy-of-Elements auf. Dies ist notwendig, da durch den Lokalisierungspfad der Schablone alle Personen-Knoten zu einer Ergebnisknotenmenge zusammengefaßt werden. Die Anwendung der self-Achse innerhalb des copy-of-Elements würde daher auch die Personen-Knoten selbst übernehmen.
Zusammenfassend läßt sich die Funktionalität des Beispiels mit *Umbenennung aller Elemente des Types Person in Mitarbeiter* wiedergeben.

Beispiel 123: Kopieren vollständiger Elementknoten

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="//Person">
(6)     <Mitarbeiter>
(7)             <xsl:copy-of select="child::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Die vorgestellte Transformationsvorschrift ist hinreichend flexibel für Knoten des Typs Person auf beliebiger Hierarchiestufe des Eingabedokuments. Jedoch versagt sie bereits beim Versuch einen weiteren Transformationsschritt einzubauen, der auf einem der unverändert kopierten Kindelemente von Person operiert.
Das folgende Stylesheet stellt einen flexibleren Ansatz zur Umbenennung von einzelnen Knoten vor.

Gegenüber der vorhergehenden Fassung ist der Umfang um zwei weitere `template`-Elemente erweitert. Eines, das für alle `Qualifikationsprofil`-Knoten angewendet wird und eines für alle anderen Knoten. Der dort eingesetzte Lokalisierungspfad liefert als Ergebnismenge die Vereinigung aller Attributknoten (`attribute::*`) mit allen Knoten außer dem Wurzelknoten (`node()`-Funktion). Im Rumpf des Elements wird das copy-Element zur Kopie jedes Elements verwendet. Anders als das bisher herangezogene copy-of-Element übernimmt diese Variante ausschließlich das aktuelle Element in das Ergebnisdokument und läßt eventuell vorhandene Kindelemente unberücksichtigt.
Das `template`-Element mit dem Lokalisierungspfad `Qualifikationsprofil` verfügt über kein Ersetzungsmuster. Es bewirkt damit die Unterdrückung des Teilbaumes unterhalb aller Knoten des Typs `Qualifikationsprofil`.

Die korrekte Anwendung der verschiedenen Schablonen wird durch den ausführenden XSLT-Prozessor gewährleistet, er ermittelt anhand der Lokalisierungspfade das best-passendste Template und bringt es zur Ausführung. Wenn, wie im vorliegenden Beispiel, mehrere Pfadausdrücke einen Knoten des Eingabedokuments lokalisieren, so wird das am weitesten spezifizierte Muster ausgewählt. Im untenstehenden Beispiel gilt dies für alle Elemente des Typs `Person`. Jeder dieser Knoten ist sowohl durch den XPath-Ausdruck der ersten Schablone als auch durch den allgemeineren Ausdruck `node()` zugänglich. Der explizite Pfad `Person` des ersten Lokalisierungsmusters ist jedoch gegenüber der Ergebnismenge der node-Funktion (deutlich) spezifischer.

Da jeder Knoten, außer denen des Typs `Person` und `Qualifikationsprofil`, unverändert in den Ausgabestrom übernommen werden soll, wird im Beispiel wieder ein copy-Element eingesetzt. Im Unterschied zum bisher verwendeten `copy-of` jedoch mit der Einschränkung, daß copy nur den aktuellen Knoten kopiert und eventuell existierende Kindknoten unberücksichtigt läßt. Dieser Vorgang wird auch als *shallow copy* bezeichnet.

