BizWeb

Kreuzfahrtbuchungen mit Web Services

Projektdokumentation

Hochschule Bremerhaven
Studiengang Informatik / Wirtschaftsinformatik
Sommersemester 2004, Wintersemester 2004/2005

Gesamtleitung:

Prof. Dr. Dieter Viefhues, *Dipl.-Ing. Alfred Schmidt*

Studentischer Projektleiter:

Bastian Onken

Projektmitglieder:

Benny Bräuer *Dominik Buhlmann*
Christian Eggers *Remo Fulle*
Marco Junge *Christian Lefke*
Viktor Schlegel *Christopher Schrade*
Nicole Siedek *Jan Stelmaszek*
Thomas Wäsch

Dokumentation:

Bastian Onken, *Benny Bräuer*

Träger:

Hochschule Bremerhaven
An der Karlstadt 8
27568 Bremerhaven

Kooperation:

AWI
Stiftung Alfred-Wegener-Institut
für Polar- und Meeresforschung
Columbusstraße
27568 Bremerhaven

Dokumentation erstellt und gesetzt in LaTeX 2ε

Inhaltsverzeichnis

V. Anhang 462

Teil I.

Einleitendes zum BizWeb–Projekt

1. Vorwort

Das Ziel dieses Lehr– und Lernprojekts bestand darin, dass sich die Mitglieder eines studentischen Jahresprojekts in das technische Konzept der Web Services einarbeiten und den Einsatz dieser Technologie anhand eines praktischen Anwendungsbeispiels demonstrieren.

Das studentische Projekt war sehr ehrgeizig angelegt, galt es doch vielfältige Teiltechnologien (XML, SOAP, WSDL, UDDI, Java, Apache Tomcat, Axis, SAP–Java–Connector, ABAP etc.) zu erlernen und diese praktisch zu nutzen.

Im Zentrum stand die Web Services–Technologie. Web Services sind, kurz gesagt, lose gekoppelte Software–Komponenten mit einer Schnittstellenbeschreibung in einem XML–Dialekt (WSDL). Sie ermöglichen die Integration von Systemen, die auf verschiedensten Plattformen implementiert sind.

War schon die Einarbeitung in die Web Services–Technologie bereits anspruchsvoll, so forderte das Anwendungsszenario höchste Ansprüche an die Schnittstellentechnologie und machte eine tiefgehende Einarbeitung in das SAP R/3–System erforderlich.

Das Anwendungsziel war, mit Hilfe von Web Services, eine Kreuzfahrtbuchung in einem SAP–System zu realisieren. Damit soll die Abfrage nach buchbaren Kreuzfahrten, die Buchung von Kreuzfahrten usw. in einem fiktiven Kreuzfahrtbuchungssystem über ein SAP–System ermöglicht werden. Hierzu war eine Schnittstelle auf der Basis des Java Connectors zu realisieren. Dazu waren vielfältige Vorarbeiten notwendig, wie: ein Kreuzfahrtbuchungssystem im SAP–System zu realisieren, das notwendige Datenmodell zu entwickeln, etc.

Das Projekt hatte eine Laufzeit von 10 Monaten und bestand aus verschiedenen Phasen:

Phase 1: Einarbeitung in die Basistechnologien (gegenseitige Schulung und Erarbeitung von Schulungsunterlagen)

Phase 2: Jedes studentische Mitglied des Projekts hatte eigenständig eine Beispiel Web Service–Anwendung zu erstellen, zu dokumentieren und vorzuführen. Damit sollte sichergestellt werden, dass jedes Gruppenmitglied in der Lage ist, einen Webservice zu entwickeln, zu beschreiben, Clients zu generieren, etc.

Phase 3: Diese Phase bestand darin, das Anwendungsszenario „Kreuzfahrtbuchung“ mit Hilfe von Web Services in Verbindung mit dem SAP–System zu konzeptionieren und beispielhaft anzuwenden.

Neben dem fachlichen Ziel wurden weitere Lernziele verfolgt. Die Mitglieder sollten praktische Erfahrungen in der Teamarbeit machen und das Vorhaben mittels Projektmanagementwerkzeugen und –ansätzen planen und umsetzen.

An dieser Stelle ist besonders hervorzuheben, dass das Vorhaben ohne die hervorragende und fachlich hochqualifizierte Unterstützung von Herr Dipl.–Ing. Alfred Schmidt zu einem solch großen Reifestadium nicht hätte geführt werden können. Ihm gebührt großer Dank. Aber auch die Studierenden haben stets hochmotiviert, in einer unzweifelhaft sehr gelungen Teamarbeit das ehrgeizige Vorhaben zu einem äußerst positiven Abschluß gebracht. Es hat große Freude gemacht, mit ihnen zu arbeiten. Anhand der vorgelegten Projektdokumentation kann das immense Engagement der StudentInnen abgelesen werden. Sie sind für das gelungene Projekt zu beglückwünschen.

Prof. Dr. Viefhues Bremerhaven, den 15.03.2005

2. Projektbeschreibung

2.1. Projekterwartung

Das Ergebnis für das *erfolgreich* abgeschlossene Projekt stellt sich wie folgt dar:

Ein Internet–Portal ist erstellt worden, über das eine Kreuzfahrtbuchung via Web Service erfolgen kann, einschließlich einer Informationsseite über Kreuzfahrtbuchungsmöglichkeiten (ein lesender und ein schreibender Zugriff). Das Portal ist auf einem Server im IECCB der Hochschule Bremerhaven bereitgestellt und von außen erreichbar. Ebenfalls ist eine Informationsseite über das Projekt entworfen und in das Portal integriert worden.

2.2. Projektauftrag und Ziele

2.2.1. Auftrag

Um das Projekt erfolgreich abzuschließen, sollen Geschäftsprozesse über das Internet mit Java– und XML–basierenden Web Services über ein SAP–System am Beispiel von Kreuzfahrtbuchungen durchgeführt werden können.

Dabei gilt es mindestens zwei Geschäftsprozesse (Kreuzfahrtinformation und Kreuzfahrtbuchung) mittels Web Service technisch zu realisieren und dokumentieren.

Genutzt werden sollen die standardisierten Techniken für Web Services:

`XML`, `SOAP`, `WSDL`, `UDDI`, `Java`, `Apache Tomcat Server`, `JAVA Server Pages`, `AXIS`, `JAVA Servlets`, `SAP JAVA Connector`, `ABAP`, `HTTP` und `TCP Monitoring`.

2.2.2. Ziele

Das Projekt untergliedert sich in zwei zu erreichende, aufeinander aufbauende Ziele. Folgend sind die Schritte aufgezählt, die innerhalb der festgelegten Projektzeit abzuarbeiten sind. Dabei ist in der angegebenen Reihenfolge vorzugehen:

1. Schritt: Schulung; Entwicklung Web Service ohne SAP–System–Programmierung
2. Schritt: Mit SAP–System–Programmierung: Datenhaltung in SAP; Systemintegration mittels JCo

Abgeschlossen ist das Projekt mit der Erstellung eines eigenständigen Web Services ohne SAP–Verbindung, wenn eine Kreuzfahrtbuchung durchgeführt werden kann. Dies ist aber die mindeste Umsetzung der Projektaufgabe.

Persönliche Ziele

Natürlich sollten die persönlichen Ziele der Mitarbeiter bei der Teilnahme an diesem Projekt nicht vernachlässigt werden. Obwohl die eigentlichen Projektziele Vorrang haben, sollten persönliche Ziele, wie bspw. das *Erlernen neuer Technologien*, *gute Arbeit* abzuliefern und letztendlich auch eine *gute Note* im Vordergrund stehen. Ebenfalls soll das Projekt auch bei einer Diplomarbeitssondierung behilflich sein.

3. Terminplanung

Das Projekt erstreckt sich über einen Gesamtzeitraum von zwei Semestern und ist in zwei Phasen unterteilt: zum Einen die Schulungsphase, welche im ersten Zeitabschnitt durchgeführt wird, und zum Anderen die Realisierungsphase, die an den erabeiteten Ergebnissen der Schulung im darauf folgenden Abschnitt anknüpft.

Projektbeginn ist am 25.03.2004, Projektende am 15.03.2005.

Folgende Abbildung zeigt den Ablauf des Projekts und die Unterteilungen der einzelnen Phasen im Detail:

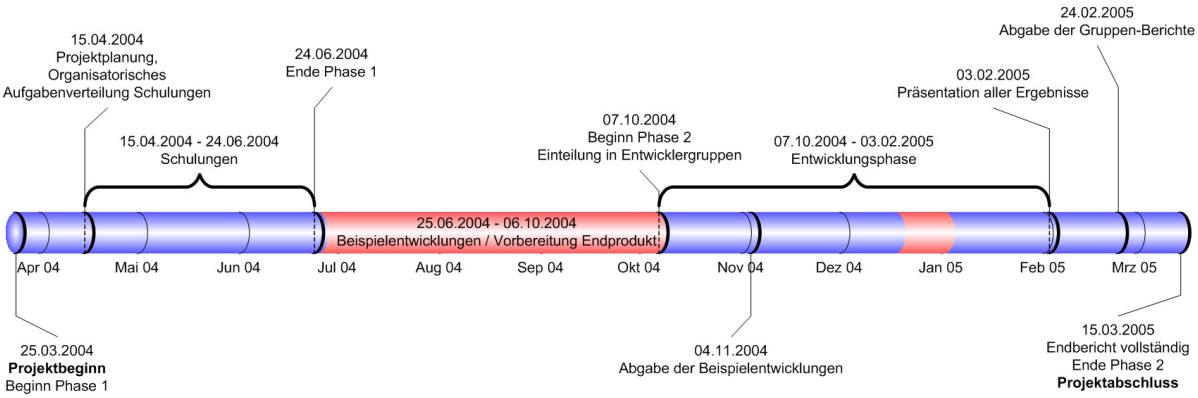


Abbildung 3.1.: Zeitplanung

Inhalte der Schulungsphase:

- **22.04.2004** Vorführung und Schulung Lotus–Notes
- **29.04.2004** XML–Schulung Teil 1
- **06.05.2004** XML–Schulung Teil 2
- **13.05.2004** XML–Schulung Teil 3, Vorführung/Schulung Eclipse
- **27.05.2004** Tomcat / JAVA-Servlets–Schulung, JSP–Schulung
- **03.06.2004** SOAP–Schulung, Vorstellung AXIS
- **10.06.2004** WSDL–Schulung, UDDI–Schulung

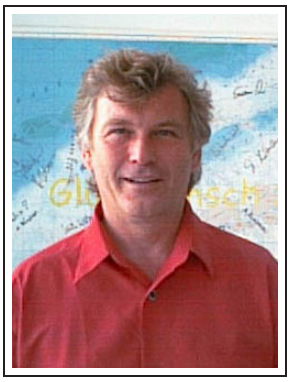
Inhalte der Realisierungsphase:

- **14.10.2004** Datenmodell: Diskussion des Entwurfs; Schulung SAP–Connector; Konzeption Intermediary
- **21.10.2004** Diskussion aller Gruppen für den Entwurf ihrer Aufgaben; Festlegung Portal / Portalstandard; Schulung SAP–Connector
- **28.10.2004** Vorstellung der Konzepte der Gruppen
- **04.11.2004** Präsentation der individuellen Web Services (Beispielentwicklung); Datenbank funktionstüchtig mit Testdaten
- **11.11.2004** Schnittstelle zu SAP läuft; Datenzugriff per Java-Connector auf SAP möglich
- **02.12.2004** Präsentation aktueller Stand der Entwicklergruppen mit ersten Programmierergebnissen

4. Mitglieder

Gesamtleitung

Prof. Dr. Dieter Viefhues–Veensma



„Die Web Services–Technologie ist eine noch sehr junge Disziplin: die wichtigsten Standards befinden sich zum Teil noch in der Normierung, die Dokumentation dieser Technologie ist häufig unzureichend und es liegen nur wenige dokumentierte Anwendungserfahrungen vor. In diesem Umfeld wurde ein studentisches Jahresprojekt definiert. Bei der Festlegung des Gesamvorhabens bestanden zunächst erhebliche Zweifel, ob das anspruchsvolle Lern– und Entwicklungsziel erreicht werden könnte. Die konkrete Umsetzung hat alle Erwartungen übertroffen. Zwölf höchst motivierte StudentInnen haben an dem Vorhaben mit großer Motivation und mit großem Engagement mitgewirkt. Sie haben ein Teamprojekt realisiert, in dem auf einem höchsten technischen und konzeptionellen Niveau ein sehr anspruchsvolles Anwendungsproblem gelöst wird. Die Teamarbeit war so hervorragend.

Den Studierenden ist höchstes Lob für die intensive Arbeit an dem Projekt auszusprechen. Es ist gelungen, dass sich alle Mitglieder des Projekts in die vielfältigen Technologien (XML, Web Services, Java Connector, SAP–ABAP – Programmierung etc.) einarbeiten konnten. Jedes Mitglied musste einen eigenen Web Service entwickeln und veröffentlichen. Sie mussten letztlich gemeinsam ein Kreuzfahrtbuchungssystem realisieren.

Die Dokumentation ist vorbildlich gelungen und hat durch ihre Web–Veröffentlichung die besonders hervorzuhebende Wirkung, in dem nachfolgende Studentengruppen auf die erarbeiteten Ergebnisse aufbauen können.

Allen Beteiligten gebührt großes Lob. Als Projektleiter habe ich mit großer Freude und Respekt vor den Fähigkeiten, dem Engagement der Studierenden an diesem Vorhaben mitgewirkt.“

Dipl.–Ing. Alfred Schmidt

„*Der Weg ist das Ziel!* Wenn die Dinge richtig laufen, dann lernen wir voneinander und wachsen aneinander. Und genauso war es in diesem Projekt. Anfangs war ich skeptisch, ob alle mitziehen würden und im Grunde genommen hatte ich einige Teilziele des Projekts abgeschrieben. Spätestens nach Ende des ersten Projektsemesters wurde klar, dass dieses Projekt erfolgreich abgeschlossen werden würde. Erstens beeindruckte mich der Zusammenhalt der Projektmitglieder und zweitens die fachliche Kompetenz innerhalb der Gruppe. Die eigentlichen Highlights waren im zweiten Projektsemester festzustellen, z.B. das eigenständige Bewältigen des Axis Typemapping sowie der Umgang mit Intermediaries und vieles andere. Mich freut insbesondere, dass eine Reihe von Diplomarbeiten aus dem Projekt hervorgegangen sind. Ich hoffe, dass alle Mitglieder einen theoretischen und praktischen Nutzen für das spätere Berufsleben aus dem Projekt ziehen konnten. Abschließend möchte ich noch sagen, dass ich einen ungeheuren Spaß am Projekt hatte, sehr viel dazu gelernt habe und hoffe, mit allen in Kontakt zu bleiben.“

Studentischer Projektleiter

Bastian Onken

WSDL Schulung, Web Service „Textanalysierer“, Team: Intermediary

„Das BizWeb–Projekt hat mir gezeigt, welchen hohen Aufwand, aber auch welches erfreuliches Ergebnis ein einjähriges Hochschulprojekt haben kann. Ich als studentischer Projektleiter, und damit als Ansprechpartner und Verantwortungsträger, habe dabei eine besondere Erfahrung mitnehmen können. Denn bis zu diesem Projekt hatte ich bisher keine vergleichbare Funktion zu bestreiten.
Insgesamt gesehen ist das Projekt für mich eine besondere Erfahrung zum Abschluss des Studiums, von dem ich nur Gutes berichten kann. Obwohl es sehr intensive, zeitraubende und anstrengende Teilstrecken auf dem Weg bis zum erfolgreichen Abschluss gab und man anfangs nicht wusste, was auf mich und meine Mitarbeiter zukommt, ist dennoch enorm neues Wissen angeeignet und an Erfahrung hinzugewonnen worden, welche uns in unserer nahen beruflichen Zukunft von der Masse absetzt.
Da ich durch mein Amt für die Organisation der Dokumente und die abschließende Endberichterstellung verantwortlich war, konnte ich besonders auf dem Gebiet der Dokumentation Erfahrung sammeln: Der Endbericht ist in dem professionellen Textsatzsystem LaTeX gesetzt und umfasst rund 550 Seiten – Einen besseren Lerneffekt für das Erstellen meiner baldigen Diplomarbeit hätte ich kaum erzielen können.
Besonders gefreut hat mich, dass das Alfred–Wegener–Institut in das Projekt integriert werden konnte und somit die Technik der Web Services durch das Implementieren von Intermediaries voll ausgereizt wurde.

Abschließend möchte ich nun sagen, dass mir das Projekt sehr viel Spaß und Freude bereitet hat und es mich mit Stolz erfüllt dabei gewesen zu sein. Ich hoffe, die Gruppe verliert sich nun nicht in der nahen Zukunft aus den Augen, denn solch willenstarke und vorantreibende Mitarbeiter, wie sie das Projekt hatte, die es auch noch geschafft haben erfolgreich im Team zu arbeiten, gibt es sehr selten.“

Projektmitarbeiter

Benny Bräuer

Lotus Notes Schulung / WSDL Schulung, Web Service „Stückliste“, Team: Intermediary

„Die Arbeit in einem solch umfangreichen Projekt zählt zu den Höhepunkten des Studiums. In diesem einjährigen Zeitraum lernt man, was es heisst, in einem Projekt zu arbeiten, wie es verwaltet werden muss, welche Schwierigkeiten und Probleme zu bewältigen es gilt etc. – und das ersteinmal ohne die fachlichen Inhalte, welche eine weitere Komponente darstellen.
Zunächst ist zu sagen, daß die Arbeit zwischen den Gruppenmitglieder ausgezeichnet funktionierte, bis auf verschiedene Schwierigkeiten bei der Einhaltung von Terminen (was verständlich ist, denn nebenbei waren ja zusätzlich noch Aufgaben anderer Vorlesungen zu erledigen) lief alles reibungslos. Jedes Projektmitglied, sei es bei den Schulungen, den Beispiel-Web Services oder der eigentlichen Projektaufgabe, erfüllte seine fachlichen Aufgaben mit Bravour.
Für mich selbst kann ich sagen, daß die Arbeit mit den verwendeten Technologien eine Erfahrung war, welche ich nicht missen möchte. Das Entwickeln einer solchen Anwendung, welche auch einen gewissen Bezug zur wirtschaftlichen Praxis hat, war kurz vor dem Abschluss des Studiums eine interessante Herausforderung an das bislang erworbene Können. Für die Bewerkstelligung der Aufgabe sowie die gute Arbeit und den Zusammenhalt in der Gruppe möchte ich mich bei allen Mitgliedern sowie der Administrative herzlich bedanken.“

Dominik Buhlmann

Tomcat/Servlet Schulung, Web Service „Währungen“, Team: JCo

„Zu Beginn des 6. Semesters war ich sehr froh in diesem Projekt gelandet zu sein, denn viele Studenten wollten mitmachen und es war klar das hier etwas neues und sehr interessantes gemacht werden soll. Es wurde eine relativ große Gruppe, die sich gleich am Anfang zusammengerauft und ins Zeug gelegt hat.
Es war am Anfang natürlich ein sehr großer Berg von Aufgaben, den es zu bewältigen galt um das Endprodukt herauszubekommen, aber ich hatte so gut wie nie die Befürchtung, dass wir das nicht schaffen könnten. Ein Grund dafür war die gute Atmosphäre, die jedes Mal herrschte, wenn wir zusammen kamen. Ein weiterer Grund war die Tatsache, dass das gesamte Projekt von Anfang an sehr gut konzipiert und der Ablauf sorgfältig durchdacht war.
Fachlich hat das Projekt sehr viel gebracht. Das Gebiet der Web Services war für mich völliges Neuland. Auch bei dem Thema, dass ich am Anfang bearbeitete (Tomcat–Server und Servlets) fing ich bei Null an. Aber genau das hat ja den Reiz des Ganzen ausgemacht. Durch die Schulungen in den anderen Themengebieten, wie z.B. XML und SOAP, wurde ein sehr guter Überblick über die zu verwendenden Technologien gegeben und sie stellten insgesamt eine wichtige Grundlage für die Erstellung des ersten, kleinen Beispiel–Web–Services dar. Mit dem Projekt lernte ich auch die Entwicklungsumgebung „Eclipse“ kennen, welche ich jetzt nicht mehr missen möchte. Die Herausforderung des zweiten Projektsemesters war es, die Anbindung des Web–Services an das hochschuleigene SAP–System mit dem Java–Connector zu realisieren. Auch die Lösung dieser Aufgabe war hochinteressant, zum einen weil die Zusammenarbeit mit anderen Projekt–Untergruppen sehr wichtig war und zum anderen, weil wieder eine für mich neue Technologie eingesetzt wurde.
Das Projekt war auch über die fachlichen Inhalte hinaus lehrreich. Das Arbeiten im Team sticht dabei hervor. Die Zusammenarbeit innerhalb des Projekts und innerhalb der einzelnen Untergruppen hat immer sehr gut funktioniert. Das Lösen von Problemen, welche mehrere Untergruppen betrafen, ist der beste Beweis für eine gute Teamarbeit.
Die Arbeit im Projekt hat über die gesamte Länge von zwei Semestern sehr großen Spaß gemacht und ich würde die Entscheidung für ein solches Projekt immer wieder treffen.
Zum Schluss möchte ich mich bei allen Teammitgliedern für die gute Zusammenarbeit bedanken. Besonders hervorzuheben ist das stets hohe Engagement und die Unterstützung von Alfred Schmidt und Prof. Dr. Dieter Viefhues.
Ein solches, semesterübergreifendes Projekt sollte auf jeden Fall weiterhin Bestandteil der Vorlesungsplanung bleiben, da dort unverzichtbare Erfahrungen gesammelt werden können.“

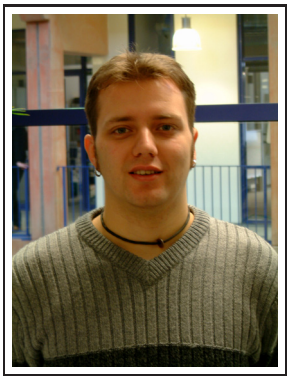
Christian Eggers

Eclipse Schulung, Web Service „Aktienkurs“, Team: JCo

„*Ein Projekt über zwei Semester* hieß es am Anfang des sechsten Semesters auf dem Stundenplan. Alle bisherigen Projekte während des Studiums hatten einen überschaubaren Bearbeitungszeitraum von ein paar Wochen mit maximal drei Personen; jetzt waren es rund viermal so viele Personen und einige Wochen mehr! Kann so etwas gut gehen? Mit gutem Gewissen kann ich im Nachhinein sagen: Ja, es kann gut gehen!
Im Gegensatz zu anderen Projekten war die Zusammenstellung der Projektmitglieder bezogen auf die Informatikschwerpunkte weniger stark gemischt. Besser gesagt bestand bis auf einen Informatiker der ganze Trupp aus Wirtschaftsinformatikern, was aber nicht als negativ ausgelegt werden soll: Die gesamte Gruppe hat sich von Beginn des des Projektes an ins Zeug gelegt, die starke Zusammenarbeit war von Anfang an zu sehen, nicht zuletzt, weil jeder jeden kannte. Es gab fast keine Unruhestifter.
Das Thema Webservices war für mich zu Anfang komplettes Neuland. Bis dahin war ich weder mit XML und WSDL, noch mit Axis oder JSP in Berührung gekommen. Jedoch ist das genannte Spektrum und noch einiges darüber hinaus an Techniken und Tools notwendig, um mit Webservices sicher umgehen zu können. Diese erworbenen Kenntnisse konnte ich im siebten Semester in der einen oder anderen Vorlesung gewinnbringend verwenden und erweitern. Wenn auch teilweise etwas nervenaufreibend hat sich die Arbeit mit dem Eclipse–Framework mehr als ausgezahlt. Es stellt ein mächtiges Werkzeug dar, das mit einer Fülle an PlugIns nahezu unbegrenzt erweitert werden kann.
Die Planung des Gesamtprojektes wurde sehr sorgfältig während der ersten Treffen durchdacht, so dass jedes Teammitglied prinzipiell wusste, was es die beiden Semester über zu tun hatte. Dazu gehörten die Schulungen im ersten Semester, die schon sehr interessant waren, wenn auch ziemlich geballt und lang, so dass man sich am Ende eines Tages kaum noch auf den Inhalt konzentrieren konnte. Man sollte diesen Part etwas anders gestalten, aber wie, ist die Frage. Leider ist mir diesbezüglich keine bessere Idee gekommen.
Nicht zuletzt des betreuenden Professors und der mehr als tatkräftigen und kompetent–professionellen Unterstützung des technischen Angestellten war es zu verdanken, dass das Projekt nahezu reibungslos die zwei Semester über voranschritt und es zu keinen nennenswerten Unterbrechungen oder thematischen Problemen kam.
Meine anfänglichen Befürchtungen, ob so ein relativ großes Projekt über zwei Semester glatt über die Bühne gehen würde, waren demzufolge unnötig, was sehr viel mit dem gesamten Team zusammenhing. Die zwei Semester haben diesbezüglich viel Arbeit, aber auch Spaß gemacht. Hoffentlich bleibt diese Planung weiterhin im Stundenplan für die Semester nach uns erhalten.“

Remo Fulle

XML Schulung, Web Service „Flugbuchungsbestätigung“, Team: Portal



„Die Arbeit in diesem Projekt war eine der Interessantesten während der gesamten Studienzeit. Zu Beginn der Projektarbeit wirkten die große Menge an zu nutzenden Techniken und Programmen, sowie die zu lösenden Aufgaben sehr unübersichtlich, unkoordiniert und fast abschreckend. Im Laufe der ersten Schulungen verstärkte sich dieser Eindruck noch, da schnell das Gefühl aufkam, alles perfekt beherrschen zu müssen. Nachdem aber die Arbeiten an den ersten Aufgaben begannen, legte sich die Nervosität dann doch recht schnell, da es wirklich Freude machte, mit den einzelnen Techniken zu arbeiten. Als der erste Web Service dann erstellt war, wurden alle Zusammenhänge mit einem Mal klar und alle unübersichtlichen Einzelbereiche fügten sich zu einem schlüssigen Gesamtbild zusammen. Diese Erfahrung war sehr interessant und lehrreich.
Im weiteren Verlauf war zu erkennen, dass alle Projektmitarbeiter sehr motiviert waren. Überhaupt hat die Teamarbeit in dieser Gruppe sehr viel Spaß gemacht. Ich kann mich nicht erinnern, dass es einmal zu Unstimmigkeiten kam, auch nicht wenn der Druck einmal größer wurde. Auch der Umgang mit den Projektverantwortlichen war immer sehr angenehm und recht locker. Das gute Gesamtklima hat sich insgesamt sehr positiv auf die Motivation der Gruppe ausgewirkt, was hoffentlich am Ergebnis zu sehen ist.
Alles in allem bin ich mehr als froh, mich für dieses Projekt gemeldet zu haben. Ich konnte mein Wissen in Bezug auf Web Services, die dazu gehörigen Techniken, sowie im Bezug auf Team– und Projektarbeit enorm vertiefen und ausbauen. Im kommenden Berufsleben werde ich vieles von dem erlernten gebrauchen können.
Bleibt zu hoffen, dass wir im Berufsleben mit den Teams genau so viel Glück haben, wie im Projekt...“

Marco Junge

SOAP Schulung, Web Service „Kreuzfahrtliste“, Team: Web Service

„Im Rahmen des Projektes „BizWeb“ an der Hochschule Bremerhaven sollte ein Web Service implementiert werden, der eine Liste der angebotenen Kreuzfahrten ausgibt und mit dem einzelne Kreuzfahrten gebucht werden können. Als Datenbasis für die Transaktionen (Backend) dient dabei ein SAP R/3–System. Bei der Implementierung wurden die Programmiersprache Java mit der IDE Eclipse und der Applikationsserver Tomcat mit AXIS eingesetzt. Alle verwendeten Technologien (mit Ausnahme der R/3–Software) sind Open Source–Projekte.
Meine Aufgaben waren die Schulung von „SOAP in der Java–Welt“ zusammen mit Christian Lefke im ersten Teil und die Implementierung und das Deployment des Web–Services in Zusammenarbeit mit Christian Lefke und Jan Stelmaszek im zweiten Teil.
Der Erfolg des Projektes lässt sich auf die sehr gute Zusammenarbeit zwischen allen Projektteil-

nehmern zurückführen. Die vorbildliche Projektleitung, die reibungslose Kommunikation und Hilfsbereitschaft sowie der außerordentliche Arbeitseinsatz aller Beteiligten waren ausschlaggebend für den rechtzeitigen Projektabschluss.
Die fachlichen und persönlichen Erfahrungen, die ich in diesem Projekt gewinnen konnte, sind eine gute Grundlage für den weiteren Berufsweg. Meines Erachtens bieten die geleisteten Arbeiten auch eine gute Basis für Folgeprojekte. An dieser Stelle möchte ich mich noch einmal bei allen Beteiligten für die hervorragende Projektarbeit bedanken.“

Christian Lefke

SOAP Schulung, Web Service „Taschenrechner“, Team: Web Service

„Meiner Meinung nach war die Abbildung eines Geschäftsprozesse einer Kreuzfahrtbuchung (Informieren und Buchen) mittels Web Services in Zusammenarbeit mit der Standardsoftware SAP R/3 eine sehr zeitintensive, aber auch herausfordernde Aufgabe, die Spaß gemacht hat.
An Hand dieses Projektes habe ich gelernt, die theoretischen Kenntnisse aus der ersten Projektphase und dem zusätzlichen Studium von Literatur aus verschieden Quellen in eine praktische Arbeit (Einzelarbeit und Projektarbeit) umzusetzen.
Für mein späteres Berufsleben war der Themenkomplex Web Services ein wichtiger Aspekt, da durch die zunehmende Digitalisierung der Geschäftsprozesse der Schwerpunkt Web Services in Zukunft eine tragende Rolle spielen wird.
Sowohl die Zusammenarbeit in den einzelnen Teams als auch in dem gesamten Projekt war wirklich hervorragend. Wir mussten nicht darauf hoffen, dass der Eine die ihm zugeteilte Arbeit erfüllt, sondern wir konnten sicher darauf vertrauen. Jede Teilaufgabe wurde miteinander besprochen und für das weitere Bearbeiten des Projektes berücksichtigt.“

Viktor Schlegel

JSP Schulung, Web Service „Kreuzfahrtbuchung“, Team: Datenbank

„Ich fand das Projekt echt interessant, da sehr viele Technologien miteinander kombiniert wurden. Besonders die gelungene Zusammenarbeit mit Standardsoftware wie SAP R/3 und sehr neue Technologien wie SOAP, XML, UDDI, Tomcat usw., die mit dem Freeware Entwicklungstool „Eclipse“ zusammengebracht wurden, fand ich genial. Ich hätte nach den Schulungen und den Schwierigkeiten am Anfang wirklich nicht gedacht, dass es überhaupt funktionieren würde. Wobei ich nicht an useren Datenbank-Teil denke, sonderen an alles was nach dem JConnector kommt.
Schlussendlich hat sich aber aufgrund der vorhandenen Schnittstellen die SAP R/3 mit dem JConnector und den Schnittstellen die Java mit sich bringt alles zusammengearbeitet. Was nicht zuletzt dem Teamwork zu verdanken ist, denn jeder hatte irgendwie eine passende Idee gehabt, da jeder sein „Spezialgebiet“ aus den Schulungen heraus hatte.