Steuerung der Transformationsreihenfolge durch `apply-templates`-Elemente

Standardmäßig durchläuft ein XSLT-Prozessor den aus dem Eingabedokument erzeugten Baum ausgehend vom Wurzelknoten in Preorder Reihenfolge. Während des Traversierungsvorganges werden die zum jeweiligen Knoten „passenden“ Schablonen ausgewertet. „Passend“ deutet hierbei auf das Enthaltensein des besuchten Knotens in der Ergebnismenge eines XPath-Ausdrucks innerhalb eines `match`-Attributes hin. Jedoch ist auch die anwenderdefinierte Beeinflussung der vorgegebenen Abarbeitungsreihenfolge möglich. Hierzu enthält das Beispiel zwei `apply-templates`-Elemente. Diese lösen einen Rekursionsschritt aus, dergestalt, daß an jeder Stelle, an der sich ein `apply-templates`-Aufruf findet, der Prozessor versucht, weitere passende Schablonen anhand der angegebenen Lokalisierungspfade zu ermitteln. Diese können an der gegebenen Stelle ausgewertet werden. Der Vorgang entspricht damit der Substitution in funktionalen Programmiersprachen.
Im vorliegenden Fall wird nach Ausgabe des öffnenden Tags `Mitarbeiter` -- nachdem ein `Personen`-Knoten im Eingabedokument ermittelt wurde -- nach weiteren Knoten im Eingabedokument gesucht, zu denen Lokalisierungspfade in der XSLT-Transformationsvorschrift existieren. Dies ist für alle Attribute und Kindknoten von `Person` der Fall, da sie durch den Lokalisierungspfad `attribute::*|node()` zugänglich sind. So wird innerhalb des neu erzeugten Elements `Mitarbeiter` des Ausgabestroms das Ersetzungsmuster ausgeführt, das die Elemente und Attribute mit ihren Inhalten unverändert übernimmt. Als Besonderheit nutzt das `apply-templates`-Element im „allgemeinen“ (dritten) `template` das Attribut `select`. Es erlaubt die anwenderdefinierte Steuerung der Menge, innerhalb der nach weiteren *passenden* Lokalisierungspfaden gesucht werden soll. Standardmäßig ist diese Menge mit allen dem aktuellen Element nachfolgenden (`following`-Achse) Elementknoten gefüllt. Im vorliegenden Beispiel wird sie durch den XPath des Attributwertes um alle Attributknoten erweitert.

Beispiel 124: Flexible Umbenennung und Löschung von Elementen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:apply-templates/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)<xsl:template match="Qualifikationsprofil"/>
(12)
(13)<xsl:template match="attribute::*|node()">
(14)    <xsl:copy>
(15)            <xsl:apply-templates select="attribute::*|node()"/>
(16)    </xsl:copy>
(17)</xsl:template>
(18)
(19)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Übung 3: Verfolgung der Aufrufreihenfolge
Lassen Sie sich mit von einem XSLT-Prozessor die Aufrufreihenfolge der einzelnen Templates ausgeben.
Beispielsweise mit dem Prozessor Xalan aus dem Apache-Projekt.

Benannte Ersetzungsschablonen und Parameterübergabe

Neben den Lokationspfad-gesteuerten Schablonen kann der Anwender auch benannte Schablonen, ohne `match`-Attribut, definieren. Diese werden während der Abarbeitung des Eingabebaumes nicht berücksichtigt, sondern müssen manuell durch `call-template` aufgerufen werden.
Konzeptionell entsprechen sie Funktionsaufrufen herkömmlicher Programmiersprachen.
(In Spezifikation nachschlagen).
Die Definition erfolgt analog den bisher genutzten Schablonen, mit der Ausnahme, daß statt dem `match`-Attribut ein eindeutiger Name (Attribute `name`) angegeben wird.

Als neuer Freiheitsgrad beim nun notwendigen manuellen Aufruf tritt die Möglichkeit der Parameterübergabe hinzu. Als Parameter können beliebige Dokumentbestandteile als Knotenmenge, Ergebnisse von Funktionsausdrücken oder Konstanten übergeben werden.
Eine Parameterrückgabe ist nicht möglich, sie wird durch den Anteil der Schablone an der Ausgabe realisiert.
Die Aufrufsyntax lautet:

```
(1)<xsl:call-template name="QName">
(2)     <!-- Content: xsl:with-param* -->
(3)</xsl:call-template>
(4)<xsl:with-param name="QName" select="eingeschränkter XPath-Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:with-param>
```

Anmerkung: Wie in allen funktionalen Sprachen kommt den Funktionen dieses Typs besondere Bedeutung, als Ausgangspunkt rekursiver Aufrufe, zu.

Die Vorgabe-Transformationsregeln

Das einführende Beispiel dieses Kapitels griff bereits auf die in den XSLT-Prozessor „eingebauten“ Standard-Transformationsvorschriften (*built-in templates*) zurück.
So lautet die Definition, welche für alle Text- und Attributknoten der Eingabe den Inhalt in die Ausgabe kopiert:

```
(1)<xsl:template match="text()|@*">
(2)     <xsl:value-of select="."/>
(3)</xsl:template>
```

Ferner existiert ein `template` zur rekursiven Abarbeitung des Eingabebaumes, welches immer dann Anwendung findet, wenn sich keine spezialisiertere Transformationsvorschrift findet.

```
(1)<xsl:template match="*|/">
(2)     <xsl:apply-templates/>
(3)</xsl:template>
```

Ergänzt wird diese Zusammenstellung durch eine Schablone zur Eliminierung von Namensraum- und Kommentarknoten.

Elemente der Ablaufsteuerung

Ähnlich zu herkömmlichen Programmiersprachen bietet auch XSLT Sprachmittel zur Selektion und bedingten Verarbeitung, abhängig von den Eingabedaten.
So erlaubt das `if`-Element die Bearbeitung der umschlossenen Elemente nur, wenn die im `test`-Attribut formulierte Boole'sche Bedingung wahr ist.
(In Spezifikation nachschlagen)
Die Syntax lautet:

```
<xsl:if test="Boole'scher Ausdruck">   <!-- Content: template --> </xsl:if>
```

Das Beispiel zeigt die Nutzung des `if`-Elements. Verfügt ein Knoten des Typs `Person` über mehr als einen Kindknoten des Typs `Vorname` so wird der Inhalt des `if`-Elements abgearbeitet, der durch das XSLT-Element `message` während des Transformationsvorganges eine Bildschirmausgabe erzeugt.
Diese gibt zunächst den Inhalt des `Nachnamen` Knotens gefolgt von einem statischen Text aus.

Angewandt auf das Beispieldokument liefert es auf Kommandozeile die Ausgabe: `Obermüller hat mehr als einen Vornamen!`. Der Inhalt des Eingabedokuments wird unverändert kopiert.

Beispiel 125: Bedingte Verarbeitung durch Verwendung des if-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:if test="count(Vorname)&gt;1">
(8)                     <xsl:message><xsl:value-of select="Nachname"/> hat mehr als einen
Vornamen!</xsl:message>
(9)             </xsl:if>
(10)            <xsl:apply-templates/>
(11)    </Mitarbeiter>
(12)</xsl:template>
(13)
(14)<xsl:template match="Qualifikationsprofil"/>
(15)
(16)<xsl:template match="attribute::*|node()">
(17)    <xsl:copy>
(18)            <xsl:apply-templates select="attribute::*|node()"/>
(19)    </xsl:copy>
(20)</xsl:template>
(21)
(22)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

In Erweiterung der simplen Selektion bildet choose die Möglichkeiten einer Mehrfachselektion, oder einer simplen if-then-else-Struktur ab.
(In Spezifikation nachschlagen)
Die Syntax lautet:

```
(1)<xsl:choose>
(2)     <!-- Content: (xsl:when+, xsl:otherwise?) -->
(3)</xsl:choose>
(4)<xsl:when test="Boole'scher Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:when>
(7)<xsl:otherwise>
(8)     <!-- Content: template -->
(9)</xsl:otherwise>
```

Das Beispiel zeigt die Transformation des Beispieldokuments in eine XHTML-Ausgabe.
Hierbei wird zunächst der XHTML-Dokumentrahmen bei Auftreten des Dokumentknotens (Suchmuster `/`) erzeugt. Nach dem öffnenden XHTML-Rumpfelement body werden innerhalb des Quelldokuments wahlfrei weitere, auf eines der angegebenen Suchmuster passende, Knoten gesucht (`apply-templates`-Aufruf).
Bei Auftreten des Knotens `ProjektVerwaltung` wird -- nach einigen Überschriftszeilen -- der Kopf einer XHTML-Tabelle, bestehend aus den Elementen `table` und der Kopfzeile (eingeschlossen durch `tr`), erzeugt. Den Rumpf der Tabelle schreibt ein anderes Template. Dieses wird jedoch nicht direkt aufgerufen, sondern der Prozeß der Ermittlung neuer „passender“ Knoten neu initiiert; jedoch diesmal, durch das `select`-Attribut des `apply-templates`-Elements, nur auf Knoten des Typs `Person` beschränkt.
Die Ersetzungsregel für `Person` speichert zunächst den Inhalt des Attributs `PersID` in einer Variable. Anschließend werden die Tabellenelemente der in der Ersetzungsregel für `ProjektVerwaltung` geöffneten Tabelle geschrieben. Hierzu werden nacheinander die Ersetzungsmuster für Knoten des Typs `Vorname` bzw. `Nachname`, die Kindknoten des aktuellen Knotens sind (die `PersID` des aktuellen `Personen`-Knotens findet sich in der zuvor belegten Variable), aktiviert.
Ein zweites Tabellenelement enthält einen XHTML-Hyperlink zu den durch den Mitarbeiter bearbeiteten Projekten. Als Sprungziel wird hierbei der Inhalt des Attributs `mitarbeitInProjekt` eingetragen.
Das Ersetzungsmuster `Vorname` übernimmt durch `value-of` die in einen Zeichenkettenwert gewandelten Inhalte des Elements. Zusätzlich existiert ein zweites Ersetzungsmuster für Knoten des Typs `Vorname`, das jedoch durch ein Prädikat nur auf das zweite und alle folgenden Elemente dieses Typs angewendet wird. Es gibt vor der Übernahme des Elementinhaltes ein Leerzeichen aus.
Das Element `Nachname` bettet den Elementinhalt in eine benannte Sprungreferenz ein. Zur Erzeugung der dokumentweiten eindeutigen Benennung wird die Funktion `generate-id` herangezogen, die für jedes Element des Information Sets des Eingabedokuments einen eindeutigen Bezeichner liefert.
Nach Abschluß der Tabellengenerierung im durch den Lokalisierungspfad `ProjektVerwaltung` gekennzeichneten Template werden durch den Aufruf des benannten Ersetzungsmusters `wasteSpace` 25 Leerzeilen, jeweils gefüllt mit der Zeichenkette „...“, erzeugt. Als Besonderheit ist in diesem Muster sehr deutlich die Nutzung der Rekursion zur Modellierung einer Schleife zu sehen.