Abschließend möchte ich hinzufügen, dass das Projekt aus meiner Sicht das interessanteste von allen gebotenen Projekten war und hoffentlich weiterentwickelt wird.“

Christopher Schrade

XML Schulung, Web Service „Literatursuche“, Team: Portal

„Das Projekt hat zunächst einmal unheimlichen Spaß gemacht. Die Teamarbeit war einmalig und hat gezeigt was möglich ist, wenn innerhalb eines Projekts alle reibungslos ohne größere Konflikte zusammenarbeiten. Probleme die aufgetreten sind wurden sofort gemeinschaftlich gelöst. Vorbildlich fand ich, dass viele, sobald sie mit ihren Arbeiten fertig waren, sich sofort erkundigt haben wo noch etwas zu tun ist und dort ihre Hilfe angeboten haben und sich nicht auf ihrer vollbrachten Arbeit ausgeruht haben.
Inhaltlich habe ich während des Projekts sehr viel gelernt und würde behaupten mich im Bereich der Web Services, XML und dem Apache Cocoon Framework ohne Probleme zurechtzufinden und damit in Zukunft arbeiten zu können.
Die Schulungen zu Beginn des Projekts waren äußerst sinnvoll. Leider blieben meist nur Grundzüge hängen, so dass man sich, sobald man selbst mit diesem Thema zu tun hatte, zusätzlich fast vollständig einarbeiten musste. Ideal war hierbei das man sofort von den Referenten der entsprechenden Themen Hilfe bekam.
Das Erstellen des eigenen Web Services verlief ohne Probleme.
Das Erstellen des BizWeb Portals verlief ebenfalls problemlos, wobei ich häufig gemerkt habe, dass sich Cocoon erst in einem frühen Entwicklungsstadium befindet. So waren nicht alle von unserer Teilgruppe geplanten Umsetzungen durchzuführen. Im Großen und Ganzen halte ich das Ergebnis unserer Gruppe, wie auch das Gesamtergebnis für äußerst gelungen und denke, dass es eine ideale Grundlage für spätere Bewerbungen darstellt.“

Nicole Siedek

UDDI Schulung, Web Service „Hotelbuchung“, Team: Datenbank

„Das Projekt BizWeb beinhaltet die Web Service Entwicklung eines Geschäftsprozesses im Szenario der Kreuzfahrtbuchung. Die Gliederung des Projekts in zwei Phasen war sehr sinnvoll. Wichtige Termine wurde in unterschiedlichen Teilbereichen des Projekts durchgesprochen und vollendet. Es war komplett durchorganisiert und verlief durch die gute Zusammenarbeit der Kollegen planmäßig. Es ist vom Anfang bis zum Ende ein sehr anspruchsvolles Projekt gewesen und hat sehr viel Spaß gemacht. Mit der Auseinandersetzung unterschiedlichster Techniken (XML, SOAP, WSDL, UDDI, Java, Apache Tomcat-Server, Java-Server-Page-Engine, AXIS, JAVA-Servlets, SAP-Java-Connector, ABAP, HTTP-Protokoll, TCP-Monitoring) wurde ein sehr hoher Lernerfolg erzielt.
Für die Zukunft hoffe ich das die Hochschule weiterhin solche Projekte fördert.“

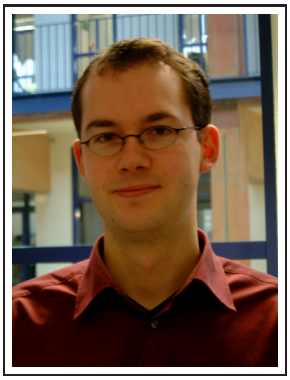
Jan Stelmaszek

Lotus Notes Schulung / UDDI Schulung, Web Service „Weltzeituhr“, Team: Web Service

„Das gesamte Projekt angefangen bei der Schulung bis hin zur Realisierung des Web Services erwies sich als äußerst aufwändig und zeitintensiv.
Der Anspruch war meiner Meinung nach durch die vielen unterschiedlich eingesetzten und benötigten Technologien durchaus hoch und forderte jeden einzelnen heraus. In der ersten Phase des Projektes war es ein wenig schwierig den Überblick zu behalten und die einzelnen Schulungen in den Gesamtkontext einzuordnen. Die gestellte Semesteraufgabe war aus meiner Sicht daher eine sehr gute Idee, um die erworbenen Kenntnisse anzuwenden und besser zu verstehen.
Die Realisierung des Web Services in der zweiten Phase des Projektes war meiner Meinung die eigentliche Herausforderung des Projektes und machte sehr viel Spaß. Trotz des relativ hohen Zeitdrucks, gegeben durch die enge Terminplanung, haben alle im Team super zusammengearbeitet und jeder für sich einen wichtigen Beitrag zur Endleistung beigesteuert.
Abschließend lässt sich sagen, dass dieses umfangreiche Projekt sehr viel Spaß gemacht hat und nicht nur die erworbenen Kenntnisse, sondern vielmehr die Zusammenarbeit in einer Projektgruppe meiner Meinung nach sehr hilfreich beim Einstieg in die Berufswelt sein werden.“

Thomas Wäsch

XML Schulung, Web Service „Produktsuche“, Team: Portal



„Zusammenfassend kann ich das Projekt als erfolgreich abgeschlossen bewerten. Für mich war es eine große Herausforderung an diesem Projekt teilnehmen zu dürfen und an einem der jeweils sehr komplexen Teilprojekte mitwirken zu können.
Ohne die perfekte Teamarbeit und Leistungsfähigkeit eines jeden Einzelnen hätten wir dieses gesamte Projekt nie zu Ende gebracht. Die vorher geplanten Meilensteine konnten Dank der Teamfähigkeit und Gruppendisziplin gut eingehalten werden. Die komplette Gruppe arbeitete aus meiner Sicht hervorragend miteinander, was bei einem Aufeinandertreffen von verschiedenen Charakteren nicht immer selbstverständlich ist. Für besonders sinnvoll hielt ich die Teilung des Projektes in Schulungs– und Realisierungsphase. Somit wurde jeder Projektteilnehmer in sämtliche verwendete Technologien eingearbeitet, bevor das eigentliche Projektziel realisiert wurde. Leider blieben bei den Schulungen durch die geballte Wissensvermittlung nur Grundzüge hängen. Nicht zu vergessen sind natürlich die zwei gemeinsamen Essen, die wiederum den guten Zusammenhalt der Gruppe widerspiegelten.
Trotz der teilweise viel zu verrichtenden Arbeit hat es sehr viel Spaß gemacht an diesem Projekt mitzuwirken. Ich bin davon überzeugt, dass jeder von uns viele neue Erkenntnisse mitnehmen konnte und dass das Projekt eine gute Bewerbungsgrundlage bietet.“

Teil II.

Einführung und Schulung

5. Einleitung

Um das Erlernen der neuartigen und komplexen Web Service-Technologien zu erleichtern, sind diverse Schulungen zu Beginn des Projektes durchgeführt worden. Diese zumeist intensiven Schulungen umfassten grundlegende aber auch sehr tiefgehende Informationen zu den verwendeten Technologien und sollten den Mitgliedern, die teilweise schon grundlegende Kenntnisse der Technologien vorweisen konnten, ein fundamentäres Wissen in den verschiedenen behandelten Themen vermitteln.

Hintergedanke war jener, dass sich die Mitglieder auf mindestens ein Gebiet spezialisieren, damit sie beim späteren Einsatz der entsprechenden Technologie im Endprodukt professionelle Lösungen erarbeiten.

Um Zeit zu sparen unterrichteten sich die Mitglieder dabei gegenseitig, denn je mehr Zeit für die zeitraubende Schulung verwendet wird, desto unangenehmer würde die Durchführung der Projektschritte am Ende werden.

Durch die Vorbildung der teilnehmenden Projektmitglieder mussten nur spezielle Themen geschult werden, auf die im Folgenden kurz eingegangen wird.

5.1. Überblick

Die Projektplanung, Kommunikation und Dokumentenverwaltung wurde mittels Lotus–Notes organisiert. Da dazu ein „Teamroom“ entworfen wurde, war es notwendig, eine kurze Schulung zur Anwendungsprogrammierung und Nutzung von Lotus–Notes durchzuführen.

Des Weiteren ist eine grundlegende Schulung in XML[1] notwendig gewesen, da dieses Thema im vorangegangenen Studienverlauf kein Bestandteil der Vorlesungen gewesen ist. Aus diesem Grund wurde diese Schulung als Einzige durch das breit gefächerte Spektrum in drei Themengebiete aufgeteilt: Grundlagen, Parser sowie XPATH mit Xalan.

Natürlich war es unumgänglich eine passende Entwicklungsumgebung für die gesamte anfallende Programmierung festzulegen. Das Open Source Projekt „Eclipse“ wurde als Entwicklungswerkzeug ausgewählt, zu dem ebenfalls eine kurze Schulung über Funktionen und Anwendung durchgeführt wurde.

[1] Extensible Markup Language

Die speziellen Schulungen, welche den Web Service betreffen, sind durch deren Komplexität in Schwerpunkte aufgeteilt worden: Tomcat (Applikationsserver für den zukünftigen Web Service); AXIS als Web Service Provider; Java Servlets und JSP[2] zur Programmierung der Oberflächen; SOAP[3], WSDL[4] und UDDI[5] für den Entwurf der Web Services.

Letztendlich wurde durch das komplizierte Zusammenspiel der Technologien umfangreiche Einstellungen an den Entwicklungsrechnern benötigt, so dass es notwendig war, die Projektmitglieder zur Einrichtung einer lauffähigen Umgebung zu schulen.

In den folgenden Kapiteln lassen sich diese umfassend dokumentierten Schulungen wiederfinden.

[2]Java Server Pages
[3]Simple Object Access Protocol
[4]Web Service Description Language
[5]Universal Description, Discovery and Integration

6. Lotus Notes

6.1. Lotus Notes

6.1.1. Erste Schritte

Im Rahmen der Schulung, die eine Einführung in das teamorientierte Arbeiten mit Lotus Notes geben soll, werden in diesem Arbeitspapier die notwendigen Konfigurationen des Clients erklärt. Nach abgeschlossener Installation wird der Benutzer beim Start von einem Assistenten bei der Konfiguration des Clients unterstützt. (Abb. 6.1)

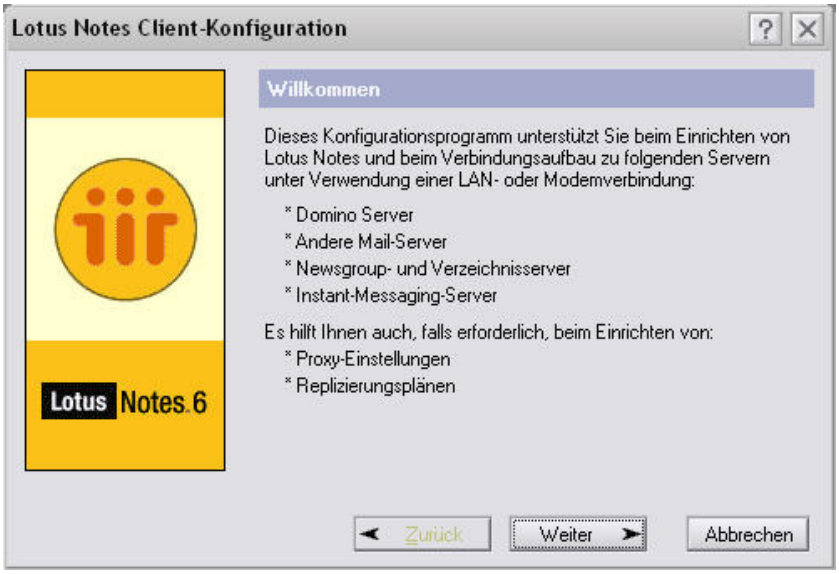


Abbildung 6.1.: Lotus Notes: Assistent Client Konfiguration

Der Benutzer klickt auf „Weiter“ und gelangt zur Eingabe der Benutzerinformation (Abb. 6.2). Die Eingabe des Namens kann beliebig sein. Das Kontrollkästchen „Verbindung zu einem Domino Server“ muss aktiviert werden. Die Serverangaben sind in diesem Fall nexus.hs-bremerhaven.de.

Abbildung 6.2.: Lotus Notes: Eingabe von Benutzerinformationen

Die Eingaben mit „Weiter“ bestätigen und im nächsten Schritt den Pfad der Benutzer–ID angeben. (Abb. 6.3)

Anmerkung: Die Benutzer–ID setzt sich in den meisten Fällen aus dem ersten Buchstaben des Vornamens und dem Nachnamen zusammen. Sollte diese Kombination mehr als acht Zeichen beinhalten wird der Rest abgeschnitten. Die Benutzer–ID's befinden sich auf dem FTP ftp://www.bwl.hs-bremerhaven.de/pub/people.

Abbildung 6.3.: Lotus Notes: Benutzer–ID

Die Meldung „Benutzer–ID in Installationsordner kopieren“ mit JA bestätigen, die serverseitig gespeicherte ID wird lokal auf dem Rechner kopiert. Wurde die ID gespeichert baut der Client eine Verbindung zum Nexus Server auf. Das Passwort ist bei der ersten Anmeldung **initinit**. Die Änderung des Passworts wird unter 6.1.2 erläutert.

Abbildung 6.4.: Lotus Notes: Passwortabfrage

Möchte man den Lotus Notes Client als „stand–alone“ Emailanwendung nutzen könnte man hier die Option Dienst: Internet Mail Server wählen (zum Beispiel um eine Verbindung zum T–Online Server aufzubauen). Für unsere Zwecke werden keine weiteren Dienste benötigt, so dass auf „Weiter“ geklickt werden kann.

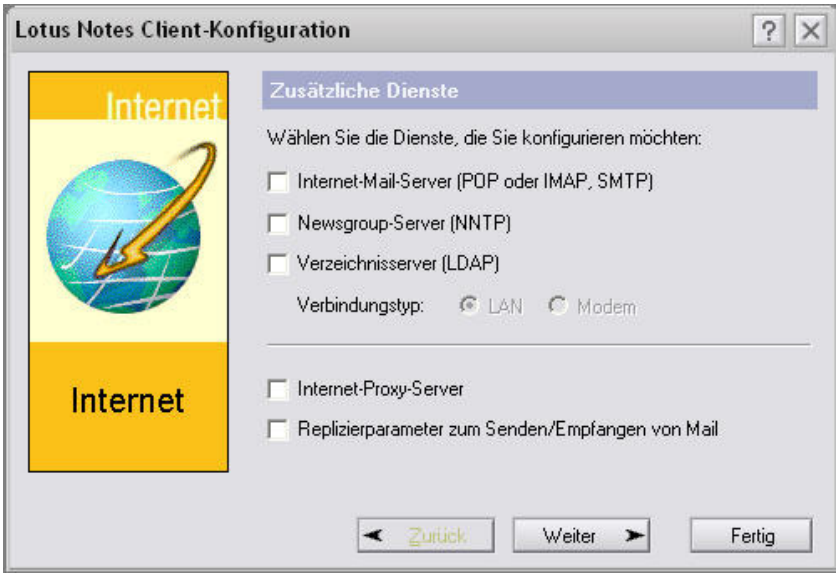


Abbildung 6.5.: Lotus Notes: Zusätzliche Dienste

Die ersten Einstellungen sind nun abgeschlossen. Die Frage, ob Lotus Notes als Standardemailprogramm verwendet werden soll, mit „Nein“ beantworten.

6.1.2. Passwort ändern

Das Passwort kann über den Menüpunkt: Datei →Sicherheit →Benutzersicherheit geändert werden.

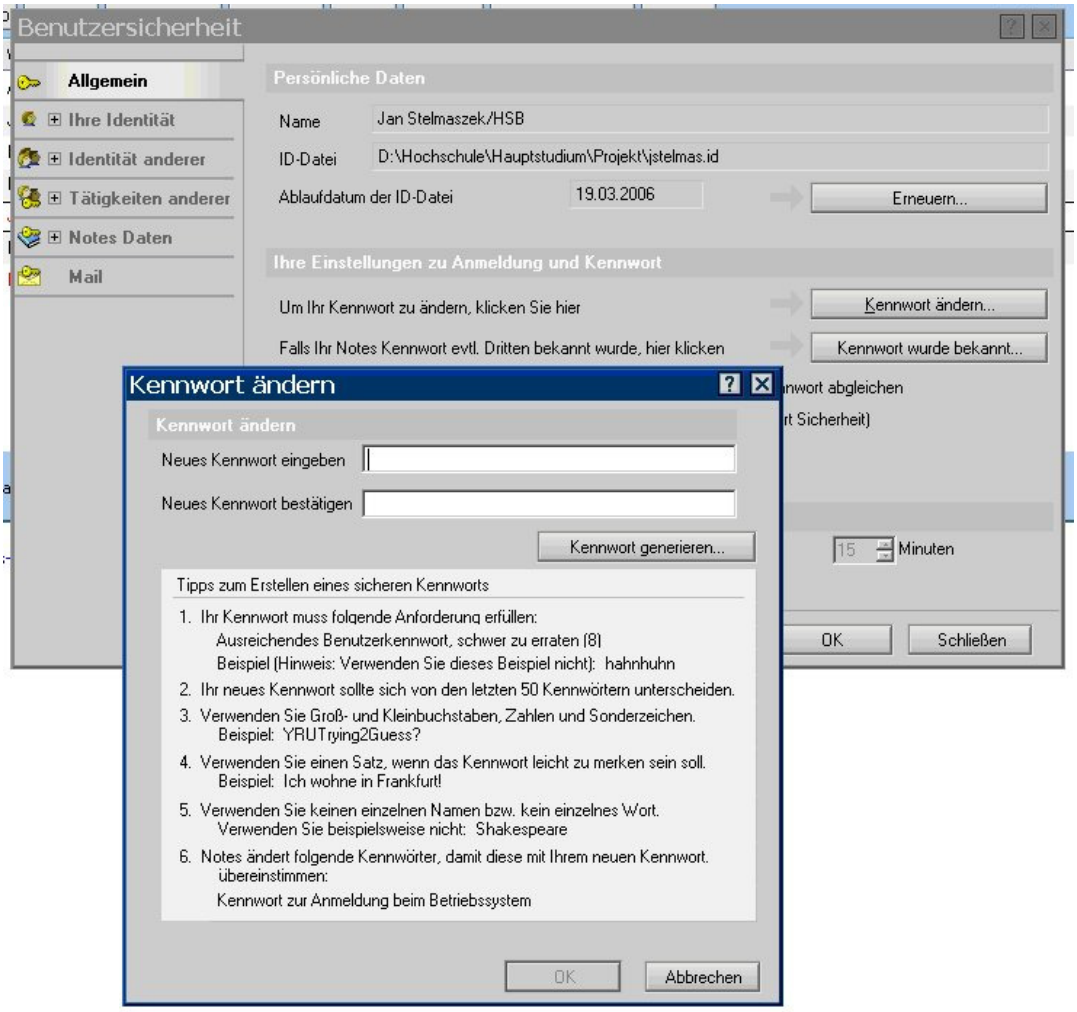


Abbildung 6.6.: Lotus Notes: Kennwort ändern

Anmerkung: Passwortänderungen werden in der aktuellen ID gespeichert und wirken sich lediglich auf diese ID aus. Sollte die ID an anderen Stellen gespeichert sein, gilt dort weiterhin das alte Passwort!

6.1.3. Einrichten der Arbeitsumgebung

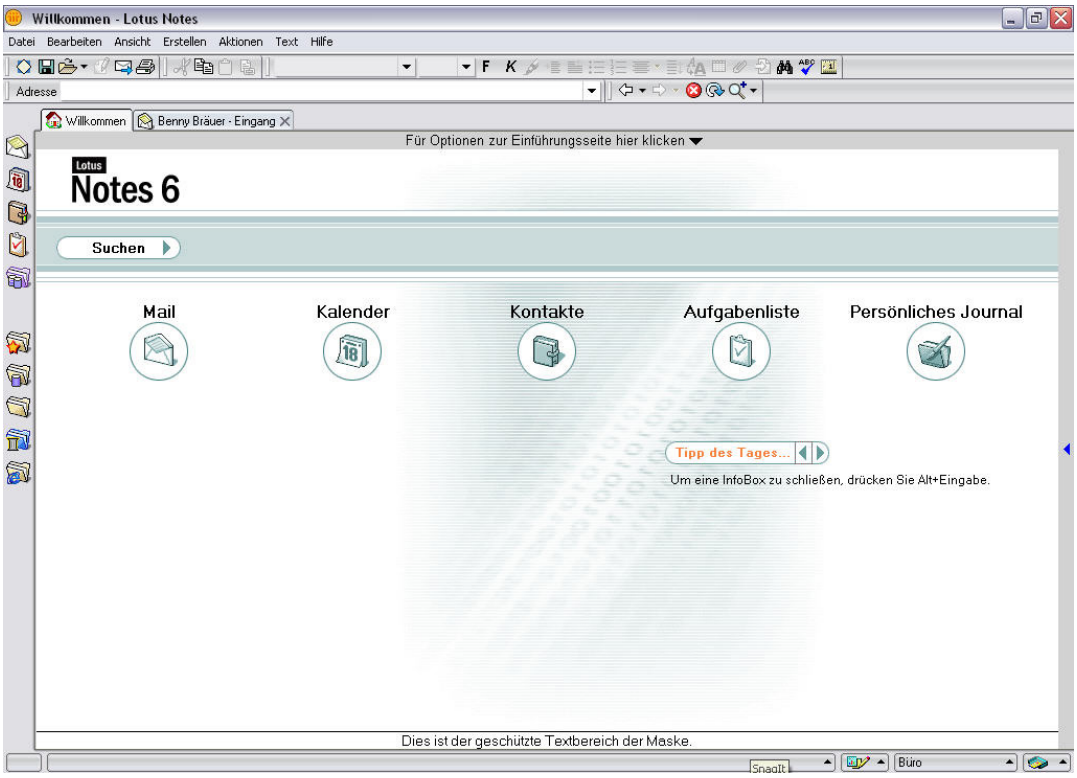


Abbildung 6.7.: Lotus Notes: Einführungsseite

Im nächsten Schritt wird das Einrichten einer Arbeitsumgebung erklärt. Zum Einrichten der Arbeitsumgebung wie folgt vorgehen: Datei →Mobil →Arbeitsumgebung. Die aktuell genutzte Arbeitsumgebung (Büro (Netzwerk)) markieren, und in das gleiche Verzeichnis kopieren.

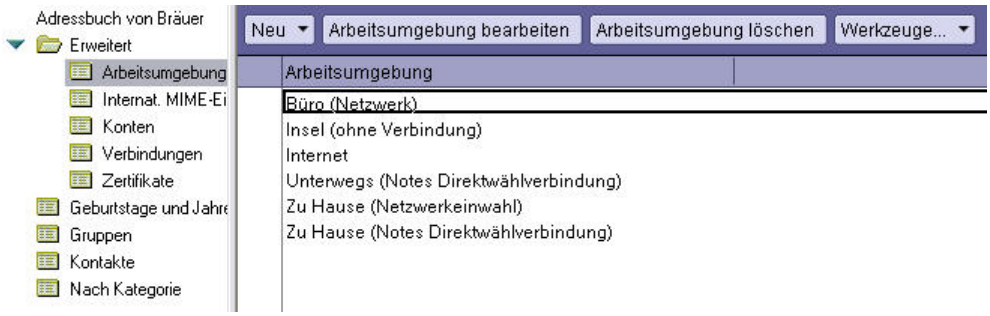


Abbildung 6.8.: Lotus Notes: Arbeitsumgebung

Die erstellte Kopie markieren und den Button „Arbeitsumgebung bearbeiten“ klicken. Im Register „Allgemein“ kann der Name der Arbeitsumgebung geändert werden (z. B. der eigene Name).

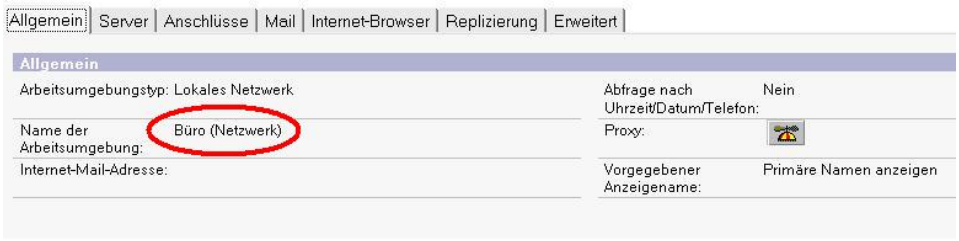


Abbildung 6.9.: Lotus Notes: Arbeitsumgebung bearbeiten

Im Register „Mail“ muss der Name der Mail–Datei geändert werden: **mail\Benutzer–ID** (siehe Abb. 6.9).

Abbildung 6.10.: Lotus Notes: Ändern der Mail–Datei

Die Arbeitsumgebung ist nun erfolgreich eingerichtet. Die aktuell eingestellte Arbeitsumgebung wird in der rechten Ecke des Notes Arbeitsbereiches angezeigt:

Abbildung 6.11.: Lotus Notes: Auswahl Arbeitsumgebung

Anmerkung: Die Arbeitsumgebung wird lokal auf dem Rechner gespeichert. Meldet sich der Benutzer das nächste Mal auf einem anderen Rechner an, so muss ein weiteres Mal die Arbeitsumgebung eingerichtet werden. Es empfiehlt sich daher nach Möglichkeit den gleichen Arbeitsplatz zu wählen.

6.1.4. Arbeitsbereich einrichten

Durch die Vielzahl der verschiedenen Nutzer von Lotus Notes im Labor für Standardsoftware empfiehlt es sich für das Projekt einen eigenen Arbeitsbereich einzurichten, in dem man die für das Projekt notwendigen Datenbanken geöffnet hat.

Zunächst einmal muss der Arbeitsbereich geöffnet werden. Dies geschieht durch Klicken des Icons Datenbanken in der links angeordneten Symbolleiste (siehe Abb. 6.12).

Abbildung 6.12.: Lotus Notes: Arbeitsbereich öffnen

Über einen Rechtsklick auf einen der farbig markierten Reiter kann der Benutzer die Eigenschaften des gewünschten Reiters festlegen. (Abb. 6.13)

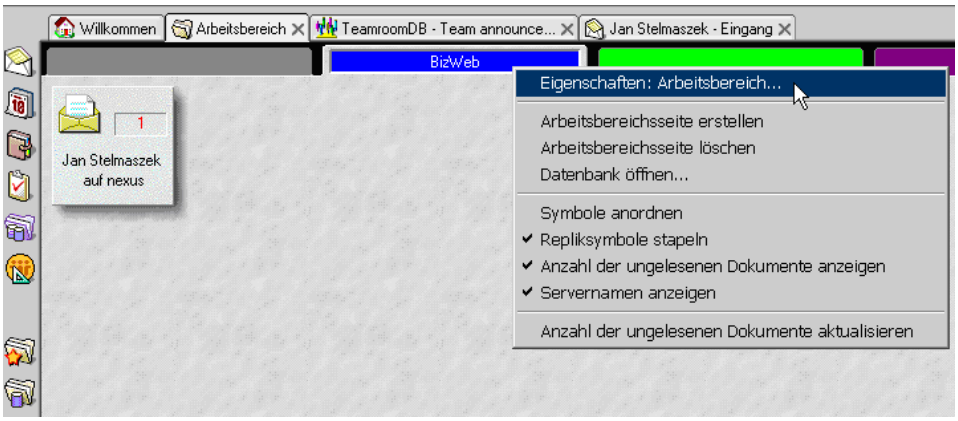


Abbildung 6.13.: Lotus Notes: Arbeitsbereich Eigenschaften

Hier können der Name sowie die Farbe des Arbeitsbereiches eingestellt werden (Abb. 6.14). Sind die Einstellungen abgeschlossen kann der Benutzer bereits geöffnete Datenbanken mittels drag and drop von einem Arbeitsbereich in den neu erstellten Bereich schieben (Abb. 6.15).

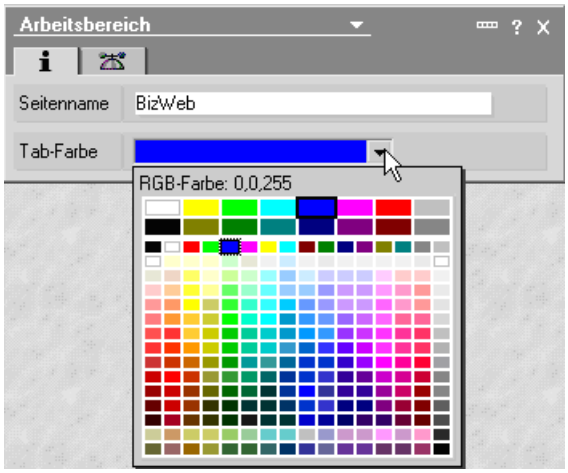


Abbildung 6.14.: Lotus Notes: Einstellungen Arbeitsbereich

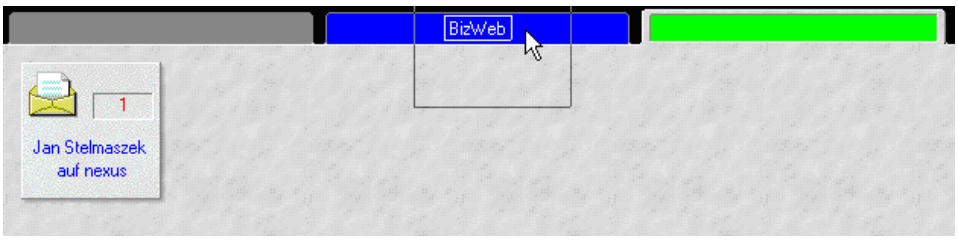


Abbildung 6.15.: Lotus Notes: Drag & Drop

6.1.5. Verbindung zur Email Datenbank herstellen

Um Emails zu verschicken muss sich der Benutzer einmalig mit der Emaildatenbank verbinden. Diese liegt auf dem Nexus Server. Mit dem Befehl Strg–O kann eine beliebige Datenbank geöffnet werden. Über das Pull–Down Menü wählt man zunächst den Server **nexus/HSB** aus. Die Datenbank liegt im Verzeichnis Mail (Abb. 6.16).