Zum Abschluß wird nochmals eine Tabelle, nun mit den Mitarbeitern und ihren Projekten, erzeugt.

Beispiel 126: Erzeugung eines XHTML-Reports

```
(1)<?xml version="1.0"?>
(2)<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(3)
(4)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes"
(5)             xmlns="http://www.w3.org/1999/xhtml" />
(6)
(7)<xsl:template match="/">
(8)     <html>
(9)             <head>
(10)                    <title>XML-Vorlesung Sommersemester 2001 -- Projektverwaltung</
title>
(11)            </head>
(12)            <body>
(13)                    <xsl:apply-templates/>
(14)            </body>
(15)    </html>
(16)</xsl:template>
(17)
(18)<xsl:template match="ProjektVerwaltung">
(19)    <center><h1>Projektverwaltung</h1></center>
(20)
(21)    <h2>Projekte</h2>
(22)    <table border="1">
(23)            <tr>
(24)                    <td><b>Projektnummer</b></td>
(25)                    <td><b>Projektleiter</b></td>
(26)            </tr>
(27)            <xsl:apply-templates select="Projekt"/>
(28)    </table>
(29)
(30)    <pre>
(31)            <xsl:call-template name="wasteSpace">
(32)                    <xsl:with-param name="maxLines">25</xsl:with-param>
(33)                    <xsl:with-param name="curLines">0</xsl:with-param>
(34)            </xsl:call-template>
(35)    </pre>
(36)
(37)    <h2>Mitarbeiter</h2>
(38)    <table border="1">
(39)            <tr>
(40)                    <td><b>Name</b></td>
(41)                    <td><b>Projekt</b></td>
(42)            </tr>
(43)            <xsl:apply-templates select="Person"/>
(44)    </table>
(45)</xsl:template>
(46)
(47)    <xsl:template name="wasteSpace">
(48)            <xsl:param name="maxLines"/>
(49)            <xsl:param name="curLines"/>
(50)            <xsl:if test="number($curLines) &lt; number($maxLines)">
(51)<xsl:text>...
(52)</xsl:text>
(53)                    <xsl:call-template name="wasteSpace">
(54)                            <xsl:with-param name="maxLines"><xsl:value-of
select="$maxLines"/></xsl:with-param>
(55)                            <xsl:with-param name="curLines"><xsl:value-of
select="$curLines + 1"/></xsl:with-param>
(56)                    </xsl:call-template>
(57)            </xsl:if>
(58)    </xsl:template>
(59)
(60)<xsl:template match="Projekt">
(61)    <xsl:variable name="prjLeiter" select="@Projektleiter"/>
(62)    <tr>
(63)            <td>
(64)                    <xsl:element name="a">
(65)                            <xsl:attribute name="name"><xsl:value-of select="@ID"/></
xsl:attribute>
(66)                            <xsl:value-of select="@ID"/>
(67)                    </xsl:element>
(68)            </td>
(69)            <td>
```

```
(70)                    <xsl:element name="a">
(71)                            <xsl:attribute name="href">#<xsl:value-of select="generate-
id(//Person[@PersID=$prjLeiter]/Nachname)"/></xsl:attribute>
(72)                            <xsl:value-of select="//Person[@PersID=$prjLeiter]/
Nachname"/>
(73)                    </xsl:element>
(74)            </td>
(75)    </tr>
(76)</xsl:template>
(77)
(78)<xsl:template match="Person">
(79)    <xsl:variable name="persNr" select="@PersID"/>
(80)    <tr>
(81)            <td><xsl:apply-templates select="Vorname[parent::Person/@PersID=$persNr]"/
> 
(82)                     <xsl:apply-templates select="Nachname[parent::Person/@PersID=
$persNr]"/></td>
(83)            <td>
(84)                    <xsl:element name="a">
(85)                            <xsl:attribute name="href">#<xsl:value-of
select="@mitarbeitInProjekt"/></xsl:attribute>
(86)                            <xsl:value-of select="@mitarbeitInProjekt"/>
(87)                    </xsl:element>
(88)            </td>
(89)    </tr>
(90)</xsl:template>
(91)
(92)<xsl:template match="Vorname">
(93)    <xsl:value-of select="."/>
(94)</xsl:template>
(95)
(96)<xsl:template match="Vorname[position() &gt; 1]">
(97)    <xsl:text> </xsl:text><xsl:value-of select="."/>
(98)</xsl:template>
(99)
(100)<xsl:template match="Nachname">
(101)   <xsl:element name="a">
(102)           <xsl:attribute name="name"><xsl:value-of select="generate-id()"/></xsl:
attribute>
(103)           <xsl:value-of select="."/>
(104)   </xsl:element>
(105)</xsl:template>
(106)
(107)<xsl:template match="text()"/>
(108)
(109)</xsl:stylesheet>
```