Abbildung 6.16.: Lotus Notes: Datenbank öffnen

Die Liste aller bekannten User wird angezeigt (Abb. 6.17) der Benutzer wählt seinen Namen aus, bestätigt die Auswahl mit „Öffnen“.

Abbildung 6.17.: Lotus Notes: Auswahl

Die Datenbank wird geöffnet und der Benutzer kann Emails schreiben und Empfangen.

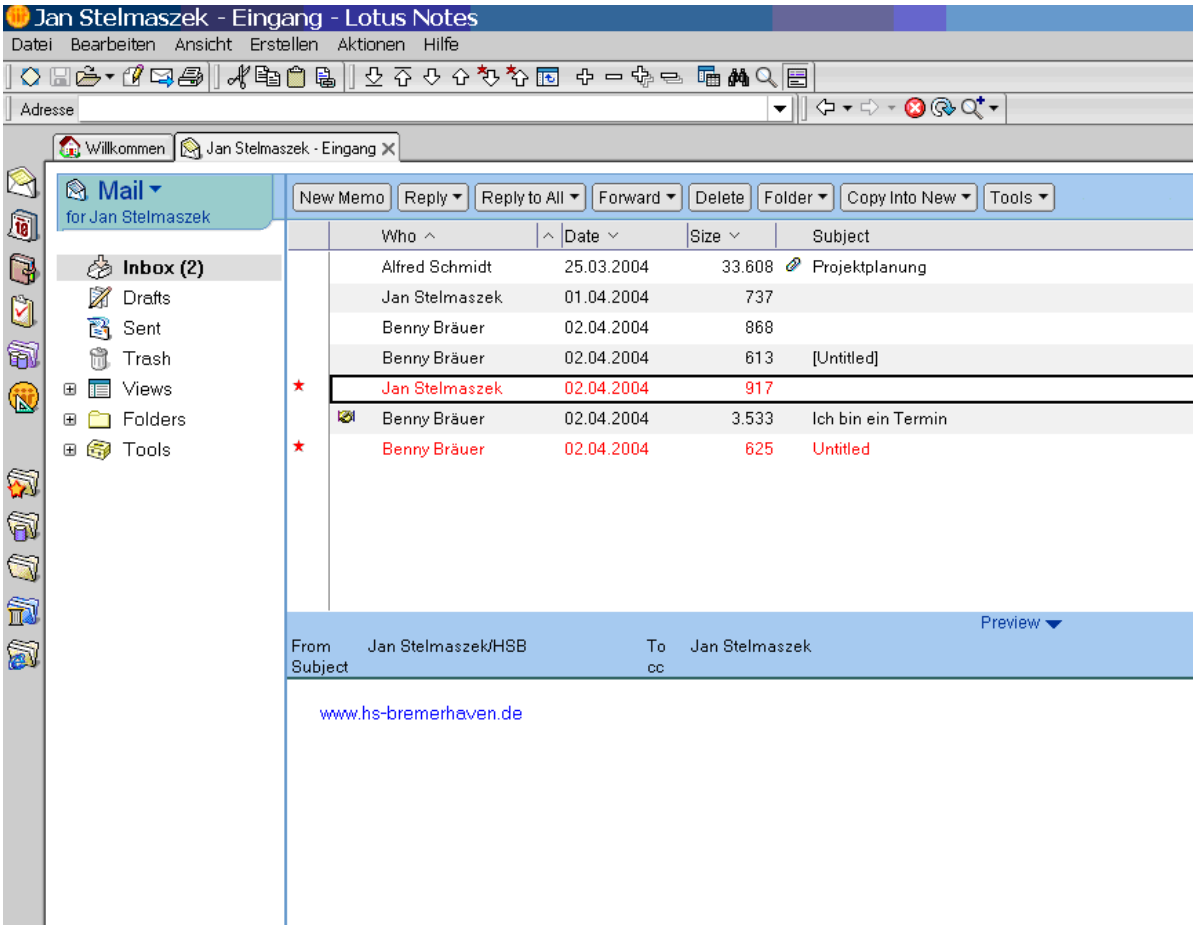


Abbildung 6.18.: Lotus Notes: Email Eingang

6.2. Teamroom

6.2.1. Verbinden mit der Datenbank

Per **Strg** + **O** wird der Auswahldialog zur Auswahl der Datenbank aufgerufen und dort der Server **nexus/HSB** ausgewählt. Die benötigte Datenbank heißt **Teamroom_DB**.

Abbildung 6.19.: Lotus Notes: Datenbank öffnen

Es wird empfohlen, sich per entsprechenden Knopf ein Lesezeichen an die Seitenleiste bzw. auf den Arbeitsbereich zu setzen, was die zukünftige Navigation erleichtert.

6.2.2. Oberfläche

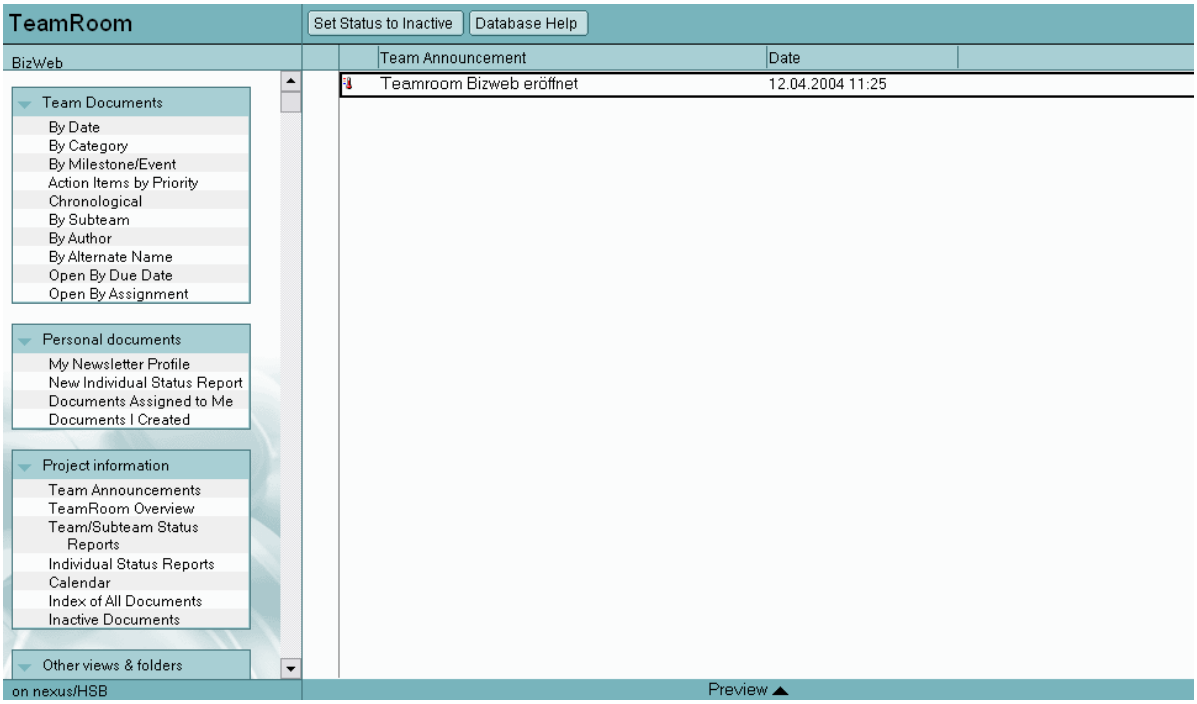


Abbildung 6.20.: Lotus Notes: Oberfläche

Nach der Einwahl in die Datenbank gelangt man in den Teamroom. Als erstes Fenster erscheint hier „Team Announcement“. Wichtige Bekanntgaben seitens der Projektleitung kann man künftig hier nachlesen.

6.2.3. Dokument anlegen

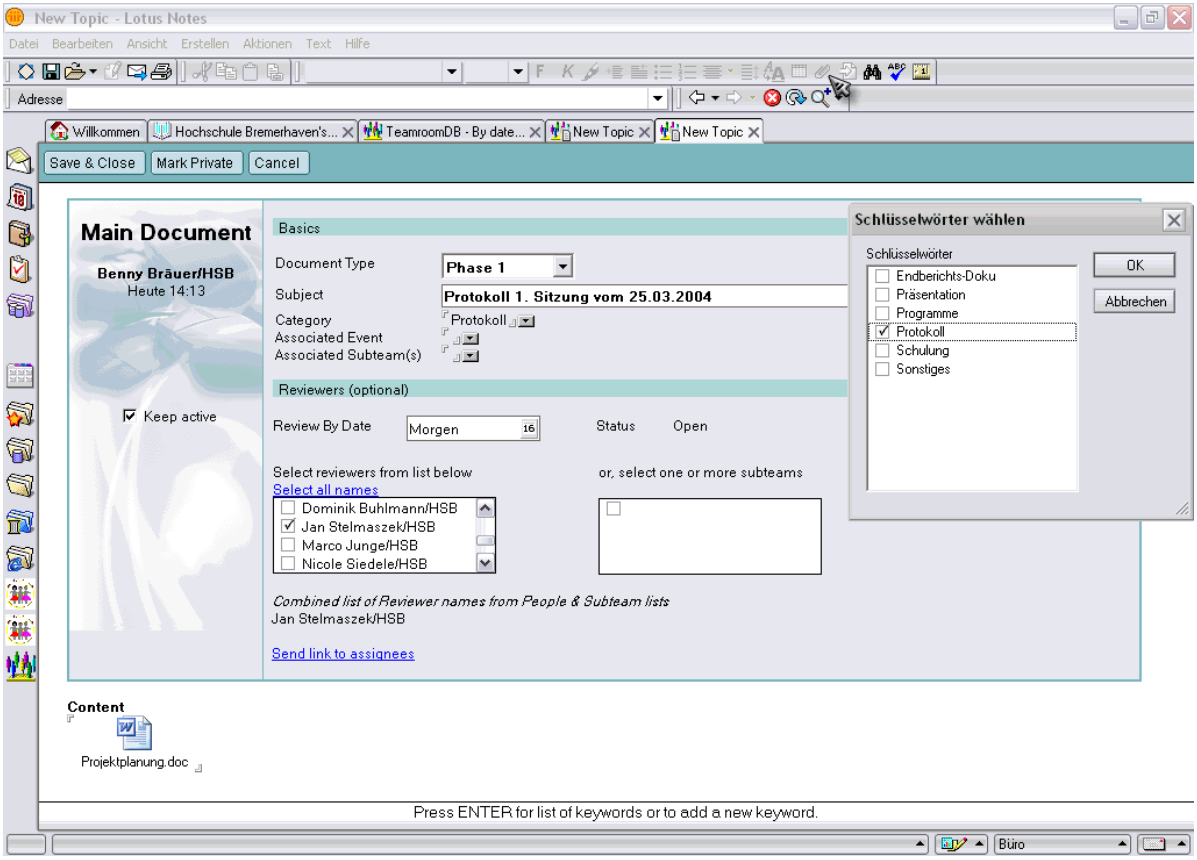


Abbildung 6.21.: Lotus Notes: Dokument anlegen

Dokumente anlegen kann man unter dem Menüpunkt „Team Documents“ – beliebiges Untermenü – Button „New Document“. Man gelangt in das unter Abb. 6.21 gezeigte Fenster. Hier wählt man einen Dokumententyp und die Kategorie (Popup Rechts) aus und gibt dem Dokument einen aussagekräftigen Namen bzw. einen Betreff. Unter „Content“ wird dann der Text eingegeben bzw. die Datei (einfügen mittels Büroklammer, s. Mauszeiger) angehängt.

Optional kann man noch ein Ablaufdatum angeben, nach dessen Überschreiten das Dokument als inaktiv markiert wird (s. „Keep Active“ links). Ferner ist es möglich, sog. „Reviewers“ zu bestimmen, die das Dokument lesen (evt. auch gegenzeichnen/bestätigen) müssen. Per Klick auf „Send link to assignees“ erreicht alle markierten Personen eine Mailbenachrichtigung, dass ein neues Dokument vorliegt. Zum Abschluss muss mittels „Save & Close“ das Dokument in der Datenbank gespeichert werden.

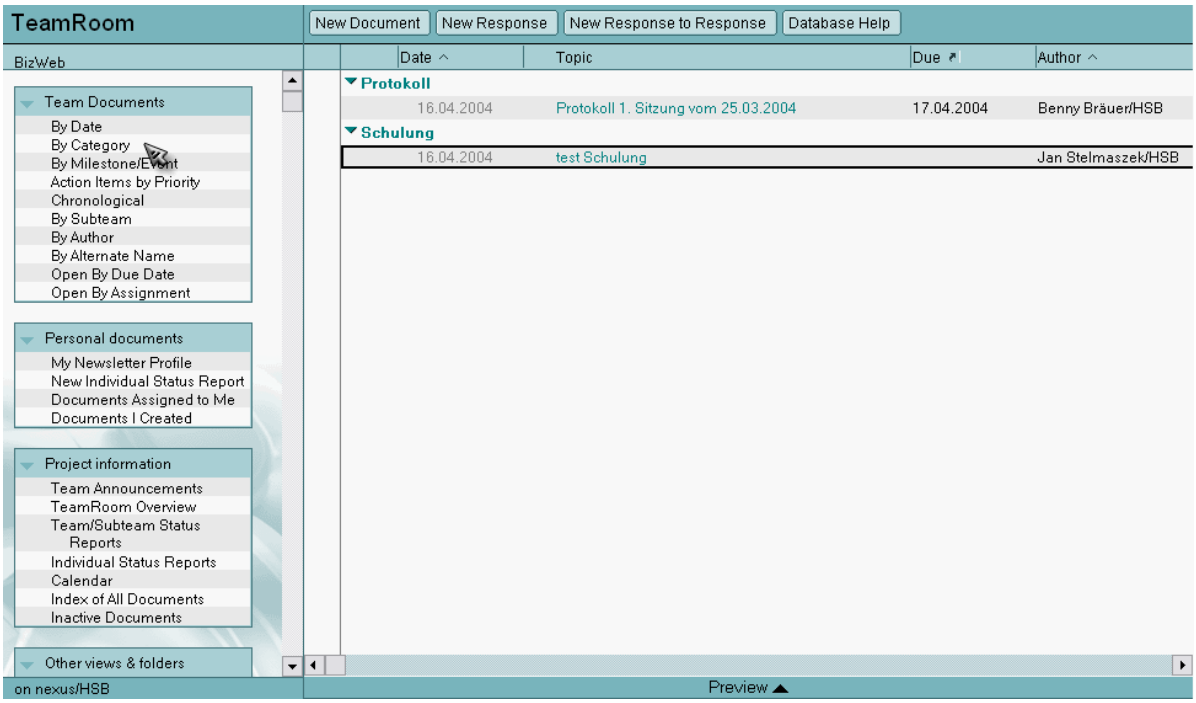


Abbildung 6.22.: Lotus Notes: Dokumentenübersicht

Hat man ein Dokument angelegt, kann man es sich in der Übersicht anschauen. Es stehen mehrere Übersichtsmöglichkeiten zur Auswahl. Die gebräuchlichsten sind dabei „By Date“, „By Category“ und „By Author“. In Abbildung 6.22 sieht man die Sortierung nach der Kategorie.

6.2.4. Persönliche Einstellungen

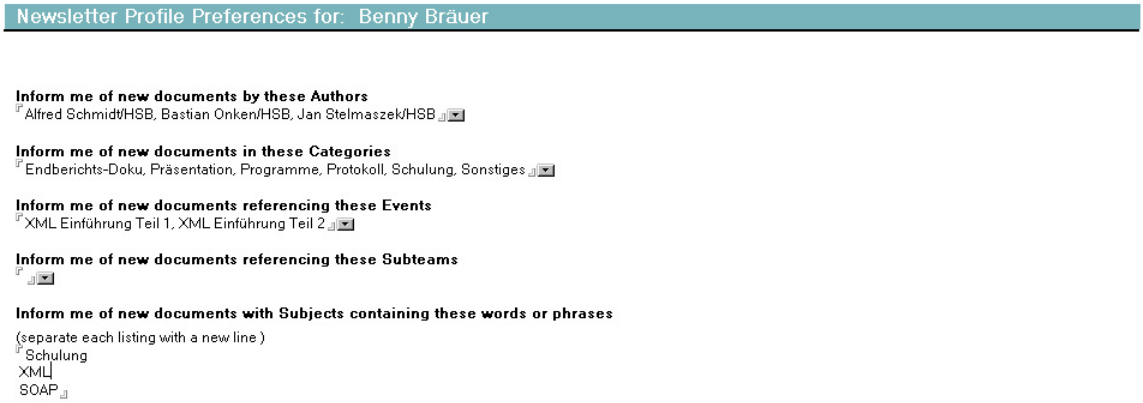


Abbildung 6.23.: Lotus Notes: Persönliche Einstellungen

In den „Newsletter Profile Preferences“ lässt sich einstellen, das man eine Nachricht zugesandt bekommt, wenn ein Dokument eingestellt wurde, welches den angegebenen Kriterien entspricht.

6.2.5. Kalender

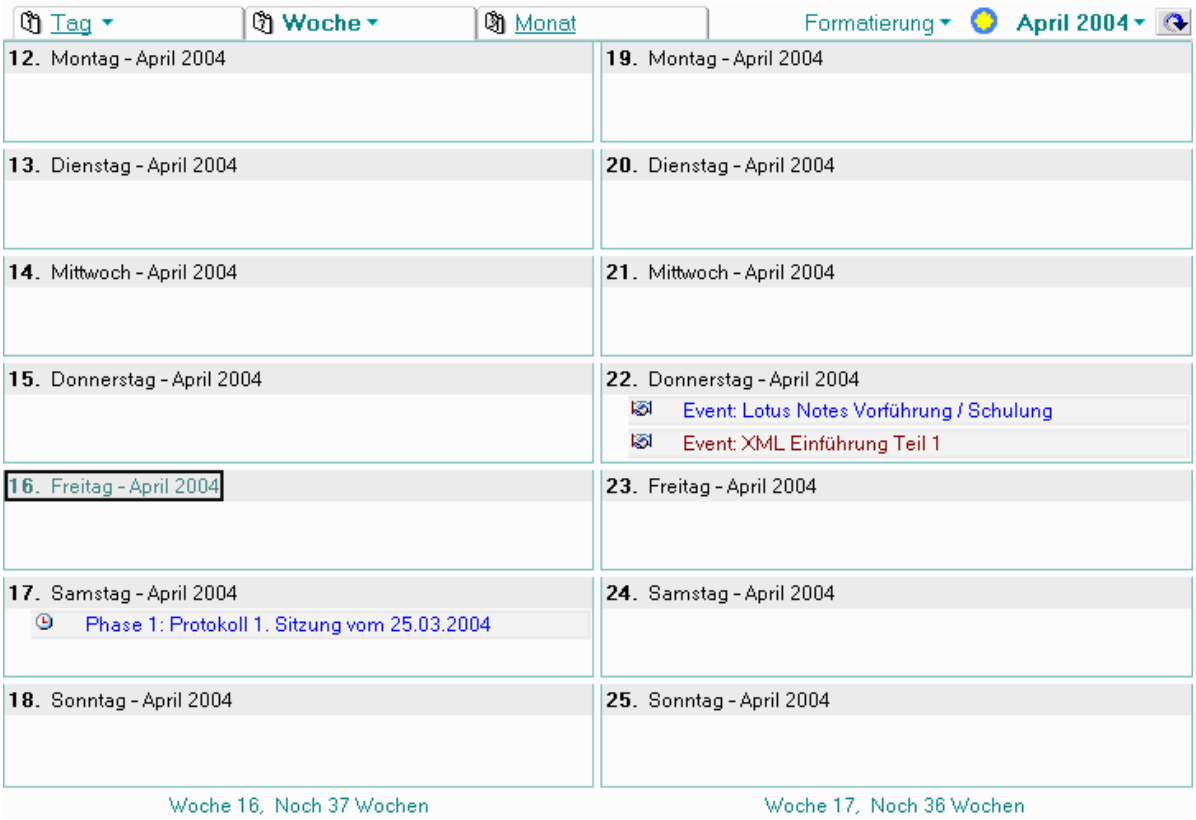


Abbildung 6.24.: Lotus Notes: Kalender

Der Kalender stellt eine weitere Übersicht für Dokumente und Ereignisse dar. Vorher definierte Meilensteine werden hier für alle Projektmitglieder sichtbar. Neue Einträge können wie gehabt über „New Document“ und „New Event“ eingetragen werden.

7. XML

7.1. Grundlagen

7.1.1. Geschichte

- Erste Anfänge in 1960er Jahren (GML[1])
- Seit 1978 Standardisierungsausschuss des ANSI, 1980 erster Entwurf für (SGML[2]), 1986 ISO–Standard
- Seit 1996 Extensible Markup Language (XML[3]), vor allem Reduktion der Komplexität

7.1.2. Allgemeines

XML basiert auf SGML und dient zum Beschreiben der (Daten–) Strukturen von Dokumenten. Bei XML werden Daten, Struktur und Format von Dokumenten separat behandelt. Das Regelwerk wird dabei in der Dokument Type Definition (DTD) festgelegt. Weiterhin kann XML (mit XSLT) in andere Dokumentformate umgewandelt werden (z.B. HTML, RTF, Text, ...).

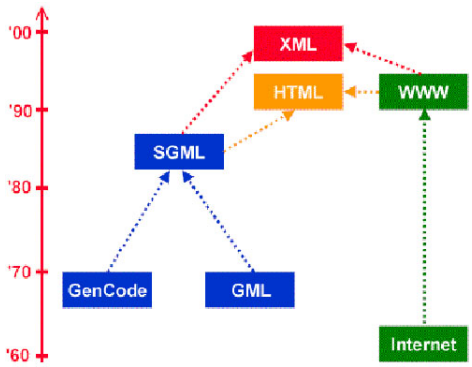


Abbildung 7.1.: XML Grundlagen: XML–Stammbaum

[1]GML – Generalized Markup Language
[2]SGML – Standard Generalized Markup Language
[3]XML – Extensible Markup Language

7.1.3. Entwurfsziele von XML

- XML soll sich im Internet auf einfache Weise nutzen lassen.
- XML soll ein breites Spektrum von Anwendungen unterstützen.
- XML soll zu SGML kompatibel sein.
- Es soll einfach sein, Programme zu schreiben, die XML–Dokumente verarbeiten.
- Die Zahl optionaler Merkmale in XML soll minimal sein, idealerweise Null.
- XML–Dokumente sollen für Menschen lesbar und angemessen verständlich sein.
- Der XML–Entwurf soll zügig abgefasst sein.
- Der Entwurf von XML soll formal und präzise sein.
- XML–Dokumente sollen leicht zu erstellen sein.
- Knappheit von XML–Markup ist von minimaler Bedeutung.

7.1.4. Dokumentenaufbau

XML verwendet das bereits von HTML her bekannte Auszeichnungskonzept unter Verwendung von Elementen, Attributen und Entities. Das eigentliche Markup bilden die Tags:

```
1 <?xml version="1.0"?>
2 <wurzelelement>
3    <unterelement>Inhalt</unterelement>
4 </wurzelelement>
```

Quelltext 7.1: XML Grundlagen: Grundschema einer XML–Datei

Tags

Mit Ausnahme des einleitenden XML–Tags, muss es zu jedem Tag ein Abschluss–Tag geben, dabei muss die Groß– und Kleinschreibung beachtet werden. Die Definition von Tags in XML sollte präzise gehandhabt werden.

Elemente

Elemente sind eindeutig identifizierbare Einheiten („Name“). Sie enthalten Text oder weitere, hierarchisch organisierte Sub–Elemente (Baum–Struktur), dabei haben sie immer ein Start–Tag und End–Tag (außer bei leeren Elementen), dazwischen steht der jeweilige Inhalt.

```
1 <titel>Einführung in XML</titel>
```

Quelltext 7.2: XML Grundlagen: Beispiel für ein Element

```
1 <titel/>
```

Quelltext 7.3: XML Grundlagen: Beispiel für ein leeres Element

Attribute

Elemente können ein oder mehrere Attribute enthalten. Attribute haben einen Namen und einen Wert (`sprache="de"`) und sie stehen im Start–Tag.

```
1 <titel sprache="de" typ="Untertitel"> ... </titel>
```

Quelltext 7.4: XML Grundlagen: Beispiel für Attribute

Aufbau eines Elementes

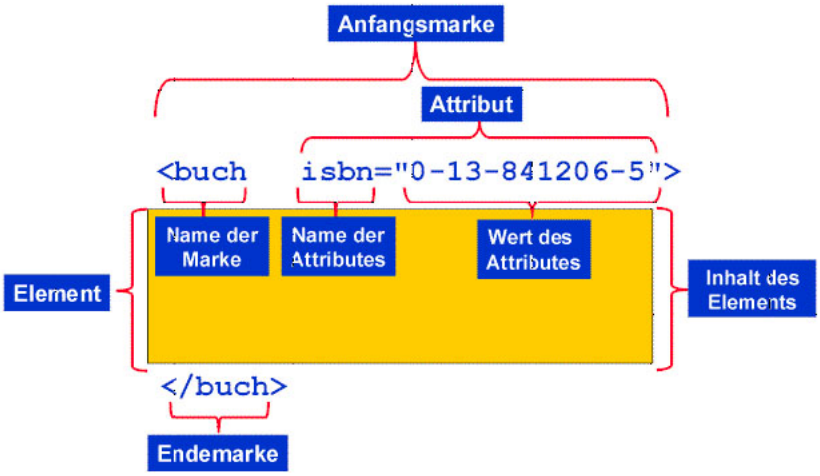


Abbildung 7.2.: XML Grundlagen: Aufbau eines Elementes

Entities

Entities definieren Zeichenkombinationen, welche im XML–Dokument durch bestimmte andere (auch externe) Inhalte wie z.B. Zeichendaten ersetzt werden (vergleichbar mit Makros). Entity–Referenzen beginnen mit dem `&`-Zeichen („Ampersand“) und enden mit einem Semikolon (`&xyz;`). Es gibt allgemeine–, externe–, Parameter– und ungeparste Entities.

Einfache Beispiel–Entity:

```
<!ENTITY KTD "Kommunikation und Technische Dokumentation">
```

Quelltext 7.5: XML Grundlagen: Definition

```
&KTD;
```

Quelltext 7.6: XML Grundlagen: Anwendung

Kommentare

```
<!-- Das ist ein Kommentar -->
<!-- Das ist
ein mehrzeiliger
Kommentar -->
```

Quelltext 7.7: XML Grundlagen: Kommentare

Namespaces

Namespaces dienen zur Vermeidung von Kollisionen bei gleichnamigen Bezeichnern. Mit dem Attribut `xmlns` kann auf Namensräume hingewiesen werden

```
<element xmlns:ad="http://www.mein.server.de/adressen"
      xmlns:bu="http://www.mein.server.de/buecher">
   <ad:vorname>...</ad:vorname>
   <ad:nachname>...</ad:nachname>
   <bu:titel>...</bu:titel>
   <!-- Weiterer Inhalt von element -->
</element>
```

Quelltext 7.8: XML Grundlagen: Beispiel für Namespaces

„ad“ bzw. „bu“ kennzeichnen zwei Namensräume, für Elemente, die sich nicht auf die aktuelle DTD beziehen. Die Elemente „vorname“ und „nachname“ sowie „titel“ stammen aus verschiedenen Namensräumen, sind aber durch deren Angabe wiederum eindeutig anwendbar. Die Adresse zum Namensraum muss nicht unbedingt physisch existieren, sie ist eine reine Konvention

Wohlgeformte Dokumente

- Am Anfang des Dokuments steht die XML–Version (`<?xml version="1.0"?>`)
- Optional kann der verwendete Zeichensatz noch mit angegeben werden

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

- Jedes Element hat genau ein Root–Element, alle anderen Elemente sind dem Root–Element untergeordnet

- Elementnamen müssen mit einem Buchstaben beginnen und dürfen aus weiteren Buchstaben, Zahlen, „_“, „:“, „-“ und „.“ bestehen
- Die Elementinhalte dürfen folgende Zeichen nicht beinhalten: „<“, „>“ und „;“
- Jedes Child–Element muss innerhalb seines Parent–Element stehen
- Es gibt leere Elemente (`<element/>`)
- Attribute müssen in Anführungszeichen stehen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<zeitungsartikel>
   <eintrag nr="1">
      <titel>Schulung</titel>
      <info>Am heutigen Donnerstag werden wir, beim Versuch die
         anderen Projektteilnehmer mit der Handhabung dieser
         Sprache vertraut zu machen, alles geben!</info>
   </eintrag>
   <eintrag nr="2">
      <titel>Kutterpullen</titel>
      <info>Dieses Jahr sind die berüchtigten Bleienten der
         Top-Favorit. Da gibt's nur eins: Den klaren Sieg!!!</info>
   </eintrag>
</zeitungsartikel>
```

Quelltext 7.9: XML Grundlagen: Beispiel für ein XML–Dokument

Die Ausgabe der im Beispiel beschriebenen XML–Datei im Browser sieht dann wie folgt aus:

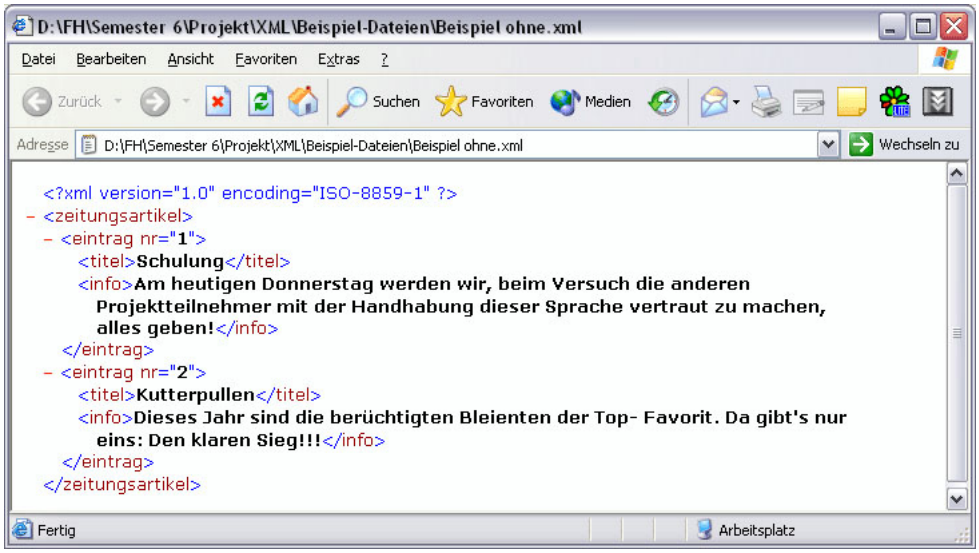


Abbildung 7.3.: XML Grundlagen: Browserausgabe einer einfachen XML–Datei

7.1.5. Dokument Type Definition (DTD)

Eine DTD enthält das Vokabular, welches in einem XML–Dokument verwendet werden darf (= „Wörterbuch“ oder „Rechtschreibung“). Benannt werden die Namen der Elemente, Attribute, Entities und Notations. Weiterhin werden Aussagen zu Anzahl und Inhalt gemacht sowie Regeln zur Verschachtelung der Elemente festgelegt (= „Grammatik“). Damit ist die DTD–Datei der Schlüssel zu einem XML–Dokument. Es gibt interne und externe DTDs, diese werden in XML–Dokumenten über die DOCTYPE–Deklaration bekannt gemacht. Ein Dokument, dem eine DTD zugeordnet ist, wird als gültig (`valid`) bezeichnet, wenn in dem Dokument gegen keine der in der DTD definierten Regeln verstoßen wird (ohne DTD ist es „well–formed“). Die DTD ist für das Parsen wichtig und sie wird im Browser nicht angezeigt.

```
<!ELEMENT zeitungsartikel (eintrag+)>
<!ELEMENT eintrag         (titel,info)>
<!ELEMENT titel           (#PCDATA)>
<!ELEMENT info            (#PCDATA)>
<!ATTLIST eintrag nr      CDATA #REQUIRED>
```

Quelltext 7.10: XML Grundlagen: Beispiel für eine externe DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE zeitungsartikel SYSTEM "dtd_beispiel1.dtd">
<zeitungsartikel>
   <eintrag nr="1">
      <titel>Schulung</titel>
      <info>Am heutigen Donnerstag werden wir, beim Versuch die
         anderen Projektteilnehmer mit der Handhabung dieser
         Sprache vertraut zu machen, alles geben!</info>
   </eintrag>
   <eintrag nr="2">
      <titel>Kutterpullen</titel>
      <info>Dieses Jahr sind die berüchtigten Bleienten der
         Top-Favorit. Da gibt's nur eins: Den klaren Sieg!!!</info>
   </eintrag>
</zeitungsartikel>
```

Quelltext 7.11: XML Grundlagen: Beispiel für die dazu gehörige XML–Datei

Die Ausgabe der XML–Datei mit DTD sieht dann folgendermaßen aus:

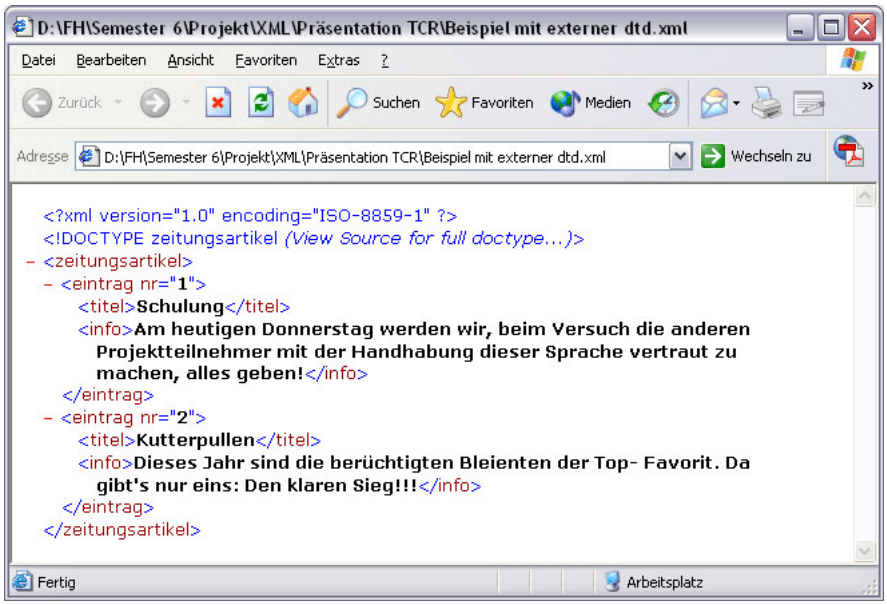


Abbildung 7.4.: XML Grundlagen: Browserausgabe einer XML–Datei mit DTD

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE zeitungsartikel [
3     <!ELEMENT zeitungsartikel (eintrag+)>
4     <!ELEMENT eintrag         (titel,info)>
5     <!ELEMENT titel           (#PCDATA)>
6     <!ELEMENT info            (#PCDATA)>
7     <!ATTLIST eintrag nr      CDATA #REQUIRED>
8  ]>
9  <zeitungsartikel>
10    <eintrag nr="1">
11       ...
12    </eintrag>
13    ...
14 </zeitungsartikel>
```

Quelltext 7.12: XML Grundlagen: Beispiel für eine interne DTD (DTD innerhalb der XML-Datei)

Die Darstellung im Browser ist bei der internen und externen DTD identisch.

7.1.6. Schemata (XSD)

Schemata haben eine ähnliche Funktion wie DTDs, sie bieten jedoch mehr Möglichkeiten als diese, so können z.B. Datentypen festgelegt werden, Typen, Werte und die Anzahl von Elementen beschränkt werden, etc. Die Syntax ist dabei die gleiche wie im XML–Dokument (DTDs haben eine eigene).

Ein Schema liegt immer als externe Datei vor und ist selbst ein gültiges XML–Dokument, welches üblicherweise mit „`.xsd`“ endet. Die Angaben über die Schema–Datei stehen im Root–Element der XML–Datei.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
   <xs:element name="Buch"/>
   <xs:element name="Gruppe" type="xs:string"/>
   <xs:element name="Nummer" type="xs:byte"/>
   <xs:element name="Preis" type="xs:decimal"/>
   <xs:element name="Titel" type="xs:string"/>
</xs:schema>
```

Quelltext 7.13: XML Grundlagen: Beispiel für eine XSD–Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Buch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="xsd_beispiel1.xsd">
   <Nummer>113</Nummer>
   <Titel>E.T.A. Hoffmann</Titel>
   <Gruppe>Biographie</Gruppe>
   <Preis>8.80</Preis>
</Buch>
```

Quelltext 7.14: XML Grundlagen: Zum Beispiel gehörende XML–Datei

7.1.7. Cascading Style Sheets (CSS)

Stylesheets dienen zur einfachen, clientseitigen (im Browser) Transformation von XML–Dateien und sind für das Aussehen, nicht für die Struktur von Dokumenten zuständig. Mehrere Stylesheets lassen sich kombinieren. Die Zuweisung der externen Stylesheets erfolgt im Kopf eines XML–Dokumentes.

Ein CSS besteht aus 2 Teilen, erst kommt der Selektor (das XML–Element, für welches das Format festgelegt werden soll), dann die Deklaration, welche in geschweifte Klammern gefasst wird. Innerhalb der Klammern kann dann die Textformatierung der einzelnen Elemente vorgenommen werden (wie vom HTML her bekannt). Formatiert werden können unter anderem die Schrift, die Farben, die Textgestaltung, die Seitenränder, Links, Bilder, die Ränder, etc.

Stylesheets haben jedoch einige Beschränkungen, so dienen sie eigentlich nur für eine einfache Wiedergabe, es können keine Änderungen in der Reihenfolge der Elemente vorgenommen werden, die Struktur kann nicht geändert werden, es kann kein Text ersetzt werden und es können keine Strukturauswertungen vorgenommen werden.

```
eintrag{
   display: block;
   margin-bottom: 20px;
}
titel{
   color: #FF0000;
   background-color: #FFFFFF;
   font-family: Arial, Helvetica, Sans-Serif;
   font-size: 120%;
   display: block;
   margin-bottom: 10px;
}
```

```
13 info{
14    color: #00CCCC;
15    background-color: #FFFFFF;
16    font-family: Arial, Helvetica, sans-serif;
17    font-size: 100%;
18 }
```

Quelltext 7.15: XML Grundlagen: Beispiel für eine CSS–Datei (Name: css_beispiel1.css)

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <?xml-stylesheet version="1.0"
3     href="css_beispiel1.css" type="text/css"?>
4  <zeitungsartikel>
5     <eintrag nr="1">
6        <titel>Schulung</titel>
7        <info>Am heutigen Donnerstag werden wir, beim Versuch die
8           anderen Projektteilnehmer mit der Handhabung dieser
9           Sprache vertraut zu machen, alles geben!</info>
10    </eintrag>
11    <eintrag nr="2">
12       <titel>Kutterpullen</titel>
13       <info>Dieses Jahr sind die berüchtigten Bleienten der
14          Top-Favorit. Da gibt's nur eins: Den klaren Sieg!!!</info>
15    </eintrag>
16 </zeitungsartikel>
```

Quelltext 7.16: XML Grundlagen: Beispiel für die dazu gehörende XML–Datei

Hier sieht man die Browserausgabe der XML–Datei mit CSS:

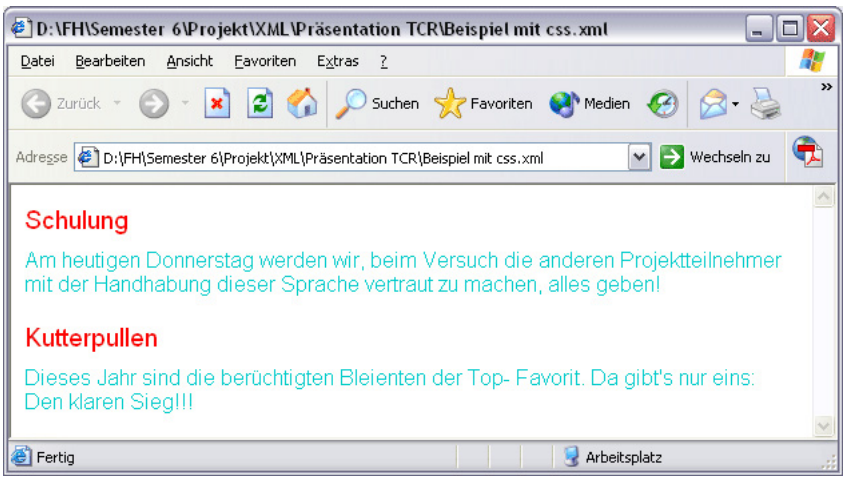


Abbildung 7.5.: XML Grundlagen: Browserausgabe mit CSS

Die Ausgabe ohne CSS–Datei ist in Abbildung 7.3 dargestellt

7.1.8. eXtensible Stylesheet Language (XSL)

XSL Dient zur Transformation, Formatierung und Darstellung eines XML–Dokuments. Dabei bietet es mehr Möglichkeiten und ist umfangreicher als CSS.

XSL besteht aus 3 Teilen:

- XSL Formating Objects (XSLFO)
- XSL Transforms (XSLT)
- XPath (wird in einem späteren Abschnitt genauer behandelt)

XSLFO

Die so genannten Formatting Objects dienen zur Formatierung von XML–Daten für die Druck– oder PDF–Ausgabe. Es bestehen Analogien zu den Cascading Style Sheets. Die praktische Realisierung erfordert spezielle Software („Formatter“) und erfolgt Server–seitig, also nicht im Browser.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master master-name="simple"
         page-height="29.7cm" page-width="21cm" margin-top="2.5cm"
         margin-bottom="2.5cm" margin-left="3cm" margin-right="2cm">
         ...
      </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-name="simple">
      ...
   </fo:page-sequence>
</fo:root>
```

Quelltext 7.17: XML Grundlagen: Beispielhafter Rahmenbau einer Vorlage

XSLT

XSLT ermöglicht die Transformation von XML–Dokumentinhalten in andere XML–Strukturen bzw. die Ausgabe als HTML oder als reiner Text. Dabei ist eine XSLT–Vorlage selbst ein wohlgeformtes XML–Dokument. Die Verarbeitung erfolgt mit speziellen Softwarekomponenten, so genannten XSLT–Prozessoren (z. B. MSXSL, Sablotron, SAXON oder XMLStarlet). In der XML–Datei kann ggf. (je nach Software) der Verweis auf die zugehörige Stylesheet–Datei angegeben werden.
Die Möglichkeiten im Bereich der Transformation sind sehr weitreichend. So lassen sich unter anderem Werte und Elemente übernehmen, es können Rekursionen, Schleifen, Sortierungen, If–Prüfungen und If–Else–Schleifen genutzt werden, etc.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"
         doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
         doctype-system="http://www.w3.org/TR/html401/loose.dtd"
         encoding="ISO-8859-1" version="4.01" indent="yes"/>
   <xsl:template match="/">
```

```
9       <html>
10         <head>
11            <title>XML-Datei mit XSL</title>
12         </head>
13         <body bgcolor="#FFFFFF" text="#00CC77">
14            <h1>Zeitungsartikel</h1>
15            <xsl:for-each select="zeitungsartikel/eintrag">
16            <h2>
17               <xsl:value-of select="titel" />
18            </h2>
19            <p>
20               <xsl:value-of select="info" />
21            </p>
22            </xsl:for-each>
23         </body>
24      </html>
25   </xsl:template>
26</xsl:stylesheet>
```

Quelltext 7.18: XML Grundlagen: Beispiel für eine XSL–Datei (Name: xsl_beispiel1.xsl)

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <?xml-stylesheet href="xsl_beispiel1.xsl" type="text/xsl"?>
3 <zeitungsartikel>
4    <eintrag nr="1">
5       <titel>Schulung</titel>
6       <info>Am heutigen Donnerstag werden wir, beim Versuch die
7          anderen Projektteilnehmer mit der Handhabung dieser
8          Sprache vertraut zu machen, alles geben!</info>
9    </eintrag>
10   <eintrag nr="2">
11      <titel>Kutterpullen</titel>
12      <info>Dieses Jahr sind die berüchtigten Bleienten der
13         Top-Favorit. Da gibt's nur eins: Den klaren Sieg!!!</info>
14   </eintrag>
15</zeitungsartikel>
```

Quelltext 7.19: XML Grundlagen: Beispiel für eine dazu gehörige XML–Datei

Hier sieht man die Browserausgabe der XML–Datei mit XSL–Datei:

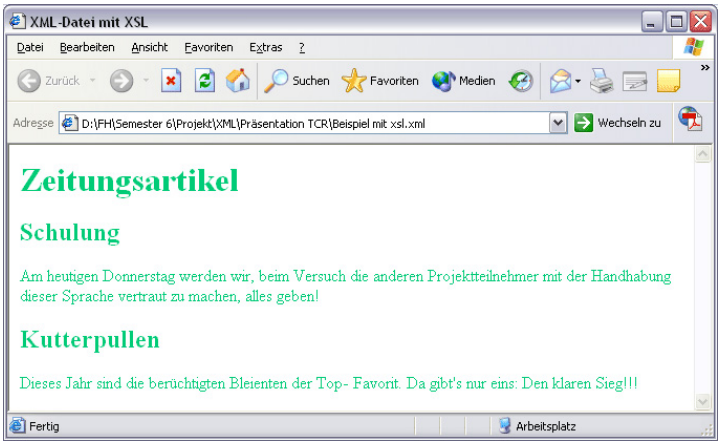


Abbildung 7.6.: XML Grundlagen: Browserausgabe mit XSL

Die Ausgabe ohne zugehörige XSL–Datei ist in Abbildung 7.3 dargestellt.

7.2. Parser

7.2.1. Einleitung

Nachdem wir im vorherigen Kapitel die Grundlagen kennen gelernt haben, stellt sich die Frage, wie wir mittels der Programmiersprache Java auf die Elemente und Attribute eines XML–Dokumentes zugreifen können. Dabei wollen wir einen XML–Datenstrom zur weiteren Verarbeitung einlesen und gleichzeitig das Dokument auf Wohlgeformtheit und wenn vorhanden auf Validität prüfen.

Dies ist mit einem so genannten Parser möglich. Ein Parser ist laut Definition ein Vorgang mit dem Parser, der eine Software ist, die den Datenstrom eines Dokumentes analysiert und entsprechend der Syntax aufbereitet. Beim Parsen werden die Informationen des Dokumentes in die Elemente gefiltert, in die die Informationen strukturiert sind. Bei SGML– und XML–konformen Auszeichnungssprachen, wie HTML oder XHTML, bedeutet dies, dass der Text gemäß den entsprechenden „Tags“ aufbereitet wird, bevor er dann durch die Layout–Engine angezeigt wird. Parsen wird u.a. auch durchgeführt bei der syntaktischen Prüfung, z.B. bei Validierung oder Compilierung.

Grundsätzlich gibt es hierfür zwei verschiedene Ansätze von Parsern, den so genannten SAX– und den DOM–Parser. Beide werden in den folgenden Kapiteln ausführlich behandelt.

7.2.2. Vorbereitungen

Arbeitsumgebung

Um die Arbeitsweise der verschiedenen Parser zu veranschaulichen, verwenden wir die Arbeitsumgebung „Eclipse“ in der Version 3.0. Sie wird in einem folgenden Kapitel ausführlich beschrieben.

Verwendete Bibliotheken

Die erforderlichen Klassen, die für die Implementierung der einzelnen Parser notwendig sind, sind in verschiedenen Bibliotheken zu finden, da einzelne Projekte immer wieder die gleichen Parser verwenden. Sie sind somit mehrfach in verschiedenen Bibliotheken zu finden. Wir haben uns auf die Verwendung der `xercesImpl.jar` und `xmlParserAPIs.jar` geeinigt, die beide im `org.apache.xerces`–Plugin zu finden sind. Zusätzlich wird zur Demonstration des JDOM–Parsers die `jdom.jar` benötigt, die unter www.jdom.org zum Download bereit steht.

Wie diese Bibliotheken in Eclipse eingebunden und ein neues Java–Projekt erstellt wird, soll nicht Gegenstand dieser Ausarbeitung sein.

7.2.3. Parser im Vergleich

SAX–Parser

Im Folgenden beschäftigen wir uns mit dem schon oben erwähnten SAX–Parser. SAX bedeutet „Simple API for XML“.

Ein SAX–Parser arbeitet Event–basiert, d.h. beim Einlesen eines Dokumentes arbeitet der Parser es sequenziell Element für Element ab und erzeugt bei bestimmten XML–Strukturen das dazu passende Ereignis. Wir benutzen diesen Parser in der Version, wie er im Apache–Xerces–Projekt implementiert ist (SAX 2.0) und unter der Adresse http://xml.apache.org/xerces2-j/ verfügbar ist. Die dazu gehörige Dokumentation kann unter http://xml.apache.org/xerces2-j/api.html eingesehen werden.

Funktionsweise von SAX SAX bietet einer Applikation ein einfaches und schlankes Programmiermodell, das folgende Funktionsweise hat.

Der SAX–Parser wird mit einem XML–Dokument initialisiert und gestartet. Der Datenstrom wird sequenziell von Anfang bis Ende durchlaufen. Im Laufe der Verarbeitung ruft der Parser registrierte Abonnenten mit einer definierten Schnittstelle zurück, wenn er auf die verschiedenen Bestandteile des XML–Dokuments stößt. Um dieses zu verdeutlichen betrachten wir einmal ein einfaches XML–Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE medienliste SYSTEM medienliste.dtd>
<medienliste type="short">
   <buch>
      <isbn>3-8274-1054-1</isbn>
      <titel>UML kompakt</titel>
      ...
   </buch>
</medienliste>
```

Quelltext 7.20: XML Parser: Einfaches XML–Dokument

Im Folgenden sehen wir, wie ein SAX–Parser auf das vorliegende XML–Dokument reagiert, d.h. welche Methodenaufrufe die einzelnen Zeilen des Dokumentes bewirken:

```
startDocument()
startElement( // <medienliste ...>
   String namespaceURI,
   String localName,
   String qualifiedName,
   Attributes attributes // <... type="short">
)
startElement( ... ) // <buch>
startElement( ... ) // <isbn>
characters( // 3-8274-1054-1
   char[] buf, int offset, int length
)
endElement( ... ) // </isbn>
...
endElement( ... ) // </buch>
endDocument()
```

Quelltext 7.21: XML Parser: SAX Methodenaufrufe

Als Kommentar ist hier zur Veranschaulichung immer die jeweilige Zeile des XML–Dokumentes hinter den Methoden zu finden. Diese und weitere Methoden werden auch als Callback–Methoden bezeichnet, die sich in den folgenden Interfaces wieder finden.

- Interface `org.xml.sax.ContentHandler`
- Interface `org.xml.sax.DTDHandler`
- Interface `org.xml.sax.ErrorHandler`
- Interface `org.xml.sax.EntityResolver`

Beispiel Um den SAX–Parser auch als Implementierung eines Programms kennen zu lernen betrachten wir folgenden Java–Code:

```
1  import org.apache.xerces.parsers.SAXParser;
2  import org.xml.sax.Attributes;
3  import org.xml.sax.helpers.DefaultHandler;
4  //import org.xml.sax.SAXParseException;
5  import org.xml.sax.SAXException;
6  import java.io.IOException;
7  //  A Simple SAX Application
8  //  Extends org.xml.sax.helpers.DefaultHandler
9
10 public class BasicSAX extends DefaultHandler {
11    // Constructor
12    public BasicSAX(String xmlFile) {
13       //  Create a Xerces DOM Parser
14       SAXParser parser = new SAXParser();
15       //  Set Content Handler
16       parser.setContentHandler(this);
17       //  Parse the Document
18       try {
19          parser.parse(xmlFile);
20       } catch (SAXException e) {
21          System.err.println(e);
22       } catch (IOException e) {
23          System.err.println(e);
24       }
25    }
26    //  Start Element Event Handler
27    public void startElement(String uri, String local, String qName,
28          Attributes atts) {
29       System.out.println(local);
30    }
31    // Main Method
32    public static void main(String[] args) {
33       BasicSAX basicSAX = new BasicSAX("weather.xml");
34    }
35 }
```

Quelltext 7.22: XML Parser: BasicSAX.java

Die in der Main–Methode übergebene `weather.xml` hat dabei folgenden Aufbau:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE WEATHER SYSTEM "Weather.dtd">
3 <WEATHER>
4    <CITY NAME="New York">
5       <TEMP>64</TEMP>
6    </CITY>
7 </WEATHER>
```

Quelltext 7.23: XML Parser: weather.xml

Führen wir das Programm aus, so werden uns auf der Konsole sämtliche Elemente, die in dem Dokument `weather.xml` vorhanden sind, ausgegeben:

```
WEATHER
CITY
TEMP
```

Vor– und Nachteile Vorteil eines SAX–Parsers ist es, dass die Dateigröße des Dokumentes egal ist, da sich das XML Dokument nie vollständig im Speicher befindet. Somit ist das Bearbeiten von XML Daten auch dann möglich, wenn das XML nicht vollständig in den Speicher passt. Ein SAX–Parser ist, wie demonstriert, einfach zu benutzen und schnell. Nachteile ergeben sich dadurch, dass kein freier Zugriff auf beliebige Teile eines Dokumentes ohne Zwischenspeicherung besteht, dass das Dokument immer von oben nach unten durchwandert werden muss. Außerdem gibt es keine Funktionen zum Schreiben oder Ändern von XML–Daten.

7.2.4. DOM–Parser

Die zweite Möglichkeit XML–Dokumente zu verarbeiten bzw. zu parsen ist der so genannte DOM[4]–Parser. Mittels DOM–Parser können XML–Dokumente eingelesen und der Applikation als so genannte DOM–Bäume zur navigierenden Verarbeitung mit Lese– und Schreiboperationen zur Verfügung gestellt werden. Diese DOM–Bäume können wiederum durch die Applikationen erzeugt und zurück in XML–Dokumente überführt werden. Zusätzlich können DOM–Bäume in neue DOM–Bäume transformiert werden. Sie können außerdem über komfortable Event–Mechanismen mit ereignisbehandelnden Softwarekomponenten verknüpft werden. DOM ist ein durch das W3C entwickelter Standard. Wir benutzen hierbei, wie auch beim SAX–Parser, die Implementierung des Apache–Xerces–Projektes (DOM Level 2).

Funktionsweise von DOM Wie der Name „Document Object Model“ ahnen lässt, erlaubt es die DOM–Schnittstelle, gezielt auf einzelne Teile des Dokuments zuzugreifen.

[4]DOM – Document Object Model

Die Applikation instanziiert den Parser und stößt den Vorgang an. Im Gegensatz zu SAX, wo die Applikation schon während des Parsings mit Informationen versorgt wird, bekommt eine DOM–Applikation den gesamten XML–Baum erst dann übergeben, wenn das Dokument vollständig verarbeitet ist. Die Applikation greift dann über die DOM–Schnittstelle auf den nun in Objekten gespeicherten XML–Baum zu. Um die Funktionsweise von DOM zu veranschaulichen betrachten wir folgendes XML–Dokument:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE awql SYSTEM "http:// ... "
<awql>
   <restriction>
      <and>
         <equal>
            <attr name="type"/>
            <object>restaurant</object>
         </equal>
         <equal>
            <attr name="nationality"/>
            <object>italian</object>
         </equal>
      </and>
   </restriction>
   ...
</awql>
```

Quelltext 7.24: XML Parser: XML–Beispiel für DOM

Dieses Dokument sieht ein DOM–Parser als Baumstruktur und kann in diesem Baum navigieren und beliebige Elemente herausgreifen. Folgende Abbildung zeigt das XML–Dokument als Baum.

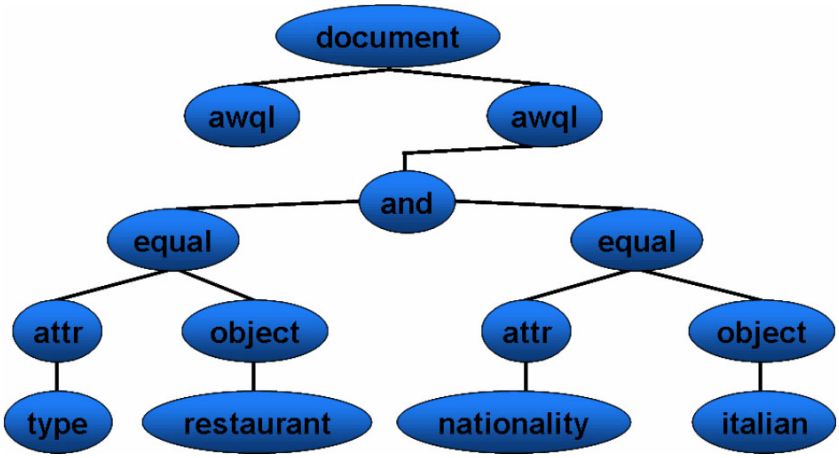


Abbildung 7.7.: XML Parser: XML-Baumstruktur

Zum Herausgreifen der einzelnen Elemente stehen in der Bibliothek u.a. folgende wichtige Methoden zur Verfügung:

- `getDocumentElement()`
- `getElementsByTagName(String)`

- `getChildNodes()`
- `getAttributes()`
- `getNodeName()`
- `getNodeType()`
- `getNodeValue()`
- `getFirstChild()`
- `getLastChild()`
- `getNextSibling`
- `getPreviousSibling`
- `getParentNode`

Beispiel Das folgende Beispiel liest das XML–Dokument `weather.xml` ein und gibt die enthaltenden Elemente auf der Konsole aus, wie es bereits beim SAX–Parser zu sehen war.

```
1  import org.apache.xerces.parsers.DOMParser;
2  import org.w3c.dom.Document;
3  import org.w3c.dom.Node;
4  //import org.w3c.dom.Element;
5  import org.w3c.dom.NodeList;
6  import org.xml.sax.SAXException;
7  import java.io.IOException;
8  //  A Simple DOM Application
9
10 public class BasicDOM {
11    // Constructor
12    public BasicDOM(String xmlFile) {
13       //  Create a Xerces DOM Parser
14       DOMParser parser = new DOMParser();
15       //  Parse the Document
16       //  and traverse the DOM
17       try {
18          parser.parse(xmlFile);
19          Document document = parser.getDocument();
20          traverse(document);
21       } catch (SAXException e) {
22          System.err.println(e);
23       } catch (IOException e) {
24          System.err.println(e);
25       }
26    }
27    //  Traverse DOM Tree. Print out Element Names
28    private void traverse(Node node) {
29       int type = node.getNodeType();
30       if (type == Node.ELEMENT_NODE)
31          System.out.println(node.getNodeName());
32       NodeList children = node.getChildNodes();
33       if (children != null) {
34          for (int i = 0; i < children.getLength(); i++)
```

```
35                traverse(children.item(i));
36            }
37        }
38        // Main Method
39        public static void main(String[] args) {
40            BasicDOM basicDOM = new BasicDOM("weather.xml");
41        }
42    }
```

Quelltext 7.25: XML Parser: BasicDOM.java

Vor– und Nachteile Vorteile von DOM–Parsern sind es, dass im Vergleich zum SAX–Parser zusätzlich zum Lesen von Elementen eines XML–Dokumentes auch das Erzeugen, Aktualisieren oder Löschen möglich ist. Der Nachteil von DOM besteht allerdings darin, dass bei der Verarbeitung von XML–Dokumenten eine komplexe Baumstruktur entstehen kann, die im Speicher gehalten werden muss, was aber den großen Vorteil hat, dass ständig auf alle Dokumententeile zugegriffen werden kann.

7.2.5. JDOM

Was ist JDOM?

JDOM steht für „Java Document Object Model“. JDOM ist ein API, um XML–Dokumente zu erstellen bzw. zu bearbeiten. Bei JDOM handelt es sich nicht (wie der Name vermuten lässt) um eine Weiterentwicklung oder Erweiterung von DOM. Vielmehr ist JDOM ein speziell für Java optimiertes eigenes API. JDOM ist verfügbar über eine Open–Source–Lizenz im Apache–Style. Das bedeutet, daß die Quellen von JDOM veröffentlicht sind und Entwickler JDOM einsetzen können, ohne gezwungen zu sein, die damit entwickelten Produkte als Open–Source zu vertreiben.