Download des Beispiels
Download der Ergebnisdatei

Die nachfolgende XSLT-Transformation ermittelt zu jedem beliebigen Element- und Attributknoten (Muster: * bzw. @*) die zugehörige Namensraum-URI (Funktion namespace-uri) und gibt sie gemeinsam mit dem Namen des Knotens (Funktion name) in einer XML-formatierten Ausgabe aus.
Zur Neuselektion aller weiteren Element- und Attributknoten wird apply-templates mit dem Musterausdruck @*|node() ausgeführt, um in die Prüfung nach weiteren passenden Knoten (node()) auch explizit alle beliebigen Attribute (@*) einzubeziehen. Standardmäßig ermittelt apply-templates weitere passende Knoten nur innerhalb der Elemente eines Dokuments.

Beispiel 127: Ausgabe Namensräume jedes Elements und Attributs eines beliebigen XML-Dokuments

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes" /
>
(6)
(7)<xsl:template match="*">
(8)     <xsl:element name="element">
(9)             <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(10)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></
xsl:attribute>
(11)       <xsl:apply-templates select="@*|node()"/>
(12)    </xsl:element>
(13)</xsl:template>
```

```
(14)
(15)<xsl:template match="@*">
(16)    <xsl:element name="attribute">
(17)            <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(18)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></
xsl:attribute>
(19)       <xsl:apply-templates select="@*|node()"/>
(20)    </xsl:element>
(21)</xsl:template>
(22)
(23)<xsl:template match="text()"/>
(24)
(25)</xsl:transform>
```

Download des Beispiels

Web-Referenzen 14: Weiterführende Links

- XSLT v1.0 Spezifikation
- XSLT @ jeckle.de
- Holman, K. G.: *What is XSLT?*
- Saxon -- ein freier XSLT-Prozessor
- Apache Xalan -- ein Open Source XSLT-Prozessor
- Literatur zum Thema

Übung 4: Textuelle Aufbereitung der Baumstruktur eines XML-Dokuments

Schreiben Sie eine XSLT-Transformation, die es erlaubt die Elementstruktur beliebiger wohlgeformter XML-Dokumente in Form eines „ASCII-Baums“ (siehe Beispiel) am Bildschirm auszugeben.
Beispiel:
Eingabe:

```
(1)<root>
(2)     <childX>
(3)             <childY1/>
(4)             <childY2/>
(5)     </childX>
(6)     <childX2/>
(7)</root>
```

Ausgabe:

```
(1)root
(2)+--childX
(3)     +--childY1
(4)     +--childY2
(5)+--childX2
```

---

6 Sicherheitsaspekte

6.1 Schlüsselaustausch

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.2 Leitungssicherheit

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.3 Digitale Signatur

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.4 Verschlüsselung

TODO(Hinweis: Dieses Kapitel fehlt noch)

Definitionsverzeichnis

Schlagwortverzeichnis

Abbildungsverzeichnis

Verzeichnis der Beispiele

Service provided by Mario Jeckle
Generated: 2004-06-08T12:51:08+01:00

Feedback SiteMap
This page's original location: http://www.jeckle.de/vorlesung/eBusinessEng/script.html
RDF description for this page