Mit SAX und DOM existieren bereits Standards, um XML–Dokumente zu bearbeiten bzw. entsprechende Parser zu erhalten. Sie weisen aber einige Nachteile auf. Sie sind abstrakt gehalten und bestehen, da sie einen sprachübergreifenden Ansatz verfolgen, hauptsächlich aus Schnittstellen. Außerdem werden relativ lange Revisionszeiten zwischen den einzelnen Versionen dieser APIs beklagt, so daß sie gegenüber den aktuellsten XML–Entwicklungen nicht immer up–to–date sind. Darüber hinaus ist mit SAX weder der wahlfreie Zugriff auf XML–Dokumente möglich, noch deren Bearbeitung. Java-Programmierern ist der Event–Ansatz von SAX eher fremd. Auch DOM wirkt auf Java–Entwickler nicht intuitiv, da typische Features wie z.B. Methodenüberladung fehlen. Zudem ist DOM anspruchsvoll was Prozessorleistung und Speicher betrifft, so daß es gerade für leichtgewichtige (Web–)Anwendungen weniger attraktiv ist.

JDOM ist (fast) immer dort sinnvoll einsetzbar, wo Java Code mit XML arbeitet. Von JDOM wird behauptet, die „80/20-Regel“ zu erfüllen: In 80 Prozent der Fälle von Java/XML-Problemen bewältigt es diese mit 20 Prozent des herkömmlichen Aufwands.

Funktionsweise von JDOM

Im Folgenden soll die Funktionsweise eines JDOM–Parsers erläutert werden. JDOM besteht aus den vier Paketen:

- `org.jdom`
- `org.jdom.adapters`
- `org.jdom.input`
- `org.jdom.output`

Diese Pakete können wie gehabt über die `import`–Anweisung eingebunden werden. Soll ein JDOM Document aus bereits vorhandenen XML–Daten generiert werden, kommen die Builder–Klassen

- `org.jdom.input.SAXBuilder` und/oder
- `org.jdom.input.DOMBuilder`

zum. Diese Klassen erlauben es, SAX– bzw. DOM–basierte Parser einzusetzen und optional eine Gültigkeitsprüfung durchzuführen. Da nicht alle DOM–Implementierungen das gleiche API benutzen, arbeitet DOMBuilder mit speziellen Adapter–Klassen. Für alle gängigen DOM–Parser gibt es die erforderlichen Adapterklassen. Für den Output einer Instanz vom Typ Document stehen die drei Output–Tools

- `org.jdom.output.XMLOutputter`
- `org.jdom.output.SAXOutputter`
- `org.jdom.output.DOMOutputter`

zur Verfügung. XMLOutputter schreibt das Dokument als XML in einen zu spezifizierenden Output–Stream (siehe Beispiel weiter unten). Alternativ generiert SAXOutputter dem JDOM–Dokument entsprechende SAX–Events, die man an Anwendungskomponenten senden kann, die solche Events erwarten. In ähnlicher Weise erzeugt DOMOutputter ein DOM–Dokument.

Beispiel

Das folgende Beispiel soll zeigen wie man JDOM ein Dokument mit Elementen erzeugen kann und dieses als XML–Dokument in einem Verzeichnis ablegen kann:

```java
import java.io.*;
import org.jdom.*;
import org.jdom.output.XMLOutputter;

public class JDOMsave {
   public static void main(String[] args) {
      generateXML();
   }
   public static void generateXML() {
      Document doc = new Document(new Element("gruppeninformation"));
      Element gruppennummer = new Element("gruppennummer");
      gruppennummer.addContent("1b");
      Element teilnehmer = new Element("teilnehmer");
      teilnehmer.setAttribute("vorname", "Max");
      teilnehmer.setAttribute("nachname", "Musterstudent");
      doc.getRootElement().addContent(gruppennummer);
      doc.getRootElement().addContent(teilnehmer);
      XMLOutputter outputter = new XMLOutputter();
      File file = new File("D:\\", "Gruppeninfo.xml");
      try {
         file.createNewFile();
         FileWriter out = new FileWriter(file);
         outputter.output(doc, out);
         out.close();
      } catch (IOException e) {
         System.out.println("IO-Fehler: " + e.toString());
      }
   }
}
```

Quelltext 7.26: XML Parser: JDOMsave.java

Das erzeugte XML–Dokument hat danach folgende Form:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<gruppeninformation>
   <gruppennummer>1b</gruppennummer>
   <teilnehmer vorname="Max" nachname="Musterstudent" />
</gruppeninformation>
```

Quelltext 7.27: XML Parser: XML–Beispiel für JDOM

7.3. X–PATH

XPath ist eine pfadorientierte Lokatorsprache und ermöglicht das Auffinden von Dokumentteilen (einzelnen Elementen, Attributen, etc.) durch Pfadausdrücke, die sich an der Struktur des XML–Dokuments orientieren. Als Rückgabe erfolgt immer eine Menge von Knoten (0, 1, N). XPath wird unter anderem durch XQuery, XPointer und XSLT genutzt. XPath betrachtet ein XML–Dokument als einen Baum, der aus verschiedenen Knoten besteht (vgl. DOM). Hierbei kennt XPath folgende Knotenarten:

- Root Node
- Element Node
- Text Node
- Attribute Node
- Namespace Node
- Processing Instruction Node
- Comment Node

Die XML–Deklaration und die DTD spielen keine Rolle.

Jeder der eben genannten Knoten besitzt einen Satz von Eigenschaften die belegt oder leer sein können, welche sind:

- Knotentyp (Root Node, Element Node, Text Node)
- Name (besteht aus Local Name und Namespace)
- Content (Zeichenkette bei Comment, Attribute und Processing Instruction, sonst rekursiv aufgeführt der gesamte Inhalt der Kindknoten)
- parent (Übergeordneter Knoten)
- child (Menge aller Kindknoten, wobei Element Nodes keine Kindknoten sind)

Jede XPath Konstruktion ist ein Ausdruck, aus entweder einer Ansammlung von Knoten, einem boolschen Wert, einer Zahl oder einer Zeichenfolge.

XPath Ausdrücke bestehen aus 3 Teilen, der Achse, dem Knotentest und optionalen Prädikaten. Die Achse beschreibt die Beziehung zwischen Ausgangsknoten und Suchknoten. Der Knotentest wird von der Achse durch zwei Doppelpunkte getrennt und Prädikate können in Klammern folgen und verfeinern die Suche. Ein Beispiel für einen XPath–Ausdruck wäre:

```
1 child::absatz[position() = last()][attribute::typ = "beispiel"]
```

Quelltext 7.28: XPath: Beispiel

Bei diesem Ausdruck wird das letzte Kindelement „absatz“ ausgegeben, wenn es ein Attribut „typ“ mit dem Wert „beispiel“ besitzt

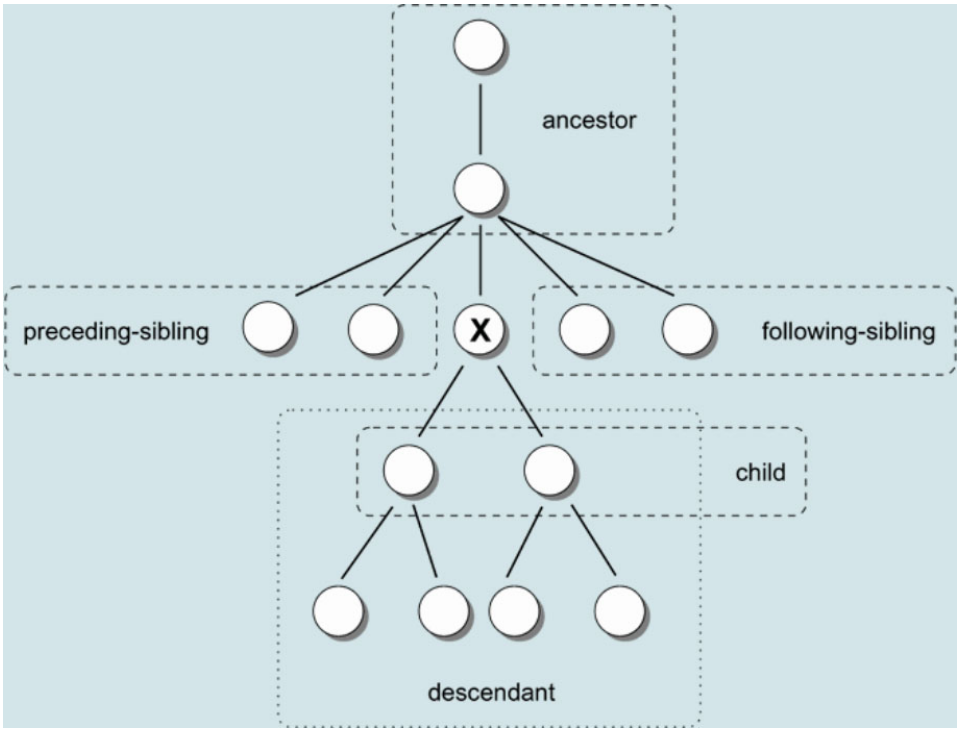


Abbildung 7.8.: XPath: Aufbau eines XML–Baums mit seinen Knotentypen

Ausdruck	Zugriff auf
/	Ursprung des Dokumentbaums, nicht das Top–Level–Element des Dokuments
textttchild::*	Alle Kindelemente des gegenwärtigen Knotens
child::absatz	Alle absatz–Kindelemente des gegenwärtigen Knotens
child::text()	Alle Textknoten des gegenwärtigen Knotens
child::node()	Alle Kindknoten des gegenwärtigen Knotens
attribute::name	Das Attribut name
attribute::*	Alle Attribute des gegenwärtigen Knotens
descendant::absatz	Alle absatz–Nachfahren des gegenwärtigen Knotens
/descendant::absatz	Alle absatz–Elemente in diesem Dokument
/descendant::kapitel/child::absatz	Alle absatz–Kindelemente, deren direkter Vorfahr kapitel ist
descendant-or-self::absatz	Alle absatz–Nachfahren des gegenwärtigen Knotens oder er selbst (wenn er ein absatz ist)
ancestor::absatz	Alle absatz–Ahnen des gegenwärtigen Knotens
ancestor-or-self	Alle absatz–Ahnen des gegenwärtigen Knotens oder er selbst
self::absatz	Der Knoten selbst, wenn es sich um einen absatz handelt
child::kapitel/descendant::absatz	Alle absatz–Elemente der kapitel–Kindelemente
child::absatz[position = 1]	Erstes absatz–Kindelement des gegenwärtigen Elements
child::absatz[position = last() - 1]	Vorletztes absatz–Kindelement des gegenwärtigen Elements
following-sibling::kapitel[position() = 1]	Das dem gegenwärtigen Knoten nächstliegende Element kapitel (nachfolgend)
child::absatz[position() = last()][attribute::typ = ‘‘beispiel‘‘]	Das letzte Kindelement absatz, wenn es ein Attribut typ mit dem Wert beispiel besitzt

Tabelle 7.1.: XPath: Beispielausdrücke

Zur Vereinfachung gibt es eine abgekürzte XPath–Schreibweise. Hierbei gilt, dass child die automatisch berücksichtigte Achse ist, d.h.:

- gibt alle Kindknoten aus.
- . ist der gegenwärtige Knoten
- .. ist der Elternknoten
- // bezieht sich auf das gesamte Dokument
- @ beschreibt ein Attribut

XPath bietet noch eine Reihe weiterer Funktionen, welche unter anderem für XSLT sehr wichtig sind. Diese sind im Folgenden aufgeführt:

Funktionsname	Rückgabewert	Rückgabetyp
`last()`	Anzahl (üblicherweise von Elementen in einem Kontext)	number
`position()`	Kontext–Position	number
`count(node-set)`	Anzahl der Knoten innerhalb des node–set	number
`id(object)`	Element mit dem Attribut id vom Wert object	node-set
`local-name(node-set?)`	Lokaler Bestandteil des ersten Knotennamens in node–set (ohne Namensraum–Bezeichner: mein–element, nicht ein–namensraum:ein–element)	string
`namespace-uri(node-set?)`	Namensraum–Bezeichner des ersten Knotennamens in node–set (ohne lokalen Bestandteil)	string
`name(node-set?)`	Namensraum–Bezeichner (falls vorhanden) und lokaler Bestandteil des ersten Knotennamens in node–set; normalerweise der im XML–Dokument verwendete Name	string

Tabelle 7.2.: XPath: Knotensatzorientierte Funktionen

Funktionsname	Rückgabewert	Rückgabetyp
`string(object?)`	Konvertiertes Objekt als Zeichenkette	string
`concat(string, string, string*)`	Die Aneinanderreihung der Argumente	string
`starts-with(string, string)`	Wahr, wenn der die erste Zeichenkette mit der zweiten beginnt (sonst unwahr)	boolean
`contains(string, string)`	Wahr, wenn die erste Zeichenkette die zweite beinhaltet (sonst unwahr)	boolean
`substring-before(string, string)`	Der Teil der ersten Zeichenkette, der dem Teil vorausgeht, den die zweite bezeichnet	string
`substring-after(string, string)`	Der Teil der ersten Zeichenkette, der dem Teil folgt, den die zweite bezeichnet	string
`substring(string, number, number?)`	Der Teil der ersten Zeichenkette, der an der Stelle beginnt, die das zweite Argument bezeichnet, bis zu der Stelle, die das dritte angibt (Beispiel: substring(äbcdefg", 3, 2) ergibt cd; das erste Zeichen wird durch 1 bezeichnet	string
`string-length(string?)`	Anzahl der Zeichen einer Zeichenkette	number
`normalize-space(string?)`	Die Zeichenkette ohne überflüssige Leerzeichen: Tabulator plus Leerschritt plus doppelter Return werden zu einem Leerzeichen	string
`translate(string, string, string)`	Die Zeichenkette des ersten Arguments, mit den Zeichen des zweiten durch die des dritten ersetzt	string

Tabelle 7.3.: XPath: Zeichenorientierte Funktionen

Funktionsname	Rückgabewert	Rückgabetyp
`boolean(object)`	Das in einen boolschen Wert konvertierte Argument	boolean
`not()`	Wahr, wenn das Argument unwahr ist (sonst unwahr)	boolean
`true()`	Wahr	boolean
`false()`	Unwahr	boolean
`lang(string)`	Wahr, wenn die Sprache des gegenwärtigen Knotens (oder des nächsten Vorfahren, in dem die Sprache definiert ist) mit dem Wert des Arguments übereinstimmt (sonst unwahr)	boolean

Tabelle 7.4.: XPath: Boolesche Funktionen

Funktionsname	Rückgabewert	Rückgabetyp
`number(object?)`	Der in number konvertierte Wert des Objekts (Zeichenkette, boolscher Wert, Knotensatz oder Objekt); fehlt das Argument, gilt der gegenwärtige Knoten	number
`sum(node-set)`	Die Summe aus den Werten, die sich daraus ergeben, die Zeichenketten der einzelnen Knoten des node–set zu number zu konvertieren (auch Kommentare und Verarbeitungsanweisungen, als Kindelemente eines Elements)	number
`floor(number)`	Die nächstkleinere Ganzzahl von number (untere Gauss–Klammer)	number
`ceiling(number)`	Die nächstgrößere Ganzzahl von number (obere Gauss–Klammer)	number
`round(number)`	Die dem Argument nächstgelegene Ganzzahl (wenn das zwei sind: die größere); Beispiel: round(0.5)=1, aber round(-0.5)=0.	number

Tabelle 7.5.: XPath: Numerische Funktionen

7.4. XSLT mit XALAN

7.4.1. XSLT

XSLT dient der Transformation in andere XML–Dokumente oder Formate wie z.B. PDF, XHTML oder Postscript. Es bedient sich der XPath Sprache zur Auswahl von Knoten, bedingten Abarbeitung und Erzeugung von Text

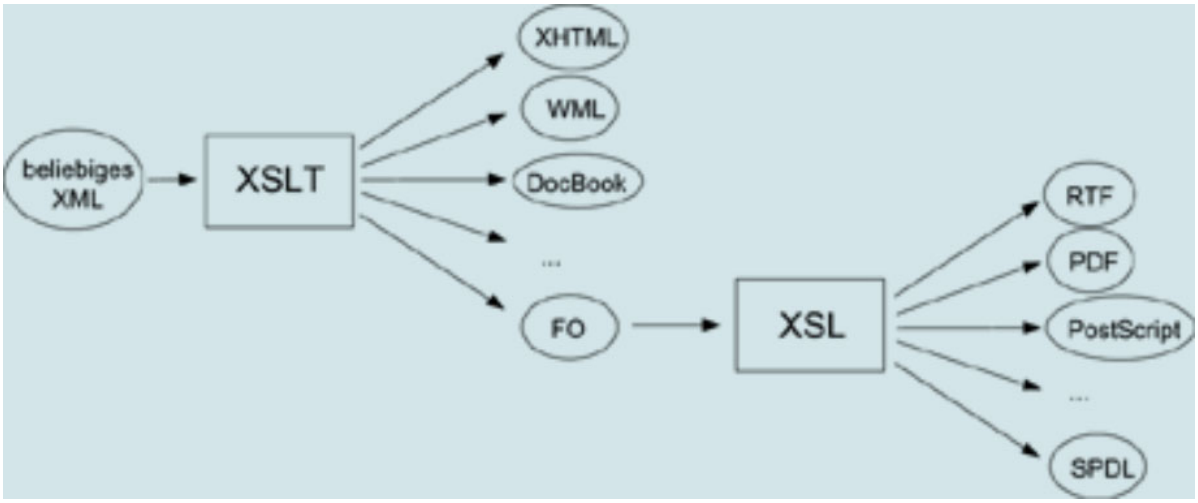


Abbildung 7.9.: XSLT: Transformationsmöglichkeiten mit Hilfe von XSLT und XSL FO

Ein XSLT–Dokument ist selbst ein XML Dokument. Es besteht aus mehreren Regeln (Templates) und alle XSLT Elemente sind dem folgendem Namensraum zugeordnet: http://www.w3.org/1999/XSL/Transform. Zur Transformation eines Dokuments wird ein XSLT Prozessor wie z.B. XALAN benötigt.

Jede Transformation beschreibt einen Satz von Regeln zur Transformation eines Quelldokuments in ein Zieldokument. Jede Regel ist eine Verknüpfung eines Auswahlmusters (Match Patterns) mit einer Schablone (Template). Die Auswahlmuster der Templates werden in XPath formuliert, sind also als eingeschränkte XPath Lokalisationspfade definiert. Von den von Xpath bekannten Achsen sind lediglich zwei erlaubt, `Child::` und `Attribute::`. Die abgekürzte XPath–Schreibweise ist erlaubt.

XSLT hat bestimmte Standardregeln für seine Templates definiert. Wird für einen Knoten z.B. kein Template gefunden werden 3 eingebaute Regeln (Build in Templates) ausgeführt. Diese bewirken für...

- ... Elemente, dass die Kindknoten rekursiv abgearbeitet werden.
- ... Text und Attribute, dass der Knoteninhalt in das Zieldokument übertragen wird.
- ... Processing Instructions und Kommentare, dass sie unterschlagen werden.

Es können auch mehrere Templates aufeinander treffen, wobei dann das höchstrangige Template ausgewählt wird. Je spezifischer der XPath–Ausdruck, desto hochrangiger ist das Template.

Element	Bedeutung
`<xsl:template>`	Template–Definition
`<xsl:apply-templates>`	Aufruf weiterer Templates aus dem aktuellen Kontext
`<xsl:call-template>`	Aufruf eines expliziten Templates
`<xsl:value-of>`	Ermittlung eines Wertes
`<xsl:text>`	Ausgabe von text
`<xsl:if>`	If–Bedingung (ohne else–Zweig)
`<xsl:choose>`	Auswahl
`<xsl:when>`	Kindelemente von xsl:choose; Bedingung
`<xsl:otherwise>`	Kindelemente von xsl:choose; Defaultverhalten
`<xsl:for-each>`	Schleife
`<xsl:number>`	Nummerierung
`<xsl:sort>`	Sortierung
`<xsl:param>`	Parameter mit Defaultwert in der Deklaration einer Funktion
`<xsl:with-param>`	Übergabeparameter bei Funktionsaufruf
`<xsl:variable>`	Konstante, der einmal ein Wert zugewiesen werden kann
`<xsl:output>`	Angaben über das Ausgabeformat
`<xsl:import>`	Einfügen von XSLT–Stylesheets
`<xsl:include>`	Einfügen von XSLT–Stylesheets
`<xsl:element>`	Erzeugung von Elementen
`<xsl:attribute>`	Erzeugung von Attributen
`<xsl:attribute-set>`	Erzeugung von Attributlisten
`<xsl:comment>`	Erzeugung von Kommentaren
`<xsl:processing-instruction>`	Erzeugung von Processing Instructions
`<xsl:message>`	Ausgabe von Nachrichten an den Benutzer

Tabelle 7.6.: XSLT: Wichtige XSLT–Ausdrücke

7.4.2. XALAN

Xalan Java ist ein XSLT Prozessor, welcher XSLT 1.0 und XPath 1.0 beinhaltet. Er transformiert XML Dokumente in z.B. andere XML Dokumente, Text oder HTML. Er basiert auf DTM (Document Table Model), welches bei XALAN aus DOM abgelöst hat. DTM stellt den Baum und Bauminhalt anhand von Integer Arrays und String Pools dar, was zu Geschwindigkeitsvorteilen gegenüber DOM führt, bietet allerdings nur Lesemöglichkeiten. XALAN ist leider nicht in Eclipse implementierbar und muss in diesem Projekt über die Konsole aufgerufen werden.

Xalan benötigt folgende Java Bibliotheken: `xalan.jar`, `xml-apis.jar`, `xerxesImpl.jar`

Ein Beispielbefehl von XALAN wäre z.B.:

```
1 java org.apache.xalan.xslt.Process -IN cds.xml -XSL libr.xsl -OUT cds.htm
```

Quelltext 7.29: XSLT: XALAN – Beispiel

Dieser Befehl wandelt die XML–Datei `cds.xml` mit Hilfe der XSL–Datei `libr.xsl` in die HTML–Datei `cds.htm` um.

8. Eclipse

8.1. Was ist Eclipse?

„Eclipse is a kind of universal tool platform – an open extensible IDE for anything and nothing in particular.“ (http://www.eclipse.org/)

Dieses ist der erste Satz, mit dem man auf der Homepage http://www.eclipse.org/ der Eclipse Stiftung begrüßt wird.
Präziser ausgedrückt verbirgt sich hinter Eclipse eine mächtige OpenSource Entwicklungsumgebung (IDE = integrated development environment) bzw. Plattform, in die ebenfalls OpenSource Tools integriert werden können. Sie bietet den Tool–Entwicklern weltweit eine hohe Flexibilität und Kontrolle über ihre Softwaretechnologien, die sie darin integrieren wollen. Die IDE ist plattformunabhängig – weil selber mit Java geschrieben – und herstellerübergreifend, was z.B. beim proprietären JBuilder nicht gegeben ist. Das bietet den Entwicklern und Programmierern eine plattformunabhängige, herstellerunabhängige und mehrsprachige Entwicklungsumgebung. Eclipse unterstützt eine auf Plug–Ins basierten Grundstruktur (framework), die eine Integration von Software–Tools vereinfacht.

Die Eclipse–Stiftung ist eine nicht auf Gewinn ausgerichtet Einrichtung, die ins Leben gerufen wurde, um die Erstellung, Entwicklung, Werbung und den Support voranzutreiben und die Verbindung zwischen OpenSource Gemeinschaft und den sich gegenseitig ergänzenden Produkten, Kapazitäten und Service zu pflegen. Auf der Eclipse–Homepage (http://www.eclipse.org/downloads/index.php) gibt es eine Vielzahl an Downloadmöglichkeiten von Servern weltweit u.a. mit Dokumentationen, der Eclipse–Plattform und dem SDK in unterschiedlichen Versionen und für alle Betriebssysteme. Sucht man unter oben angegebenen Adresse jedoch nach den angesprochenen Tools und Plug–Ins für die Eclipse Plattform, wird man enttäuscht. Diese sind hier nicht zu finden, denn man unterscheidet bei Eclipse verschiedene Projekte:

- The Eclipse Project
- The Eclipse Tools Project
- The Eclipse Technology Project
- The Eclipse Web Tools Platform Project

Die Plug–Ins sind dementsprechend unter dem „Eclipse Tools Project“ im Downloadbereich zu finden. Im Folgenden werden die weiteren Projekte mit ihrem Inhalt kurz umrissen:

8.1.1. The Eclipse Project

(siehe einleitenden Text)

8.1.2. The Eclipse Tools Project

Das „Tools–Project“ bietet eine große Auswahl der besten Tools für die Eclipse Plattform an. Hier wird ein breites Spektrum an Werkzeugen bereitgestellt, um eine Funktionsüberlappung mehrerer Tools zu verhindern und eine Plattform zum Informationsaustausch bereitzustellen.

8.1.3. The Eclipse Technology Project

Die Aufgabe des Technologie–Projekts ist der Vorantrieb und die ständige Weiterentwicklung von Eclipse durch OpenSource Entwickler, Forscher und Hochschulen. Hierbei gibt es eine Aufteilung in drei verwandte Bereiche:

- Forschung
- Umsetzung und
- (Schul–)Bildung.

Im Bereich der Forschung werden Lösungen für die Eclipse–Plattform entwickelt, wie z.B. Integration neuer Programmiersprachen, Tools und Entwicklungsumgebungen. Bei der Umsetzung geht es vornehmlich um die (Funktions–)Erweiterung der vorhandenen Eclipse–Struktur. Der Bildungsbereich hat sich letztendlich auf die Entwicklung und den Aufbau von Bildungsunterlagen, Unterrichtshilfen und Kursen spezialisiert.

8.1.4. The Eclipse Web Tools Platform Project

Die Mission hierbei ist die Errichtung einer erweiterbaren Tool–Plattform, auf welche Tool–Entwickler umfassendere und individuellere Webanwendungen und Werkzeugsammlungen erstellen können. Die Basis hierfür bildet weiterhin die Eclipse–Plattform. Damit soll den Entwicklern die Möglichkeit gegeben werden, einfach und schnell hoch

integrierte Werkzeuge zu programmieren, die virtuell mit jedem Web–Anwendungsserver arbeiten können, der die J2EE Spezifikation unterstützt.

Im Zusammenhang mit dem BizWeb – Projekt sind jedoch nur die ersten beiden Projekte interessant.

8.2. Die (Entstehungs–) Geschichte

Führende Unternehmen wie Borland, IBM, MERANT, QNX Software Systems, Rational Software3, Red Hat, SuSE, TogetherSoft3 und Webgain riefen im November 2001 das „Board of Stewards“ ins Leben. Bis zum Mai des Jahres 2003 traten noch folgende Firmen dem Board bei:
Serena, Sybase, Fujitsu, Hitachi, Instantiations, MontaVista, Scapa Technologies, Telelogic, ETRI, HP, MKS, SlickEdit, Oracle, Catalyst Systems, Flashline, Parasoft, SAP, teamstudio, TimeSys, Object Management Group (OMG), Fraunhofer Institute, Ericsson, LogicLibrary, M7 Corpororation, QA Systems, SilverMark, Inc., Advanced Systems Concepts, Genuitec, INNOOPRACT Informationssysteme GmbH, CanyonBlue, Ensemble Systems, Intel, Micro Focus, Tensilica und Wasabi.

Am 2. Februar 2004 gab das „Eclipse Board of Stewards“ die Reorganisation in ein nicht gewinnorientiertes Unternehmen (Corporation) bekannt. Ursprünglich war es IBM, die Eclipse als ein OpenSource–Projekt freigab, worauf Eclipse eine unabhängige Basis wurde, die die Plattformentwicklung trieb und weiter vorantreiben wird, was sowohl für die Softwareentwickler als auch die Endnutzer nützlich ist. Sämtliche Quellcodes und Technologien werden offen zugänglich für dieses schnell wachsende Zusammenspiel unterschiedlicher Gemeinschaften bereitgestellt.

Somit ergab sich ein Vollzeit–Eclipse–Management, das, wie oben bereits angesprochen, auf öffentliche Einrichtungen, Schulen, Forschungsstätten, etc. ausgedehnt wurde. Mit der Unterstützung von über 50 Mitgliedsfirmen umfasst Eclipse derzeit vier Haupt–OpenSource–Projekte und 19 Unterprojekte.
Um dieses Gesamtprojekt besser zu überschauen, wurde ein „Board of Directors“ gegründet mit vier Arten von Mitgliedschaften:

Strategien

- für Entwickler,
- für Konsumenten,
- für Add–in Provider und
- für Open Source Projekt Leiter.

8.3. Installation

Nun geht es in diesem Kapitel um die Installation der Eclipse Plattform, exemplarisch auf einem Windows–Betriebssystem. Hier werden alle Schritte erklärt, die durchgeführt werden müssen, um mit der Eclipse Plattform und dem Visual Editor arbeiten zu können. Links zu den notwendigen Anwendungen sind im Text enthalten. Es wird mit dem „Sun Java Standard Development Kit“ (im weiteren Verlauf JSDK oder nur SDK genannt) begonnen, das die Grundlage für alle Programmierarbeiten mit Java darstellt. Anschließend wird Eclipse entpackt und die notwendigen PlugIns hinzugefügt.

8.3.1. Sun Java SDK

Bei der Eclipse–Installation ist es zunächst notwendig die „Java 2 Platform, Standard Edition, v 1.4.2 (J2SE)“ von der Sun–Homepage unter:
http://java.sun.com/j2se/1.4.2/download.html

herunterzuladen und zu installieren, falls das Paket nicht schon auf dem Rechner installiert ist. Hierzu klickt man auf den Link „Download J2SE SDK“, akzeptiert die Lizenzbestimmungen und man gelangt auf die Downloadseite für das SDK für unterschiedliche Windows, Linux und Solaris Betriebssystemausführungen.

Nachdem das passende Paket heruntergeladen und installiert ist, geht es an die Vorbereitungen und die Durchführung der Eclipse Plattform Installation.

8.3.2. Eclipse SDK

Die Installation vom Eclipse Standard Development Kit (SDK) verläuft allerdings nicht typisch im Sinne einer „normalen“ Programminstallation unter Windows, bei der viele Aufgaben automatisch erledigt werden, Systemdateien verändert werden und sich das Programm in der Registry verewigt, was bei einer Deinstallation evtl. nicht wieder rückgängig gemacht wird. Bei Eclipse ist das Entpacken des zip–Archives und das Ausführen der eclipse.exe ausreichend. Eclipse startet!
Der Download von Eclipse erfolgt über die Seite:
http://www.eclipse.org/downloads/index.php,

auf der man aus einer Liste einen möglichst nahe gelegenen ftp– oder http–Server auswählt. Darauf wird das Verzeichnis „`S-3.0M7-200402122000`“ gewählt, was die Version enthält, mit der während des BizWeb–Projektes gearbeitet wird, und darin wiederum das Archiv „`eclipse-SDK-3.0M7-win32.zip`“ zum downloaden. Das zip–Archiv wird nun der Einfachheit halber unter `C:\Java\` entpackt. Natürlich kann jeder Benutzer sein favorisiertes Verzeichnis verwenden. Die Datei eclipse.exe im automatisch erstellen Verzeichnis `C:\Java\eclipse\` kann sofort ausgeführt werden. Beim Start sucht sich Eclipse (normalerweise) automatisch das installierte JSDK. Leider ist das jedoch nicht

immer der Fall und der Startversuch von Eclipse bricht mit einer Fehlermeldung ab, dass das JSDK oder eine ihrer Komponenten nicht gefunden wurde. Ein Neustart des Rechners bringt hier meistens Abhilfe. Hilft auch diese Maßnahme nicht, sollten das JSDK deinstalliert und neu installiert werden, evtl. mit einem anschließenden Neustart.

8.3.3. Visual Editor und alle anderen PlugIns

Hat die Installation beider Programme (JSDK und Eclipse) geklappt, sollte das Programm Eclipse beendet werden, denn jetzt soll der Visual Editor installiert werden. Die nun beschriebene Vorgehensweise zur Installation eines PlugIns kann für alle anderen PlugIns verwendet werden.
Der visuelle Editor ist für die vereinfachte Erstellung einer grafischen Oberfläche per Drag–and–Drop–Programmierung mit Swing und AWT sinnvoll. Zu finden ist das Plug–In unter: http://download.eclipse.org/tools/ve/downloads/drops/I-I20040218-200402181711

Von dieser Seite müssen folgende Archive heruntergeladen werden, die für den Visual Editor notwendig sind:

- `emf_2.0.0_20040127_1738SL.zip`
- `GEF-runtime-I20040212.zip`
- `VE-runtime-I20040218.zip` (als letztes Paket der drei zip–Archive entpacken)

Das Zielverzeichnis bzw. die Zielverzeichnisse für die einzelnen PlugIns können im Packprogramm „WinZip“ angezeigt werden, wenn man ein Archiv öffnet:

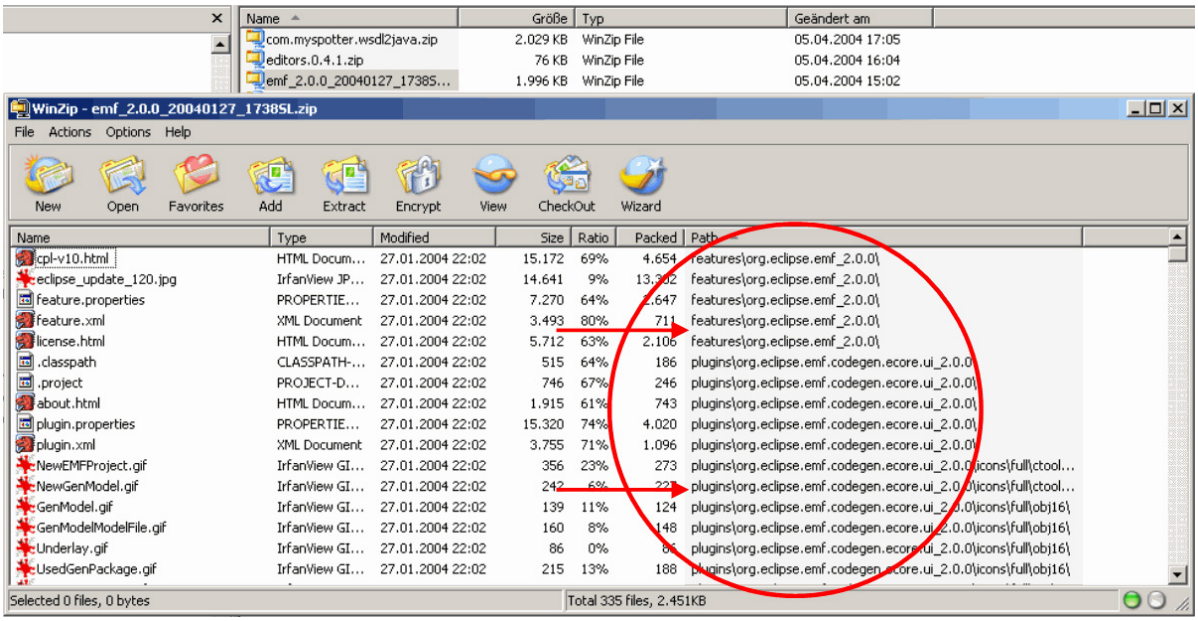


Abbildung 8.1.: Eclipse: Inhalt eines PlugIn–Zip–Archives

Das mit WinZip geöffnete Archiv[1] (Abbildung 8.1) zeigt in der rechten Spalte die Verzeichnisse an, die beim Entpacken automatisch angelegt werden. In diesem Falle bedeutet die Installation des PlugIns, dass der Inhalt des Archives in das Verzeichnis entpackt werden muss, welches die Verzeichnisse „features“ und „plugins“ enthält. Die folgende Abbildung zeigt, dass sich die beiden genannten Ordner im Eclipse–Hauptverzeichnis befinden.

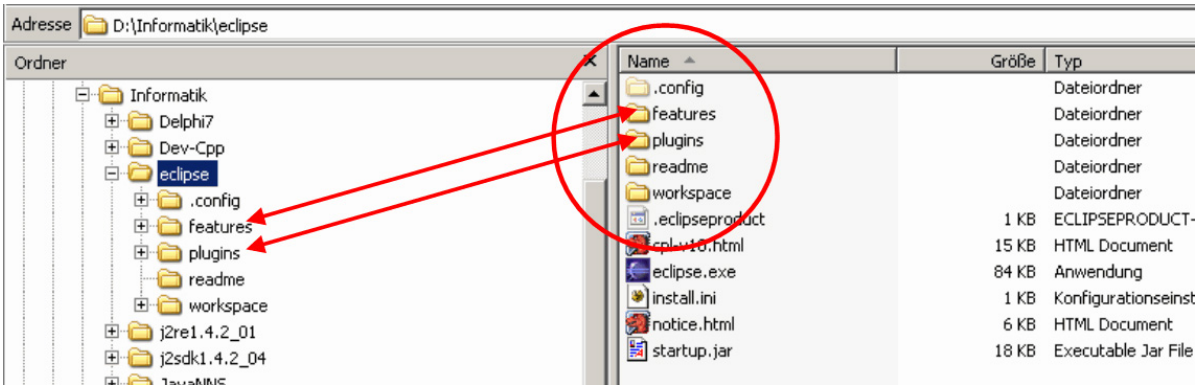


Abbildung 8.2.: Eclipse: Programmverzeichnisse

Das Entpacken kann auf zwei Arten durchgeführt werden:

1. Entpacken im übergeordneten Verzeichnis:
   - das jeweilige Archiv in den übergeordneten Ordner kopieren (das Eclipse–Verzeichnis im Falle des EMF–Pakets),
   - im Kontext–Menü (Rechts–Klick auf das Archiv) „Hier entpacken“ auswählen
2. Per Hand Ordner für Ordner in die richtigen Verzeichnisse entpacken:
   - das jeweilige Archiv in einem beliebigen Ordner entpacken,
   - die entpackten Dateien und Verzeichnisse in die passenden Verzeichnisse von Eclipse kopieren/verschieben

Kurze Erläuterung der Funktionen der Pakete

EMF (Eclipse Modeling Framework) EMF ist ein Modellgerüst und Code–Erstellungswerkzeug, mit dem Tools und andere Anwendungen generiert werden können, die auf einem strukturierten Datenmodell basieren. (siehe auch http://www.eclipse.org/emf)

GEF (The Graphical Editing Framework) GEF erlaubt es dem Entwickler, mit einem bereits vorliegenden Anwendungsmodell sehr einfach einen umfangreichen grafischen Editor zu erstellen. (siehe auch http://www.eclipse.org/gef/)

[1] emf_2.0.0_20040127_1738SL.zip

VE (Visual Editor) Das Eclipse Visual Editor Projekt ist ein Grundgerüst für die Erstellung von GUI–Gestaltern. (siehe auch http://www.eclipse.org/vep/)

Ist der Download beendet, müssen die drei neuen Archive in der oben angegebenen Reihenfolge folgendermaßen entpackt werden:

- die Datei emf_2.0.0_20040127_1738SL.zip in das Verzeichnis `C:\eclipse` kopieren und dort entpacken (im Kontextmenü des Explorer „Hier entpacken“)
- das Gleiche passiert mit der Datei `GEF-runtime-I20040212.zip`

Ist das Entpacken und automatische Einbinden des VE–Tools durch Eclipse erfolgreich verlaufen, können noch beliebig viele andere PlugIns z.B. für Tomcat, XML oder UML eingebunden werden, was hier jedoch nicht weiter beschrieben wird, weil das Prinzip der Installation immer das selbe ist.
Sind alle benötigiten PlugIns installiert, ist es ratsam unter dem Menüpunkt „Help“ → „Software Updates“ → „Manage Configuration...“ nachzusehen, ob wirklich alle entpackten PlugIns von Eclipse erkannt und eingebunden worden sind, wie es auf den folgenden Screens zu sehen ist:

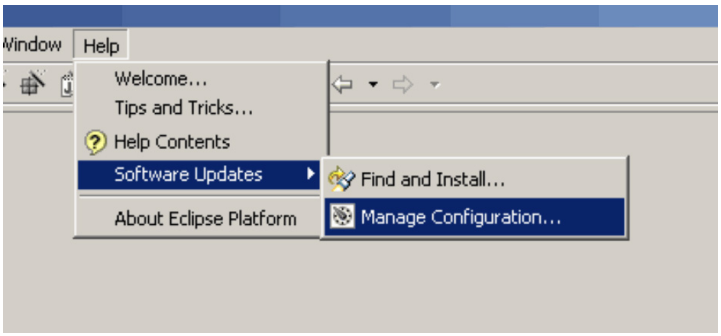


Abbildung 8.3.: Eclipse: Überprüfen der PlugIns (1)

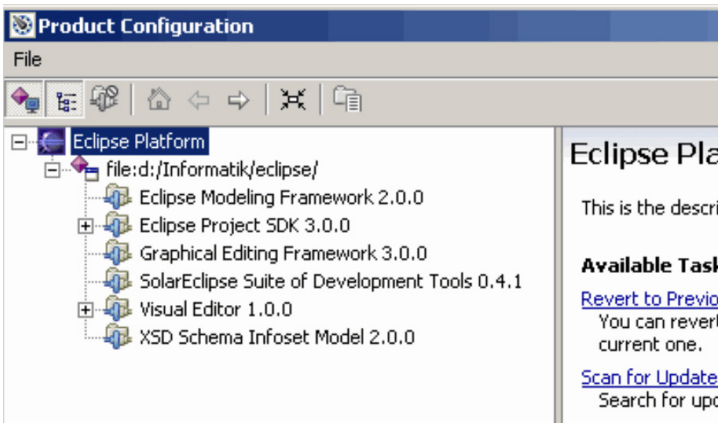


Abbildung 8.4.: Eclipse: Überprüfen der PlugIns (2)

8.4. Programmbeschreibung

Nach dem Start von Eclipse zeigt sich dem Benutzer eine übersichtliche Programmoberfläche, die an die vom JCreator oder Visual Studio erinnert.

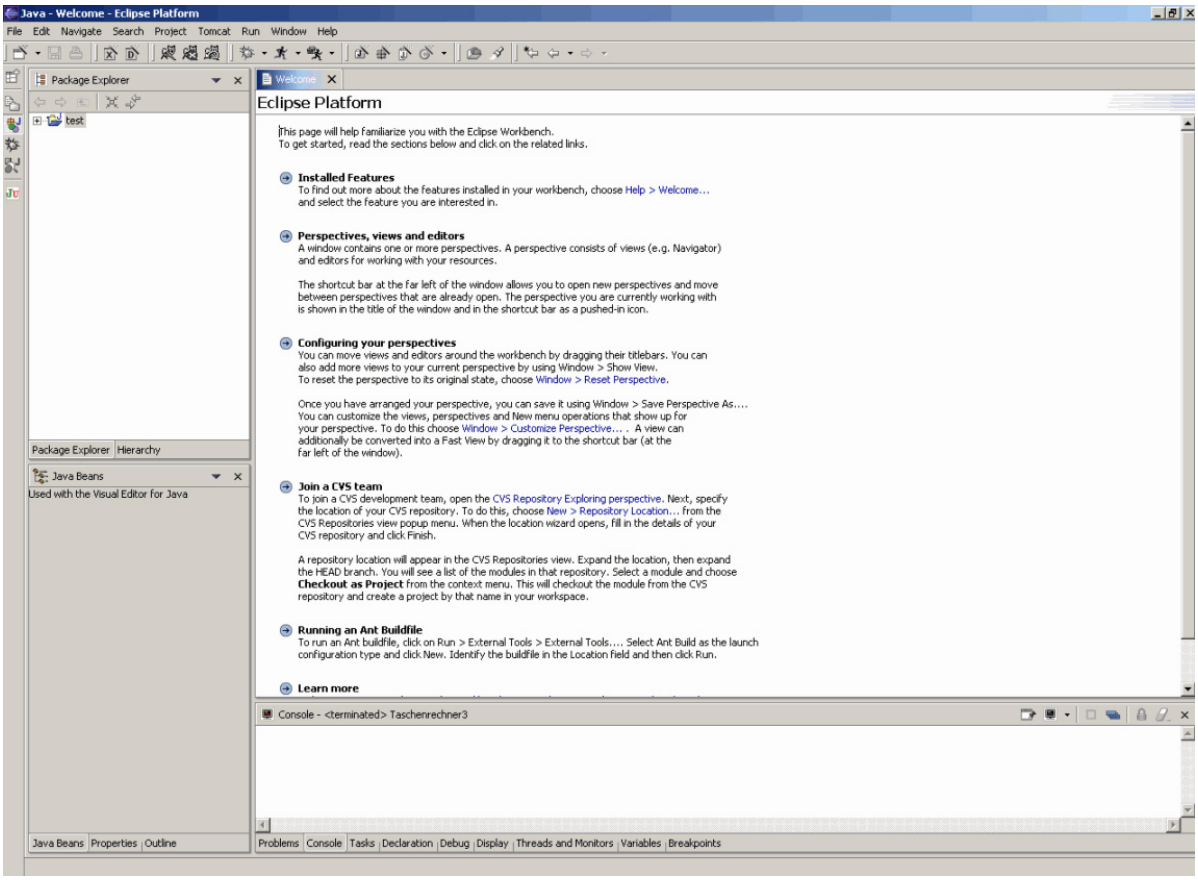


Abbildung 8.5.: Eclipse: Willkommens–Bildschirm

Die Spracheinstellung (sämtliche Beschriftungen im IDE) ist Englisch und kann (noch) nicht geändert werden.
Im weiteren Verlauf wird näher auf die unterschiedlichen Ansichten und Funktionalitäten der GUI eingegangen, die wichtigsten Einstellungen beleuchtet und anschließend anhand eines kleinen Beispiel–Javaprogramms die einzelnen Elemente wie den Source Editor, Debugger, Hilfefunktion, etc. beschrieben.

8.4.1. Beschreibung der Eclipse–Oberfläche

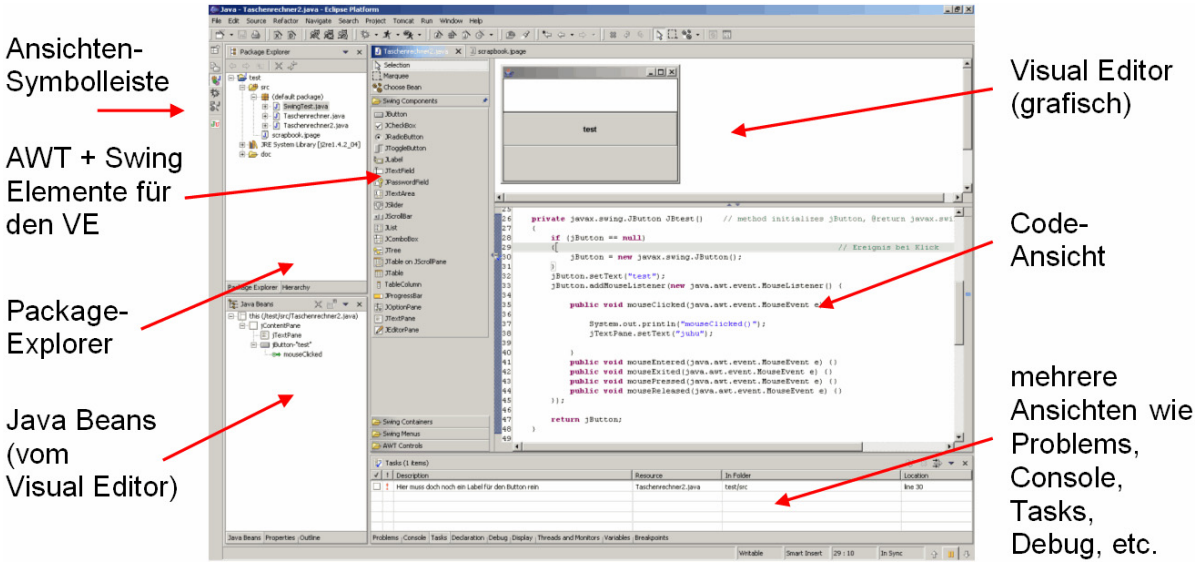


Abbildung 8.6.: Eclipse: Der Aufbau des GUI

8.4.2. Nützliche Einstellungen

Standardmäßig sind in Eclipse die meisten Voreinstellung schon gut gesetzt, so dass ein sofortiges Losprogrammieren möglich ist. Möchte man jedoch die Oberfläche und „Verhaltensweisen“ von Eclipse an die eigenen Bedürfnisse anpassen, muss man einen Blick in die sehr umfangreichen Einstellungen im Menü „Window“ → „Preferences“ werfen. An dieser Stelle sollen nicht sämtliche Einstellungen betrachtet werden, das würde den Rahmen dieser Ausarbeitung sprengen, zumal in der umfangreichen Hilfe zu nahezu jeder Frage bezüglich der „Preferences“ eine Antwort zu finden ist. Vielmehr werden hier einige wenige, aber für viele Anwender interessante und nützliche Einstellungen betrachtet, die in der Standardkonfiguration vielleicht nicht jedem Benutzer als optimal erscheinen.

Compiler und Editor auf das eigene JDK bzw. SDK umstellen

Zu Beginn der Vorstellung von Eclipse wurde das neuste Java SDK installiert. Um auch dieses und kein evtl. älteres, noch installiertes DK zu verwenden, muss unter der „Einstellung“ → „Window“ → „Preferences“ → „Java“ → „Installed JREs“‘ das Auswahlkästchen bei „j2sdk1.4.2_04“ markiert werden. So ist ein schnelles Wechseln zwischen den installierten JSDK–Version möglich.

Ordnerstruktur bei neuen Projekten

Bei jedem Compiliervorgang werden vom Compiler `.class`–Dateien erzeugt, die sich standardmäßig im gleichen Ordner befinden wie die `.java`–Datei. Schon bei kleinen Programmen mit mehreren Klassen kann dies unübersichtlich werden. Eclipse bietet als Lösung des Problems an, beim Erstellen eines neuen Java–Projektes (wie man das macht, wird im nächsten großen Kapitel beschrieben) zwei Ordner anzulegen, den einen für die Quelltextdateien (.java) und den anderen für die Klassendateien (.class). So werden Quelltext und weitere Dateien sauber getrennt. Im Menüpunkt „Java“ → „New Project“ → „Folders“ gibt es die Möglichkeit, diese Funktion standardmäßig einzustellen und die Benennung der Ordner den eigenen Vorlieben anzupassen.

Workingspace verlegen

Wenn nicht anders angegeben speichert Eclipse alle neu angelegten Projekte im vorgegebenen Verzeichnis „workspace“ im Eclipse–Verzeichnis `C:\eclipse\workspace`. Möchte man seine Projekte jedoch woanders speichern, kann dies sehr schnell geändert werden. Existiert z.B. eine Verknüpfung zu Eclipse auf dem Desktop kann das workspace–Verzeichnis folgendermaßen verschoben werden:
„Rechtsklick auf die Verknüpfung“ → „Eigenschaften“ → „Verknüpfung“ und hier bei „Ausführen in“ den gewünschten Pfad eingeben.
Z.B. `C:\Eigene Dateien\WebBiz\Java\workspace`

Java Code Generation

Eine gute Dokumentation und Kommentare im Quelltext tragen einiges zu einem guten Programm bei. Diese zumeist lästige, aber notwendige Aufgabe des Programmierers unterstützt Eclipse mit kurzen automatischen generierten Texten. Zum Beispiel wird an den Anfang jeder neuen `.java`–Datei mit den Standardeinstellungen folgender Text eingefügt:

```
1  /*
2  * Created on 16.04.2004
3  *
4  * To change the template for this generated file go to
5  * Window - Preferences - Java - Code Generation - Code and Comments
6  */
7  /**
8  * @author Christian
9  *
10 * To change the template for this generated type comment go to
11 * Window - Preferences - Java - Code Generation - Code and Comments
12 */
```

Quelltext 8.1: Eclipse: Generierter Auto–Text

In den Einstellungen unter „Java“ → „Code Generation“ können z.B. beim Erstellen einer neuen Methode oder einer neuen Klasse Texte mit vordefinierten Variablen eingefügt werden.

Editor – Einstellungen

Unter „Workbench“ → „Editors“ → „Text Editor“ ist es je nach Geschmack des Programmierers möglich, die Zeilennummern mit „Show Line Numbers“ anzuzeigen.

Java–Editor – Einstellungen

Damit unter Java im Quellcodefenster die Zeilennummern angezeigt werden, muss unter „Java“ → „Editor“ die Option „Show Line Numbers“ ebenfalls markiert sein.

Javadoc

Unter „Java“ → „Javadoc“ kann die `javadoc.exe` aus dem `C:\j2sdk1.4.2_4\bin`–Verzeichnis ausgewählt werden, mit der Anhand des Quellcodes und den darin vorliegenden Kommentaren die automatische Generierung einer Dokumentation u.a. mit allen Imports, Klassen, Variablen, Methoden möglich ist.

8.4.3. Die Funktionalitäten im einzelnen

Es sollen hier nicht alle Funktionalitäten der Eclipse–Plattform beschrieben werden, dazu gibt es eine umfassende Hilfe (F1–Taste drücken oder im Menü über „Help“). Vielmehr gibt dieser Abschnitt eine Beschreibung der am häufigsten verwendeten Funktionen, ohne Garantie auf Vollständigkeit.

Ein neues Java–Projekt anlegen

Ohne ein Java–Projekt kann nicht programmiert werden. Zum Anlegen eines neuen Java–Projektes folgendes durchführen:

- im Menü: „File“ → „New“ → „Project“
- „Java“ (standardmäßig markiert) auswählen
- Namen des Projektes und Speicherort angeben
- Mit Klick auf „Finish“ wird das neue Java–Projekt angelegt

Benutzeroberfläche den Bedürfnissen anpassen

In Eclipse gibt es eine Vielzahl an Ansichten und Fenstern, die geöffnet, geschlossen und neu angeordnet werden können. So ist es z.B. möglich den Reiter eines Fensters, in dem dessen Name steht, per Maus auf einen anderen Reiter zu ziehen. Es entsteht eine neue Registerkarte mit dem verschobenen Fenster, so dass ein schnelles Hin– und Herwechseln zwischen den Ansichten möglich ist, ohne weiteren Platz auf dem Bildschirm zu belegen. Natürlich kann diese Aktion in umgekehrter Reihenfolge durchgeführt werden.
Ein Doppel–Klick auf das Register „Code–Fenster“ vergrößert selbiges und schließt automatisch die Fenster mit den Packages und Explorern. Das ist sehr sinnvoll, wenn eine Zeit lang nur programmiert oder mit dem Visual Editor gearbeitet wird.
Solle man bei den vielen Möglichkeiten der Fenster– und Oberflächengestaltung den Überblick verlieren und ein Fenster nicht mehr wieder finden, kann dieses im Menü unter „Window“ → „Show View“ aus einer Vielzahl an Ansichten ausgewählt werden.

Visual Editor

Die Arbeit mit dem Visual Editor wäre nur halb so gut, wenn es das Fenster mit den „Java Beans“ und „Properties“ nicht gäbe. Darin können so gut wie alle Einstellungen für Swing bzw. AWT–Elemente benutzerfreundlich eingestellt werden à la Visual Basic oder Delphi.

Sofortige Syntaxkorrektur und Compilieren

Ein großer Vorteil von Eclipse liegt darin, dass, ähnlich wie bei Word die Rechtschreibprüfung, der Quellcode sofort überprüft und compiliert wird, wenn Eingaben getätigt werden. Es muss also nicht jedes Mal compiliert werden, wenn der Code auf Richtigkeit überprüft werden soll, das geschieht just–in–time. Ein Fehler wird insgesamt dreimal kenntlich gemacht:

1. Im unteren Fenster „Problems“
2. Gelber Punkt an der linken Seite des Quellcode–Fensters in der entsprechenden Zeile
3. Rote Linie an der rechten Quellcode–Fensterseite in der entsprechenden Zeile. Mit Klick darauf gelangt man direkt zum Fehler.

Fehlerbehandlung und Hilfen

Eclipse bietet dem Programmierer eine mächtige Hilfefunktion an, indem, wie eben angesprochen, direkt bei der Codeeingabe selbiger überprüft wird, und bei Fehlern eine

Lösungsmöglichkeit angeboten wird, wenn man auf den gelben Punkt klickt. Die Lösung muss nicht unbedingt immer so sinnvoll sein wie in der folgenden Abbildung:

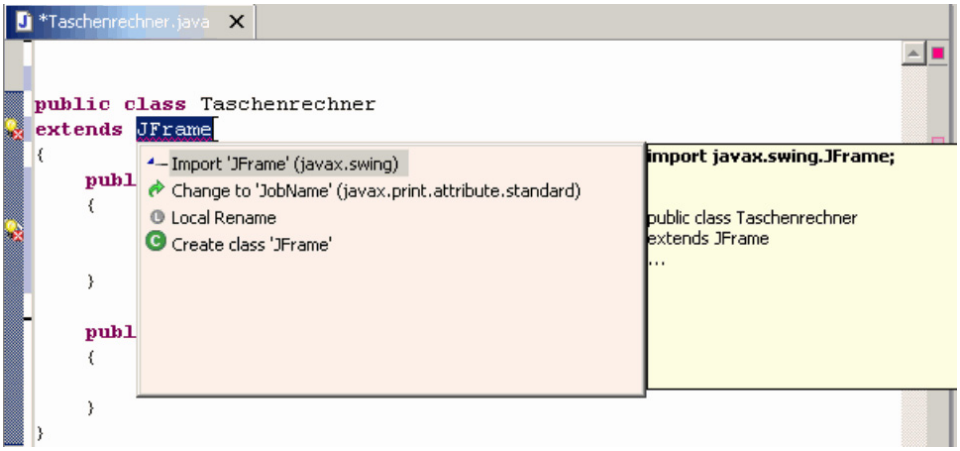


Abbildung 8.7.: Eclipse: Intelligente Hilfestellung bei Code–Problemen

Erläuterung zur Abbildung:
Die Klasse „Taschenrechner“ wird von JFrame abgeleitet, doch wird JFrame noch nicht importiert.
Die Hilfe gibt hier u.a. die Lösungsmöglichkeit an `Import JFrame (javax.swing)`, was in diesem Fall die richtige Lösung ist.

Tasks (Notzien zu Quellcodezeilen)

Eine nützliche Funktion in Eclipse sind die „Tasks“. Sie stellen eine Art Notiz dar, die für jede Code–Zeile eingetragen werden kann. Tasks werden sowohl im Fenster „Tasks“ angezeigt, als auch rechts im Quellcode–Fenster, sichtbar als blaue Linie.

Erstellung eines Tasks:

- Rechts–Klick rechts ins Graue neben die Zeile
- Im Kontextmenü „Add Task“ auswählen

Wenn das Task–Fenster nicht zu sehen ist, kann es unter „Window“ → „Show View“ → „Task“ bzw. „other...“ → „Tasks“ angezeigt werden.

Debugger und Breakpoints

Selbstverständlich gibt es auch einen Debugger, der in der gleichnamigen Ansicht gestartet und bedient werden kann. Breakpoints werden durch Doppelklick in den grauen Bereich links neben der betreffenden Zeile gesetzt.

Beim Debuggen:

- Step into = F5
- Step over = F6
- Step return = F7
- Resume = F8
- Letztes aufgerufenes Programm ausführen = Strg + F11
- Letztes aufgerufenes Programm debuggen = F11
- remove terminated programs (hier können noch aktive Programme. beendet werden)

Scrapbook page

Eine Scrapbook page ist dann nützlich, wenn man nur ein paar Codezeilen ausprobieren möchte, ohne ein neues Projekt mit main–Methode, etc. anlegen zu müssen. Der zu testende Code muss markiert werden. Mit Rechts–Klick auf den markierten Bereich kann mit dem Eintrag „Execute“ im Kontextmenü der Code getestet werden.
Eine Scrapbook page wird unter „File“ → „New“ → „Scrapbook page“ für das gerade aktuelle Projekt erstellt.

Beispiele für die Nutzung von Eclipse für Java–Projekte befinden sich im Anhang A.1.

9. Apache Tomcat / Java Servlets

9.1. Apache Tomcat

Die Apache Group, dessen Projekte auch den Tomcat Server umfassen, wurde im Februar 1995 gegründet. Ursprüngliches Ziel ist es gewesen, Fehler im Programm des NCSA-Webservers (NCSA = National Center for Super Computing Applications), des damals meist verbreiteten Webservers zu korrigieren – daher auch der Name: Apache Group = „a patchy group“. Seit 1999 nennt man jene Gruppe „Apache Software Foundation“, welche bis heute viele Open–Source–Projekte durchführt oder unterstützt. Eines dieser Projekte ist das Jakarta–Projekt, welches mit Hilfe von Sun Microsystems eine Open Source Servlet Engine für den Apache Webserver zur Verfügung stellen will. Zu diesem Projekt gehört auch der Tomcat–Server, welcher den Server (und damit die Basis) für eben jene Servlets stellt. Die erste Version von Tomcat erschien 1998 und trug die Nummer 3.0 – sie ersetzte JSDK (Server) 2.1.

Bei der Entwicklung des Servers gab es diesen Namen noch nicht, erst als die Technologie in einem Buch veröffentlicht werden sollte, kam der Name zustande. Da Bücher, die im O'Reilly–Verlag erscheinen grundsätzlich Tiere auf dem Titelbild tragen, musste sich der Entwickler (James Duncan Davidson) Gedanken machen, welches Tier sich für diese Technologie eignet. Ein Kater (engl. Tomcat) passte seiner Meinung nach am besten.

9.1.1. Was ist Tomcat?

Tomcat ist eine Referenzimplementierung der Java Servlet 2.3 und Java Server Pages 1.2 Spezifikationen und wurde bzw. wird vom Apache Jakarta Project entwickelt. (http://jakarta.apache.org). Ältere Implementationen, wie z. B. das Apache JServ–Modul, werden durch den Tomcat abgelöst.

Der Tomcat ist in der Lage dem Client außer Webanwendungen auch statische Seiten zu liefern und ist somit ein vollwertiger Web–Server. In der Praxis wird der Tomcat–Server oft in den Apache–Web–Server integriert, welcher dann für die statischen Seiten zuständig ist. Der Tomcat bearbeitet dann nur noch die Anfragen an Servlets und JSP's.

9.1.2. Vorbereitung / Installation

In den folgenden Unterpunkten werden benötigte und hilfreiche Dateien genannt, sowie die Installation des Tomcat–Servers unter Linux und Windows beschrieben.

Benötigte Dateien

Im Projekt haben wir uns auf die Tomcat–Version 5.0.19 geeinigt. Unter der URL http://jakarta.apache.org/tomcat ist die neuste Version erhältlich. Ältere Releases sind in den „Apache Archives“ unter http://archive.apache.org/dist/jakarta/tomcat-5 verfügbar.

Hilfreich für die Entwicklung von Webanwendungen ist ein Eclipse–Plugin für Tomcat von Sysdeo, welches unter http://www.sysdeo.com/eclipse/tomcatPlugin.html heruntergeladen werden kann.

Dateinamen:

- Tomcat für Linux: `jakarta-tomcat-5.0.19.tar.gz`
- Tomcat für Windows: `jakarta-tomcat-5.0.19.exe`
- Eclipse–Plugin: `tomcatPluginVxyz.zip`

Installation unter Windows

Unter Windows muss einfach die Datei `jakarta-tomcat-5.0.19.exe` ausgeführt werden, um den Tomcat–Server zu installieren. Es sollte gleich ein Administrator–Passwort vergeben und der Connector–Port von 8080 auf 80 gesetzt werden:

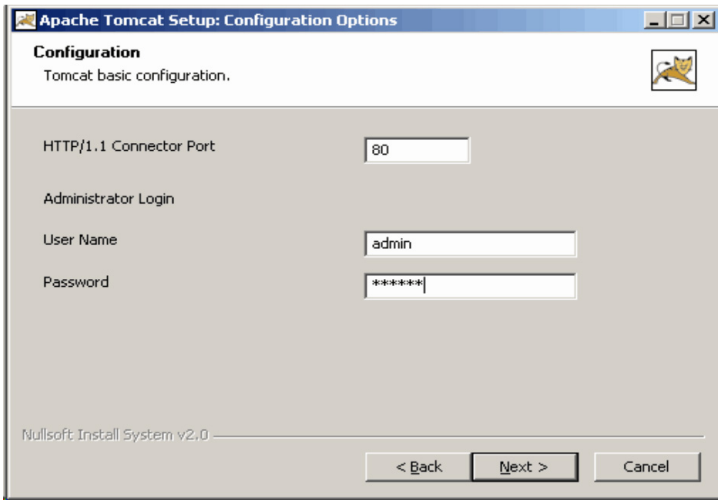


Abbildung 9.1.: Tomcat: Setup

Danach muss noch die Position des Java2 SDKs oder JREs angegeben werden. Tomcat wird standardmäßig als automatischer Dienst eingerichtet. Um zu testen ob die Installation erfolgreich war, gibt man http://localhost bzw. `http://localhost:<port>` (z.B. http://localhost:8080), wenn nicht Port 80 als HTTP–Connector–Port gesetzt wurde in den Browser ein. Natürlich kann man auch die lokale IP–Adresse oder den Hostnamen bzw. die Loopbackadresse (`127.0.0.1`) eingeben. Nun sollte folgende Seite erscheinen:

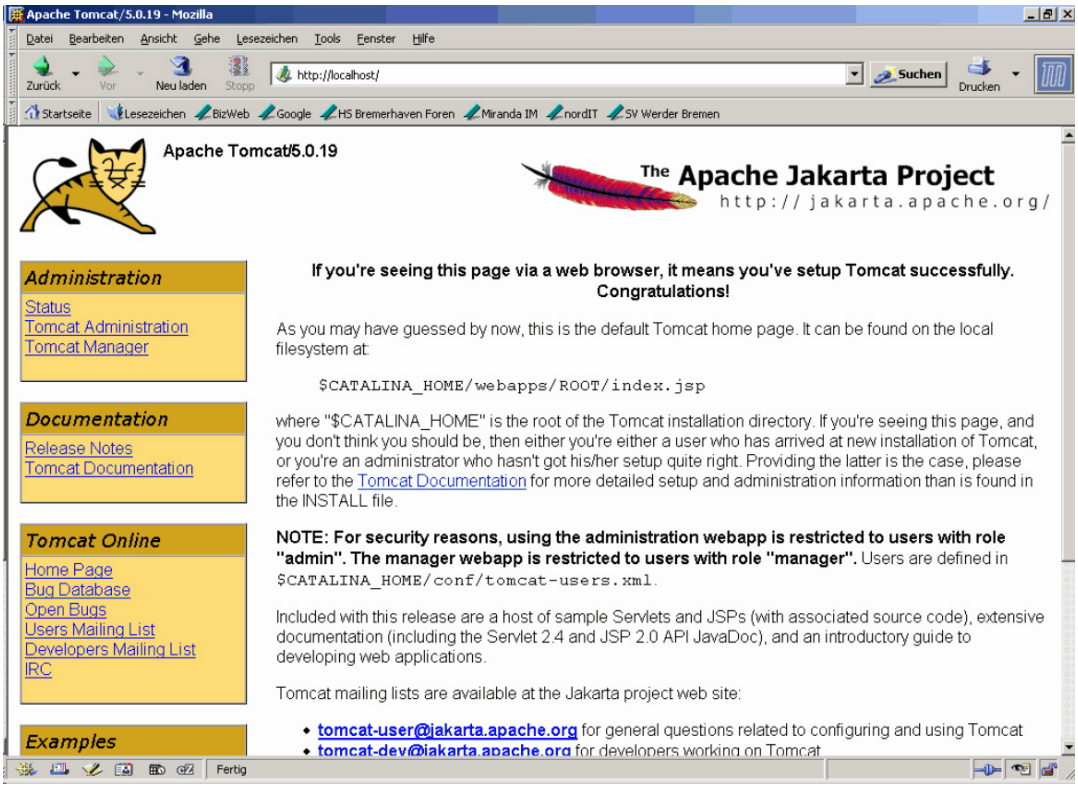


Abbildung 9.2.: Tomcat: Startseite

Installation unter Linux

Als „root“ wird die o.g. Archivdatei mit „`tar -xvzf jakarta-tomcat-5.0.19.tar.gz`“ im Verzeichnis `usr/local` entpackt. Um den Tomcat starten zu können muss vorher die Umgebungsvariable `JAVA_HOME` gesetzt werden: „`JAVA_HOME=/usr/lib/java/jdk`“. Nun wird im Verzeichnis `/usr/local/jakarta-tomcat-5.0.19/bin` das Kommando „`sh startup.sh`“ eingegeben und ausgeführt. Die ausführbare Datei `startup.sh` ist das Start–Script für den Tomcat–Server. Nachdem der Server hochgefahren ist, sollte die Startseite unter http://localhost:8080 erreichbar sein (s.o.). Herunterfahren lässt sich der Server mit dem Script `shutdown.sh`.

Nun gibt es das Problem, dass nach einem Neustart des Rechners Tomcat nicht gestartet und die Umgebungsvariable `JAVA_HOME` nicht mehr gesetzt ist. Als Lösung bietet sich

an, den Tomcat als Systemdienst, als so genannten Daemon einzusetzen. Dazu fügt man in den Run Commands von Linux ein entsprechendes Script ein.

Folgender Ablauf wird geraten:

1. Ins Verzeichnis `/etc/init.d` wechseln
2. mittels „`touch tomcat`“ eine neue Datei mit dem Namen `tomcat` anlegen
3. ins Verzeichnis `/etc/init.d/rc3.d` wechseln
4. Befehl eingeben: „`ln -s ../tomcat S99tomcat`“
5. Befehl eingeben: „`ln -s ../tomcat K01tomcat`“
6. ins Verzeichnis `/etc/init.d/rc5.d` wechseln
7. Punkte 4. und 5. wiederholen
8. wieder nach `/etc/init.d` wechseln
9. Datei `tomcat` in einem Editor öffnen (z.B. „`vi tomcat`“); Quellcode (s.u.) eingeben und abspeichern
10. Rechte für die Datei `tomcat` mittels „`chmod 755 tomcat`“ setzen
11. Rechner neu starten

Quellcode:

```
#! /bin/sh
TOMCATBIN=/usr/local/jakarta-tomcat-5.0.19/bin
export PATH=$PATH:$TOMCATBIN
export JAVA_HOME=/usr/lib/java/jdk
case "$1" in
  start)
    exec /usr/local/jakarta-tomcat-5.0.19/bin/startup.sh
    echo -n "Starting Tomcat"
    ;;
  stop)
    exec /usr/local/jakarta-tomcat-5.0.19/bin/shutdown.sh
    echo -n "Shutting down Tomcat"
    ;;
esac
```

Quelltext 9.1: Tomcat: Start–Script

Tomcat Eclipse Plugin

Das Tomcat Plugin für Eclipse wird – wie im Eclipse–Abschnitt beschrieben – installiert. In Eclipse müssen nach der erfolgreichen Installation unter Window → Preferences im Punkt „Tomcat“ noch einige Einstellungen vorgenommen werden (Abbildung 9.3). Zuerst muss angegeben werden welche Version des Tomcat verwendet wird. In unserem Fall muss das Optionsfeld „Version 5.x“ aktiviert werden. Darunter wird das Verzeichnis in dem der Tomcat–Server installiert wurde angeben. Danach wird der Ort der Konfigurationsdatei automatisch gesetzt. Das Tomcat Basisverzeichnis ist frei wählbar. Nun ist es möglich unter File → New → Project ein Tomcat–Projekt anzulegen sowie den Tomcat–Server über das Menü „Tomcat“ oder die entsprechenden Shortcuts zu starten, zu stoppen und neu zu starten (Abbildung 9.4). *Diese Angaben gelten für die Version 221.*
Das Plugin unterstützt auch das Deployment der Webanwendungen auf dem Server, worauf aber erst später bei der Erklärung der Servlet–Beispiele eingegangen wird.

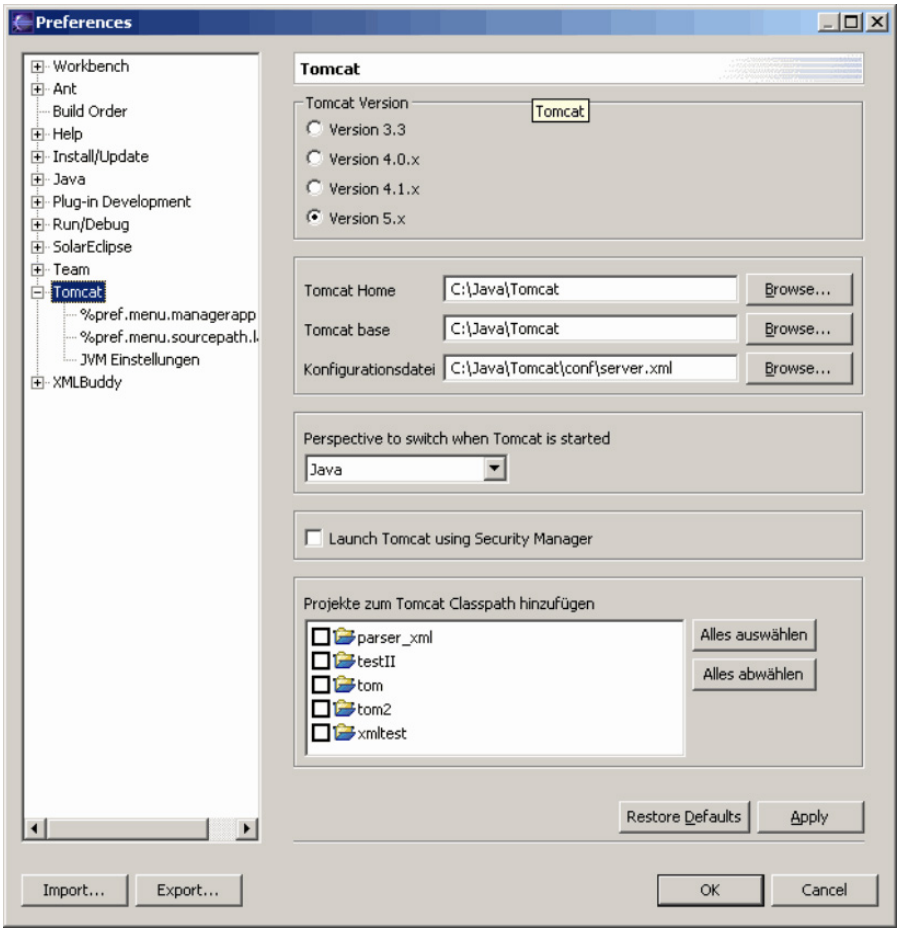


Abbildung 9.3.: Tomcat: Einstellungen des Tomcat–Plugin Version 221 für Eclipse

Abbildung 9.4.: Tomcat: Shortcuts unter Eclipse

9.1.3. Die Architektur des Tomcat: Catalina

Als sich die Tomcat 3.x–Archtitektur den neuen Anforderungen der Servlet 2.3 und JSP 1.2 Spezifikationen als nicht mehr gewachsen erwies, wurde eine komplett neue Architektur entworfen. Es sollte eine neue Architektur eines zeitgemäßen Servlet–Containers geschaffen werden. Ein Entwurfsziel der Catalina–Architektur ist es, eine vollständige Unterstützung der Servlet 2.3 und JSP 1.2 APIs zu liefern, die als Referenzanwendung von Herstellern, Internet–Providern, Entwicklern und Nutzern akzeptiert wird. Die internen Strukturen müssen dazu änderbar, wartbar und trotzdem für den produktiven Einsatz nutzbar sein.

Die Aufgabe der Basisarchitektur besteht in der Bereitstellung einer Ablaufumgebung, in welcher Servlets mit Anfragen versorgt werden.

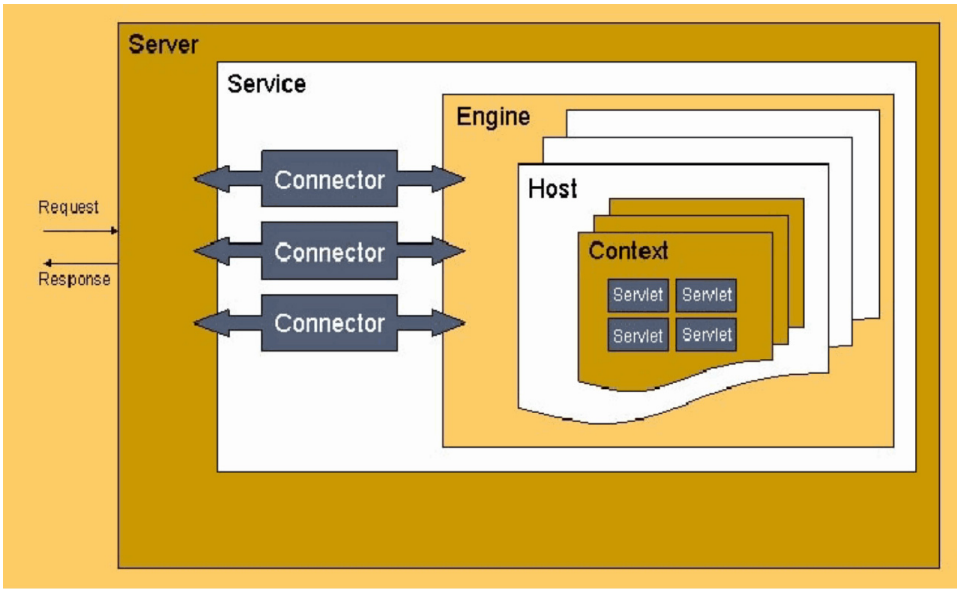


Abbildung 9.5.: Tomcat: Die Catalina–Architektur

Die Schaltzentrale des Tomcat ist eine Serverinstanz in einer JVM, die mehrere Services gleichzeitig verwaltet (s.o.). Innerhalb des Startvorgangs wird dieser Server mit Hilfe der „Bauanleitung“ `server.xml` erzeugt. Ein Service ist für die Dienste des Servers zuständig und enthält einen oder mehrere Connectoren. Die Connectoren werden in zwei Typen unterschieden. Zum einen in solche, die direkte Verbindungen mit dem Browser herstellen (z.B. der sog. Coyote–Connector für HTTP/1.1) und zum anderen in solche, die Verbindungen mit einem zwischengeschaltetem Web–Server wie Apache herstellen (AJP13–Connector). Der Service nutzt seine Connectoren um über verschiedenen Netzwerkprotokolle Anfragen entgegenzunehmen und um aus diesen ein „`Request/Response`“–Paar zu produzieren. Dieses Paar enthält alle Informationen, um an die richtige `service()`–Methode eines Servlets zu kommen. Der Connector bedient sich zur Interpretation des Protokolls und zur Umsetzung in das „`Request/Response`“–Paar in einem Pool von Verarbeitern, die als Prozessoren bezeichnet werden. Jeder Prozessor bearbeitet eine

Anfrage in einem eigenen Thread. Nach erfolgreicher Erzeugung des Paars „leitet“ der Prozessor die Anfrage an die Engine weiter. Diese Engine erhält die gesamten Anfragen eines Service und bildet somit das Herzstück der Verarbeitung. Sie hat die Aufgabe, den „Bearbeiter“ einer Anfrage zu finden.

Ein Serverrechner hat unter Umständen eine komplexe Netzwerkinfrastruktur mit mehreren Netzwerkkarten, (virtuellen) IP–Adressen oder Rechnernamen. Wenn der Tomcat auf einem Server eines Internet–Providers als Web–Plattform für mehrere Kunden dient, ist dies sehr wahrscheinlich. Jeder der verschiedenen Kunden möchte dann seine Webanwendungen mit den dazugehörigen Servlets, JSPs, Protokolldateien, Verzeichnissen, Nutzerrechten usw. bereitgestellt haben. Hierfür existiert die Schnittstelle Host, welche die nötige Infrastruktur zur Verfügung stellt. Innerhalb einer Engine können 1–n Hosts vorhanden sein. Es entsteht eine Konfiguration für jeden dieser „virtuellen“ Hosts. Der Host enthält eine Liste von Webanwendungen als Context.

Innerhalb der Catalina–Architektur sind Engine, Host und Context so genannte Container. Die Funktion eines Containers ist die Verarbeitung von eingehenden Anfragen (Requests) und die Erzeugung der Antworten (Response). Der Ablauf „innerhalb“ der Engine sieht folgendermaßen aus: Die Engine wählt bei einer Anfrage den zuständigen Host, dieser wählt den passenden Context. Der Context wählt das entsprechende Servlet und stellt die Anfrage zu. Das Servlet verarbeitet die Anfrage.

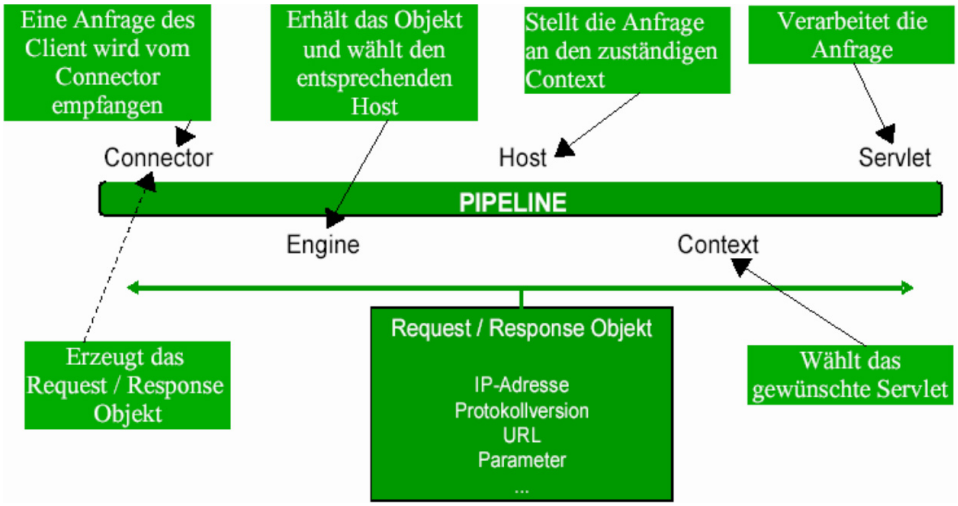


Abbildung 9.6.: Tomcat: Ablauf im Catalina–Container

Zusatzkomponenten

Innerhalb jedes Containers sind Zusatzkomponenten vorhanden, die Einfluss auf die Requestverarbeitung haben. Hier werden drei dieser Komponenten beschrieben, da diese in jedem Container verwendet werden können.

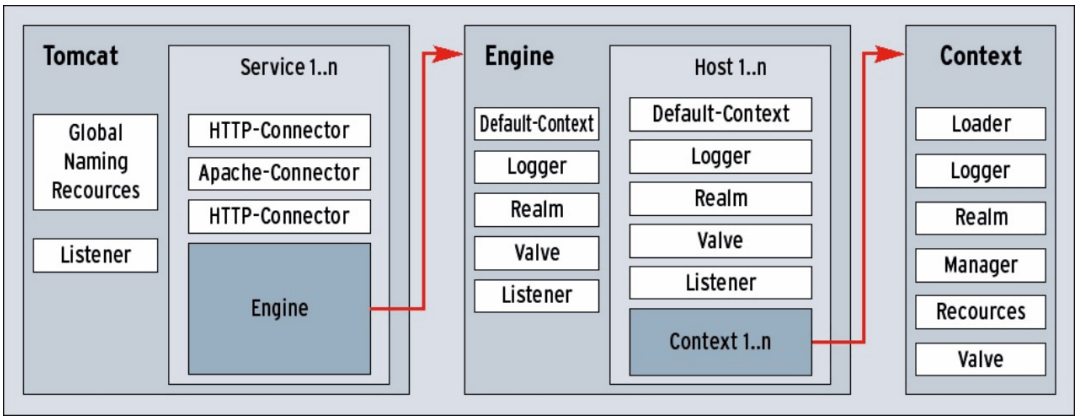


Abbildung 9.7.: Tomcat: Catalina–Architektur mit Zusatzkomponenten

Valve Durch das Konzept der Valves (Ventile) ist die Anfragebearbeitung hoch konfigurierbar. In jedem Schritt der Zustellung der Anfrage an ein Servlet kann eine Filterung und Steuerung erfolgen. Somit wird die weitere Vorgehensweise beeinflusst. Die Pipeline jedes Containers besitzt mindestens ein Valve (Basic Valve). Die Valve bekommt das „`Request/Response`“–Objekt übergeben und kann entscheiden ob die Anfrage analysiert, bearbeitet (verändert), beantwortet oder weitergeleitet werden soll. Valves werden nacheinander in der Pipeline abgearbeitet. Am Ende der Pipeline sorgt ein sog. „Mapper“ für die Weiterleitung des „`Request/Response`“–Objekts.

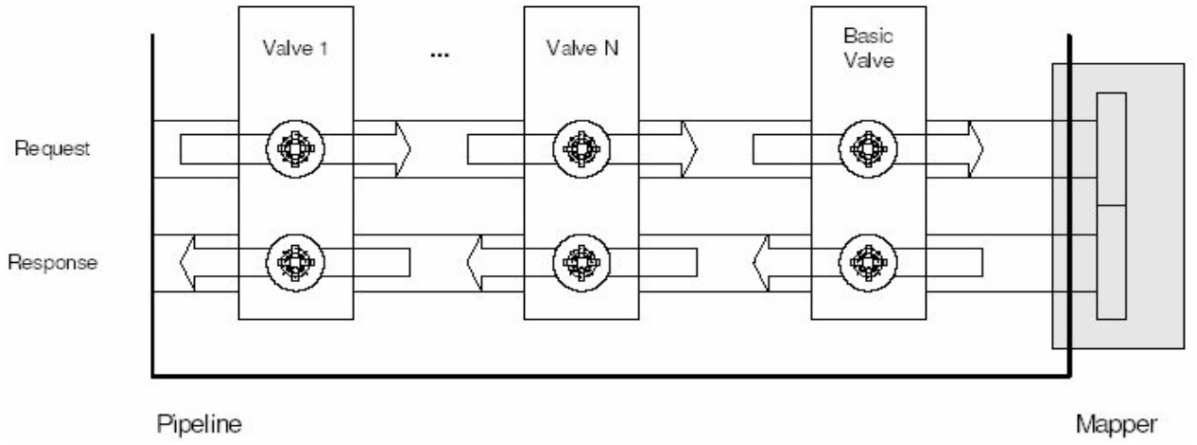


Abbildung 9.8.: Tomcat: Funktionsweise der Valves

Realm Die Sicherheit und Vertraulichkeit von Information im Internet ist von sehr hoher Bedeutung. Eine Web–Site muss vor Angriffen von außen und innen geschützt werden. Im Servlet API gibt es für jede Web–Ressource die Möglichkeit Autorisierungs– und Authentifikationsverfahren, sowie die Transportart festzulegen. Im Deployment–Descriptor `WEB-INF/web.xml` (s.u.) einer jeder Web–Applikation können die Einschränkungen eingetragen werden. Ein Realm hat die Aufgabe jede Anfrage zu authentifizieren.

Passwörter können dabei in Dateien, Datenbanken oder einem LDAP–Server hinterlegt werden.

Logger Als Logger steht ein einfaches Protokoll–API zur Verfügung. Es können Aktivitäten und Prozesse auf den fünf Standardstufen `FATAL`, `ERROR`, `WARNING`, `INFORMATION` und `DEBUG` protokolliert werden.

9.1.4. Die Verzeichnisstruktur des Tomcat

Die Verzeichnisstruktur ist unter Windows wie unter Linux gleichermaßen aufgebaut. Die wichtigsten Ordner sind unterstrichen hervorgehoben:

Ordner	**Bedeutung**
`/bin`	enthält u.a. die Dateien zum Starten und Beenden von Tomcat
`/common`	enthält Libraries (`/lib`),Parser (`/endorsed`), ausgepackte Klassen
`/conf`	enthält die wichtigsten Konfigurationsdateien
`/logs`	enthält Log–Dateien
`/server`	enthält die Tomcat Java Archive
`/shared`	enthält bereitgestellte Dateien
`/src`	Quellcode des Tomcat Servers
`/temp`	Verzeichnis für temporäre Dateien
`/webapps`	ist das Basisverzeichnis für die Web–Applikationen
`/work`	Arbeitsverzeichnis für Tomcat (JSP's, Servlets)

Die Verzeichnisstruktur einer Webapplikation:

Ordner	**Bedeutung**
`/WEB-INF`	enthält Deployment Descriptor web.xml; muss vorhanden sein; ist vom Client aus nicht zu erreichen
`/WEB-INF/classes`	enthält Servlet– und Hilfsklassen
`/WEB-INF/lib`	enthält Bibliotheken
Sonstige (z.B. Images, JSP)	vom Client aus zu erreichen

9.1.5. Konfiguration

Alle Konfigurationsdateien des Tomcat sind im XML–Format gehalten. Sie werden in globale und lokale Konfigurationsdateien unterschieden. Die globalen Konfigurationsdateien befinden sich im Verzeichnis `/conf`. Die `server.xml` ist die Hauptkonfigurationsdatei des Tomcat–Servers. Die Datei `web.xml` ist zuständig für globale Einstellungen von Web–Applikationen. Im `/WEB-INF`–Verzeichnis einer jeden Webanwendung befindet sich

die lokale `web.xml` für die anwendungsspezifische Konfiguration der jeweiligen Web–Applikation. Im Verzeichnis `/conf` ist auch die Datei `tomcat-users.xml` vorhanden, welche für die Definition von Usern und Rollen zuständig ist.

tomcat-users.xml

Die Datei `tomcat-users.xml` enthält die Benutzerdaten der Tomcat–Anwender. Der Server erlaubt es auch diese Benutzerdaten in einer Datenbank oder auf einem LDAP–Server zu hinterlegen.

```
1 <tomcat-users>
2   <role rolename="manager"/>
3   <role rolename="admin"/>
4   <role rolename="tomcat"/>
5   <user username="manager" password="manager" roles="manager"/>
6   <user username="admin" password="admin" roles="admin,manager"/>
7   <user username="tester" password="test" roles="tomcat"/>
8 </tomcat-users>
```

Quelltext 9.2: Tomcat: Anwender in tomcat-users.xml

Die Rollen stellen die Zugriffsrechte dar. Manche Dateien und Webanwendungen auf dem Server dürfen nur von Usern mit bestimmtem Rollen verwendet werden. Die Einstellungen hierfür werden in der `web.xml` der jeweiligen Webanwendung vorgenommen (siehe `web.xml`).

Server.xml

Die `server.xml` spiegelt die behandelte Catalina–Architektur wieder. Die minimale Struktur sieht folgendermaßen aus:

```
- <Server>
    <!-- einmal -->
  - <Service>
      <!-- ein oder mehrmals -->
      <Connector/>
      <!-- ein oder mehrmals -->
    - <Engine>
        <!-- genau einmal -->
      - <Host>
          <!-- ein oder mehrmals -->
          <Context/>
          <!-- beliebig -->
        </Host>
      </Engine>
    </Service>
  </Server>
```

Abbildung 9.9.: Tomcat: Minimale Struktur der server.xml

In der maximalen Struktur kommen noch alle Komponenten, die in den jeweiligen Bereichen existieren dürfen, hinzu. In der Engine und im Host sind das die behandelten Komponenten Valve, Realm und Logger. Im Context sind es zusätzlich zu diesen noch die Komponenten Loader, Manager und Ressources.

Beispiel für `server.xml`: Ändern des Ports für den HTTP–Connector:

```
<!--Define a non-SSL Coyote HTTP/1.1 Connector on the port specified during installation
-->
<Connector port="8080" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  enableLookups="false" redirectPort="8443" acceptCount="100" debug="0"
  connectionTimeout="20000" disableUploadTimeout="true" />
```

Quelltext 9.3: Tomcat: server.xml

In diesem Abschnitt der `server.xml` ändert man einfach „`8080`“ in „`80`“ um und schon horcht der HTTP/1.1–Connector auf dem „well–known–port“ `80` für das HTTP–Protokoll.
Der SSL–Connector ist standardmäßig auskommentiert. Durch entfernen XML–Kommentare und Neustart des Servers wird dieser aktiviert. Zuvor sollte allerdings mit „`<JAVA_HOME>\bin\keytool.exe`“ ein Zertifikat erzeugt werden.

web.xml

In der `web.xml` sind die globalen Einstellungen für alle Web–Applikationen des Servers enthalten. Weitere Dateien des gleichen Namens befinden sich in den verschiedenen `/WEB-INF`–Verzeichnissen der Webanwendungen. Diese enthalten dann die Einstellungen für die jeweiligen Anwendungen.

Struktur einer `web.xml`:

```
- <web-app>
    <filter/>
    <!-- Definition von Filtern -->
    <servlet/>
    <!-- Definition eines Servlets -->
    <servlet-mapping/>
    <!-- Ein Servlet auf eine URL "mappen" -->
    <session-config/>
    <!-- Timeout einer Session setzen -->
    <welcome-file-list/>
  - <!--
      Dateien, die automatisch geladen werden, wenn existent
    -->
    <taglib/>
    <!-- Definition einer Tag-Library -->
    <security-constraint/>
    <!-- Definition eines Security-Constraint -->
    <login-config/>
    <!-- Festlegen des Authentifikationsverfahrens -->
  </web-app>
```

Abbildung 9.10.: Tomcat: Struktur der web.xml

Die aufgeführten Bereiche der `web.xml` sind nicht alle zwingend notwendig und sie können auch mehrfach auftreten, wie z.B. `<servlet/>` oder `<filter/>`. Im Folgenden werden die einzelnen Bereiche kurz beschrieben.

- `<filter>`
  Hier können Filterklassen angegeben werden, welche in der Lage sind den Inhalt und die Header von `Request/Response`–Objekten zu analysieren und zu manipulieren. Dies geschieht entweder bevor ein Request das Servlet erreicht bzw. nach der Bearbeitung durch das Servlet (Response).

```
1 <filter>
2   <filter-name>Request Dumper Filter</filter-name>
3   <filter-class>filters.RequestDumperFilter</filter-class>
4 </filter>
```

- `<servlet>`
  Dieser Eintrag ist notwendig um ein Servlet zu einer Webanwendung hinzuzufügen.

```
1 <servlet>
2   <servlet-name>CalcServlet</servlet-name>
3   <servlet-class>testpack.CalcServlet</servlet-class>
4 </servlet>
```

- `<servlet-mapping>`
  Das Tag <servlet-mapping> ist dafür zuständig einem Servlet ein URL–Pattern zuzuordnen.

```
1 <servlet-mapping>
2   <servlet-name>CalcServlet</servlet-name>
3   <url-pattern>servlet/CalcServlet</url-pattern>
4 </servlet-mapping>
```

  Das Servlet ist jetzt unter:
  `http://localhost/<Verzeichnis der Webanwendung>/servlet/CalcServlet`
  zu erreichen. Zu beachten ist hierbei das `/servlet` kein Verzeichnis darstellt; es ist lediglich durch das URL–Pattern definiert.

- `<session-config>`
  In diesem Bereich wird eingestellt, wie lange (in Minuten) eine Session am Leben gehalten werden soll.

```
1 <session-config>
2   <session-timeout>30</session-timeout>
3 </session-config>
```

- `<welcome-file-list>`
  Wenn über einen Link ein Verzeichnis auf dem Server angesprochen wird, prüft das Default–Servlet, ob in diesem Verzeichnis sog. „welcome–files“ vorhanden sind. Falls eines vorhanden ist, wird dieses im Browser automatisch geladen und dargestellt. Ist keines vorhanden, wird entweder der Inhalt des Verzeichnisses angezeigt

oder der Fehlerstatus „404“ ausgegeben. Die Welcome–Files sind in der globalen `web.xml` definiert. Falls sie in einer lokalen `web.xml` gesetzt werden, überschreiben diese die Welcome–Files der globalen `web.xml`.

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

Unter http://localhost wird das Verzeichnis `/Tomcat/webapps/ROOT` angesprochen. In diesem befindet sich standardmäßig die `index.jsp` für die Startseite des Tomcat. Da `index.jsp` zu den Welcome–Files gehört und keine `index.html` oder `index.htm` im Verzeichnis vorhanden ist, wird sie automatisch geladen.

- `<taglib>`
  Um eine Tag–Library in einer Webanwendung zu verwenden, muss sie folgendermaßen definiert werden.

```
<taglib>
  <taglib-uri>
    http://jakarta.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
```

- `<security-constraint>`
  Durch das Security–Constraint werden Ressourcen/Bereiche innerhalb einer Webanwendung vor unbefugten Zugriffen geschützt. Diese werden im folgenden Tag angegeben: `<web-resource-collection>`. Außerdem werden die erlaubten HTTP-Methoden definiert. Im unteren Beispiel werden unter dem Namen „Protected Area“ alle Dateien im Verzeichnis `/protected` der Webanwendung geschützt und es ist nur HTTP–GET erlaubt. Ist die Webanwendung die ROOT–Applikation des Tomcat wäre somit http://localhost/protected geschützt.
  Innerhalb von `<auth-constraint>` werden die Rollen festgelegt, welche auf den geschützten Bereich zugreifen dürfen. Im Beispiel sind es nur Benutzer mit der Rolle „admin“.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/protected/*</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Außerhalb des Security–Constraint sollten im Tag `<security-role>` alle Rollen angegeben werden, die Zugriff auf die Webanwendung haben sollen.

```
<security-role>
  <role-name>admin</role-name>
</security-role>
```

- `<login-config>`
  Ohne `<login-config>` wären alle Angaben im Security–Constraint überflüssig, denn hier wird das Authentifikationsverfahren für den Zugriff auf die geschützten Bereiche einer Webanwendung festgelegt. Mögliche Methoden sind `BASIC`, `FORM`, `DIGEST` und `CLIENT-CERT`.

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Geschützter Bereich</realm-name>
</login-config>
```

  `BASIC` ist die standardmäßige HTTP–Authentifizierung auf der Basis von Benutzername und Passwort.

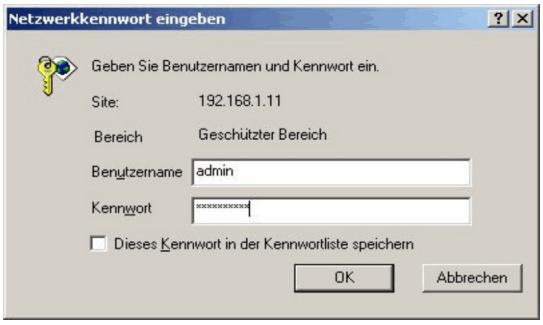


Abbildung 9.11.: Tomcat: HTTP BASIC–Authentifizierung für geschützten Bereich

  `FORM` erlaubt die `BASIC`–Authentifizierung über ein benutzerdefiniertes Login–Formular, welches zusammen mit einer Fehlerseite angegeben werden muss:

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Geschützter Bereich</realm-name>
  <form-login-config>
    <form-login-page>/protected/login.jsp</form-login-page>
    <form-error-page>/protected/error.jsp</form-error-page>
  </form-login-config>
</login-config>
```

  Bei `DIGEST` wird das Passwort verschlüsselt übergeben. `CLIENT-CERT` benötigt das HTTPS Protokoll (SSL–Verbindung) und die Daten werden mit einem Public–Key Verfahren verschlüsselt. Dies ist die sicherste Methode, aber sie benötigt eine Zertifizierung von einer autorisierten Stelle. Ausreichende Sicherheit bietet auch die Kombination von `BASIC`–Authentifizierung mit SSL.

9.2. Exkurs: http – Hyper Text Transfer Protokoll

Um den Ablauf der Kommunikation zwischen einem Servlet des Tomcat–Servers und einem Web–Browser besser zu verstehen, wird an dieser Stelle kurz auf das HTTP–Protokoll eingegangen. HTTP ist das Protokoll, mit welchem Daten zwischen einem Web–Client und einem Web–Server ausgetauscht werden. Auf beiden Seiten des Kommunikationsweges existiert eine TCP–Instanz, zwischen denen eine Verbindung aufgebaut wird. Zum Verbindungsaufbau zum Web–Server muss dessen Name bzw. IP–Adresse, sowie der TCP–Port des Serverprozesses bekannt sein. Der sog. „Well–Known–Port“ für HTTP ist `80`. Dadurch kann auf die Angabe des Ports bei der Adresseingabe meistens verzichtet werden. „Horcht“ der Serverprozess allerdings auf einem anderen Port, z.B. der häufig verwendete Port `8080`, muss dieser mit angegeben werden.

Beispiele:

- http://192.168.1.11/calculator/servlet/CalcServlet
- http://localhost:8080/test/servlet/TestSrv
- http://bizweb.hs-bremerhaven.de

HTTP verwendet die Dienste von TCP (Transmission Control Protocol) zur eigentlichen Dateiübertragung. Für die Übertragung eines Web–Dokuments wird eine TCP–Verbindung aufgebaut, über welche die Protokollnachrichten von HTTP geschickt werden. Diese Protokollnachrichten werden wiederum von geeigneter Software – auf der Client–Seite meistens Web–Browser und auf der Server–Seite Web–Server – genutzt. Das HTTP–Protokoll ist ein Request–Response–Protokoll. Mit Hilfe eines HTTP–Request fordert der Client eine bestimmte Ressource (z.B. ein HTML–Dokument) vom Server an. In einem HTTP–Response schickt der Server die angeforderte Ressource an den Client

9.2.1. HTTP–Request und HTTP–Methoden

In einem HTTP–Request–Paket wird wie oben beschrieben eine Anfrage eines Clients verschickt. Das Format eines Request–Pakets ist dabei genau definiert. Der Header eines HTTP–Request–Pakets weist folgende Struktur auf:

```
METHOD URL HTTP/version
General Header
Request Headers
Entity Header (optional)
Leerzeile
Request Entity (falls vorhanden)
```

Abbildung 9.12.: Tomcat: Struktur des HTTP–Request–Headers

Ein Beispiel für den Header eines Request–Pakets ist im Folgenden angegeben.

```
⊟ Hypertext Transfer Protocol
  ⊟ GET /calculator/servlet/CalcServlet?zahl1=5&operator=%2F&zahl2=0 HTTP/1.1\r\n
      Request Method: GET
    Host: 192.168.1.11\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7) Gecko/20040614 Firefox/0.9\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Referer: http://192.168.1.11/calculator/servlet/CalcServlet\r\n
    \r\n
```

Abbildung 9.13.: Tomcat: Beispiel eines HTTP–Request–Headers

In der ersten Zeile des Pakets steht die eigentliche Anfrage. Hier ist die HTTP–Methode, die angewendet werden soll, sowie die Ressource auf welche sie sich bezieht, angegeben. Hier wird auch die zur Anfrage verwendete HTTP–Version vermerkt. Danach folgen verschiedene Header in welchen Informationen über die Anfrage übertragen werden. Der `Body` eines Requests kann Parameter der Anfrage enthalten. Im obigen Beispiel wird die `GET`–Methode auf die Ressource

`/calculator/servlet/CalcServlet?zahl1=5&operator=%2F&zahl2=0`

ausgeführt. Hierbei handelt es sich um das Servlet „CalcServlet“, welches über die URL mit Parametern versorgt wird (`?zahl1=5&operator=%2F&zahl2=0`). Das Taschenrechner–Servlet bekommt den Auftrag `5/0` zu berechnen. In den Headern finden sich u.a. Informationen über den Browser, der für die Anfrage verwendet wurde.

HTTP verfügt insgesamt über 8 Methoden, welche in einer Anfrage ausgeführt werden können. Die wichtigsten sind `GET` und `POST`.

- `GET` ist die wohl am häufigsten verwendete Methode. Sie wird verwendet um die angegebene Ressource auf den Client–Rechner zu laden.
  (z.B. http://www.nordit.de/start.html)
  Es ist aber auch möglich Informationen an den Web–Server zu übertragen, wie z.B. die Parameter für den Aufruf eines Servlets. Dieses geschieht innerhalb der URL, welche die Ressource aufruft (s.u.).

- `POST` ist die Standardmethode um Informationen an den Server zu übermitteln (z.B. Formulardaten). Im HTTP–POST–Paket identifiziert die Ressource ein Programm, welches in der Lage ist, Informationen vom Client entgegenzunehmen. Die Informationen selbst werden im Body der Nachricht codiert.

- `HEAD` wird verwendet um lediglich den HTML–Header einer Ressource auszulesen. Dies ist nützlich, um z.B. die Erreichbarkeit von URLs zu testen, ohne das die unter Umständen sehr umfangreiche Seite geladen werden muss.

- `PUT` ermöglicht, unter Vorraussetzung der erforderlichen Berechtigung, eine neue Ressourcen auf einem Web–Server anzulegen.

- `DELETE` ist zum Löschen von Ressourcen vorgesehen.
- `TRACE` wird für diagnostische Zwecke eingesetzt. `OPTIONS` befähigt den Client etwas über die Fähigkeiten des Web–Servers herauszufinden ohne eine Ressource zu laden und `CONNECT` ist eine reservierte Methode.

9.2.2. HTTP–Response

Das Gegenstück zum HTTP–Request ist das HTTP–Response. Auch hier ist die Paketstruktur genau definiert:

```
HTTP/version Status-Code Reason-Zeile
General Header
Response Header
Entity Header (optional)
Leerzeile
Resource Entity (falls vorhanden)
```

Abbildung 9.14.: Tomcat: Struktur eines HTTP–Request–Pakets

Das Response–Paket welches als Antwort auf die obige Beispiel–Anfrage an den Client geschickt wurde sieht folgendermaßen aus:

```
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
      Response Code: 200
    Content-Type: text/html\r\n
    Content-Length: 569\r\n
    Date: Mon, 20 Sep 2004 13:45:41 GMT\r\n
    Server: Apache-Coyote/1.1\r\n
    \r\n
⊟ Line-based text data: text/html
    <html><head><title>Servlet-Taschenrechner</title></head>
    <body bgcolor='#0E115E'><center>
    <H1><font color='#FFFFFF'>Servlet-Taschenrechner</font></H1>
    <form method='get' action=''>
    <br><font color='#FFFFFF'>Zahl1</br>
    <input type='text' name='zahl1' size='20'>
    <br></br>
    OP
    <br><input type='text' name='operator' size='1'></br>
    <br>Zahl2</br>
    <input type='text' name='zahl2' size='20'>
    <br></br></font>
    <input type='submit' value='Ergebnis berechnen'>
    </form>
    <big><blink><font color='#FFFFFF'>Division durch Null!</blink></big>
    </center></body></html>
```

Abbildung 9.15.: Tomcat: Beispiel eines HTTP–Response–Pakets

In der ersten Zeile der Antwort befindet sich die tatsächlich verwendete HTTP–Version, eine Statusmeldung des Servers, sowie deren Erläuterung. Anschließend folgen wiederum die Header mit verschiedenen Informationen und der Body, in welchem sich in den meisten Fällen die angefragten Daten befinden. Im obigen Beispiel wird HTTP/1.1 verwendet und der Status ist 200 („`OK`“). Es hat alles funktioniert und im Body befindet sich der vom Servlet generierte HTML–Code, der nun im Browser dargestellt werden kann.

9.3. Java Servlets

9.3.1. Was sind Servlets?

Servlets sind vereinfacht ausgedrückt serverseitige Java–Programme, welche HTML–Code liefern bzw. generieren. Der Vorteil besteht darin, dass somit nur der Server entsprechend konfiguriert werden muss und auf Client–Seite kein Browser vorhanden sein muss, der Applets unterstützt bzw. eine JVM benötigt. Im Grunde sind sich Servlets und CGI–Skripts sehr ähnlich, beide generieren im Wesentlichen eine HTML–Seite. Die Technologien im Hintergrund unterscheiden sich jedoch deutlich.

Ein Vorteil von Servlets gegenüber CGI–Programmen ist, dass diese wesentlich schneller in der Ausführung sind. Servlets laufen nicht als eigener Prozess, sondern die Servlet–Engine (in unserem Fall der Tomcat) eines Web–Servers startet lediglich einen neuen Thread. Ist das Programm einmal aufgerufen, so bleibt es im Speicher. Bei CGI–Skripts hingegen muss im Fall der am häufigsten anzutreffenden Skriptsprache Perl bei jedem Request erneut der Interpreter gestartet werden, was zu erheblichen Performance–Einbußen führt. Außerdem sind dadurch, dass ein Servlet im Speicher gehalten wird, wieder verwendbare (persistente) Objekte möglich. Das ist zum Beispiel für Applikationen die auf eine Datenbank zugreifen vorteilhaft, da die Verbindung zur Datenbank nur ein einziges Mal hergestellt werden muss.

Ein Nachteil von Servlets ist, dass man an die Programmiersprache Java gebunden ist, was bei CGI nicht der Fall ist.

9.3.2. Lebenszyklus eines Servlets

Der Tomcat–Server (oder eine andere Servlet–Engine) ruft während des Startvorganges die Methode `init(ServletConfig)` des Servlets auf und initialisiert es dadurch. Das Herzstück eines jeden Servlets ist die `service()`–Methode. Diese bekommt die Objekte `ServletRequest` und `ServletResponse` übergeben und wird bei jeder Anfrage an das Servlet ausgeführt. Wird der Tomcat–Server beendet oder will dieser das Servlet stoppen, wird die `destroy()`–Methode des Servlets aufgerufen.

Sämtliche dieser Methoden können überschrieben werden. Dieses bietet sich bei `init()` und `destroy()` an, um z.B. eine Datenbankverbindung zu öffnen bzw. zu beenden oder um Objekte zu laden bzw. zu speichern. Die `service()`–Methode sollte bei HttpServlets allerdings nicht überschrieben werden, da diese die Grundlage der Kommunikation darstellt (s.u.).

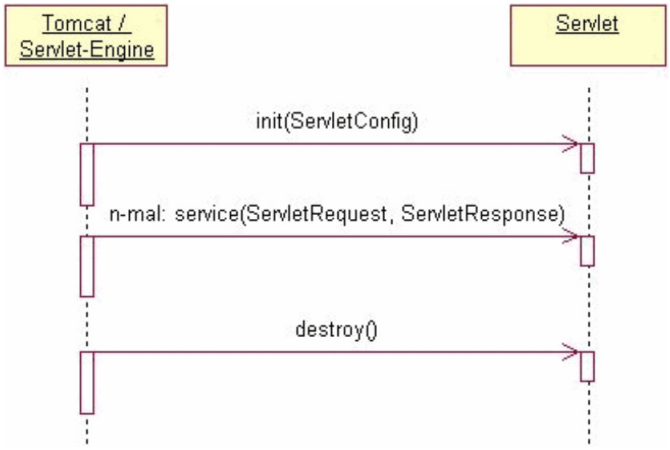


Abbildung 9.16.: JAVA Servlets: Lebenszyklus eines Servlets

9.3.3. Allgemeiner Ablauf

Die Kommunikation beginnt damit, dass ein Browser eine Anfrage (`Request`) an eine Servlet–URL sendet. In der `service()`–Methode wird die vom Client verwendete HTTP–Methode festgestellt und entsprechend die passende Methode des Servlets aufgerufen. Standardmäßig werden die HTTP–Methoden `GET`, `POST`, `HEAD`, `PUT`, `DELETE`, `OPTIONS` und `TRACE` unterstützt. Die zuständigen Servlet–Methoden heißen genauso wie die HTTP–Methoden, bekommen jedoch ein „`do`“ vorangestellt (also `doGet()`, `doPost()`, `doPut()`, usw.). Die Objekte, welche diesen „`do`–Methoden“ übergeben werden heißen HttpServletRequest, in welchem der Request des Browsers verborgen ist und HttpServletResponse, welches genutzt wird um z.B. generierten HTML–Code an den Browser zurückzuschicken oder um Daten an eine andere Stelle weiterzuleiten. Beispielweise wird ein Request mit `GET` verschickt, was dazu führt, dass die `doGet()`–Methode ausgeführt wird, welche dann den HTML–Code generiert. Dieser HTML–Code wird dann im Response–Paket als Antwort an den Browser zurückgeschickt.

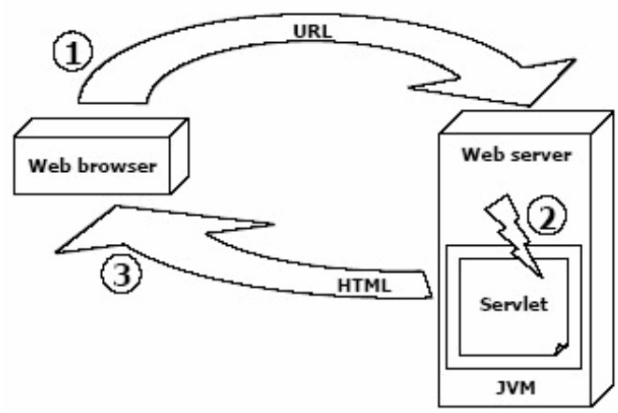


Abbildung 9.17.: JAVA Servlets: Allgemeiner Ablauf

9.3.4. Servlet–Beispiele

In diesem Abschnitt werden zwei beispielhafte Servlets vorgestellt. Sie wurden mit Hilfe des Tomcat–Plugins für Eclipse geschrieben und als Webanwendung auf dem Tomcat–Server installiert. Diese Vorgehensweise wird nun beschrieben.

Zuerst wird ein neues Tomcat–Projekt, z.B. mit dem Namen „Servlet_Beispiele“, erstellt, welches die beiden Beispiele aufnehmen soll. In diesem Projekt finden wir eine ähnliche Verzeichnisstruktur vor, wie sie auch innerhalb einer Webanwendung im Ordner `%TOMCAT_HOME%\webapps` sein sollte. Außerdem sind natürlich die Libraries für die Servlet– und JSP–APIs vorhanden. Im Ordner `WEB-INF/src` wird ein neues Package erstellt, z.B. mit dem Namen „testpack“. In diesem Package sollen die beiden Beispiel–Klassen angelegt werden.

Hello World

Das erste Beispiel ist natürlich ein „Hello World“–Servlet. Servlets sind Java–Programme in Form einer Klasse, die von `javax.servlet.http.HttpServlet` abgeleitet sind. Daher müssen in der neuen Klasse (in diesem Fall „TestSrv“) zuerst `javax.servlet.*` und `javax.servlet.http.*` importiert werden. Außerdem wird `java.io.*` für die Ausgabe des HTML–Codes benötigt. Um ein `GET`–Request zu handhaben muss die `doGet()`–Methode überschrieben werden. Diese bekommt, wie oben beschrieben, die benötigten Objekte übergeben und muss eine `ServletException`, sowie eine `IOException` abfangen.

In dieser Methode wird dann mit „`response.setContentType(''text/html'');`“ das Ausgabeformat auf HTML gesetzt. Der PrintWriter out ist dafür zuständig, dass der HTML–Code mit „`out.println(('<html>...</html>'');`“ verschickt werden kann.

```
1  package testpack;
2
3  import java.io.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class TestSrv extends HttpServlet
8  {
9    public void doGet(HttpServletRequest request,HttpServletResponse response)
10   throws ServletException, IOException
11   {
12     PrintWriter out = response.getWriter();
13
14     out.println("<html><head><title>BizWeb - Servlet Test");
15     out.println("</title></head><body>");
16     out.println("Hallo Welt !!!");
17     out.println("</body></html>");
18   }
19 }
```

Quelltext 9.4: JAVA Servlets: TestSrv.java

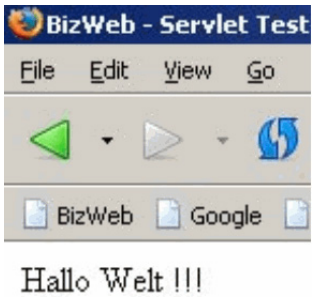


Abbildung 9.18.: JAVA Servlets: Ausgabe des Hello World–Servlets

Servlet–Taschenrechner

Dieses etwas größere Beispiel soll unter anderem den Umgang mit Übergabe–Parametern erläutern. In diesem Fall werden die Parameter per `HTTP-GET` in der URL übertragen. Sie werden in folgender Form an das Servlet übergeben:

- http:// ... /servletname?parameter1=wert1&parameter2=wert2

Es können zwar theoretisch beliebig viele Parameter jeweils durch „Ampersand“ getrennt angegeben werden, jedoch werden diese als String über eine Umgebungsvariable an das Servlet übergeben. Diese Umgebungsvariable wird vom System auf eine bestimmte Länge beschränkt. Alles was über diese Länge hinausgeht wird abgeschnitten. Aus diesem Grunde wird bei Anwendungen, bei denen eine größere Datenmenge zu übergeben ist (wie z.B. der Inhalt eines Anmeldeformulars) oder die Größe nicht vorherbestimmt werden kann, `HTTP-POST` verwendet. Bei dieser Methode werden die Parameter im `HTTP-Request`–Paket übergeben und nicht in der URL angezeigt, was natürlich auch ein Vorteil ist.

In der Klasse `CalcServlet` wird ein kleiner Taschenrechner implementiert. Für Eingabezahlen und das Ergebnis werden drei `float`–Variablen angelegt. Beim ersten Aufruf des Servlets wird ein kleines HTML–Formular angezeigt:

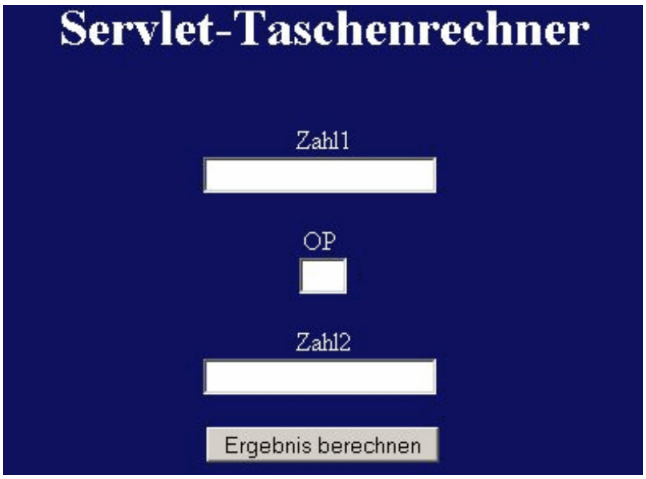


Abbildung 9.19.: JAVA Servlets: Ausgabe Servlet–Taschenrechner

Der Name des ersten Eingabefeldes ist „zahl1“, der des zweiten „operator“ und der des dritten „zahl2“. Diese Namen gelten als Bezeichnung für die Parameter, welche an das Servlet übergeben werden. Dies geschieht wenn auf den Button „Ergebnis berechnen“ geklickt wird, da jetzt das Formular ein `HTTP-GET-Request` absetzt (`<form method='get'...`). Mit `request.getParameter()` werden die Parameter in einem String gespeichert (z.B. `String z1 = request.getParameter('zahl1');`).

Das Servlet berechnet nun in Abhängigkeit dieser Variablen das Ergebnis und schickt es an den Browser.:

Abbildung 9.20.: JAVA Servlets: Ergebnisausgabe

Die `doPost()`–Methode am Ende der Klasse ruft einfach die `doGet()`–Methode auf. Dies passiert wenn ein Request mit `HTTP-POST` verschickt wurde. In diesem Beispiel ist es dazu nützlich sich die unterschiedlicher Verfahrensweise bei der Parameter–Übergabe zu verdeutlichen. Ändert man bei „`out.println('<form method='get' action=''>');`“ „`get`“ in „`post`“ um, so werden die Parameter per `HTTP-POST` übertragen und nicht mehr in der URL angezeigt.

```
1  /*
2   * Created on 06.05.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  package testpack;
8
9  /**
10  * @author DOMiNiON
11  *
12  * To change the template for this generated type comment go to
13  * Window - Preferences - Java - Code Generation - Code and Comments
```

```
14  */
15 import java.io.*;
16 import javax.servlet.*;
17 import javax.servlet.http.*;
18
19 public class CalcServlet extends HttpServlet
20 {
21
22   public void doGet(HttpServletRequest request, HttpServletResponse response)
23   throws ServletException, IOException
24   {
25     response.setContentType("text/html");
26     PrintWriter out = response.getWriter();
27
28     float num1=0, num2=0, result=0;
29     boolean b = true;
30
31       String z1 = request.getParameter("zahl1");
32     String op = request.getParameter("operator");
33     String z2 = request.getParameter("zahl2");
34
35       out.println("<html><head><title>Servlet-Taschenrechner</title></head>");
36       out.println("<body bgcolor=#0E115E><center>");
37       out.println("<H1><font color='#FFFFFF'>Servlet-Taschenrechner</font></H1>");
38     out.println("<form method='get' action=''>");
39     out.println("<br><font color='#FFFFFF'>Zahl1</br>");
40     out.println("<input type='text' name='zahl1' size='20'>");
41     out.println("<br></br>");
42     out.println("OP");
43     out.println("<br><input type='text' name='operator' size='1'></br>");
44     out.println("<br>Zahl2</br>");
45     out.println("<input type='text' name='zahl2' size='20'>");
46     out.println("<br></br></font>");
47     out.println("<input type='submit' value='Ergebnis berechnen'>");
48     out.println("</form>");
49
50     if(z1 !=null && op !=null && z2 !=null)
51     {
52       try
53       {
54         num1 = Float.parseFloat(z1);
55         num2 = Float.parseFloat(z2);
56       }
57       catch(NumberFormatException ne)
58       {
59         out.println("<font color='#FFFFFF'>Bitte nur Zahlen und einen ");
60         out.println("Punkt statt eines Kommas bei Dezimalstellen.</font>");
61         b = false;
62       }
63
64       if (op.equals("+")||op.equals("-")||op.equals("*")||op.equals("/")&& b)
65       {
66         try
67         {
68
69           if(op.equals("+"))
70           {
71             result = num1+num2;
72             out.println("<big><font color='#FFFFFF'>Ergebnis: "+result+"</font></big>");
73           }
74           else if(op.equals("-"))
75           {
76             result = num1-num2;
77             out.println("<big><font color='#FFFFFF'>Ergebnis: "+result+"</font></big>");
78           }
79           else if(op.equals("*"))
```

```
80            {
81              result = num1*num2;
82              out.println("<big><font color='#FFFFFF'>Ergebnis: "+result+"</font></big>");
83            }
84            else if(op.equals("/") && z2.equals("0"))
85            {
86              out.println("<big><blink><font color='#FFFFFF'>Division durch Null!</blink>
87                  </big>");
88            }
89            else if(op.equals("/"))
90            {
91              result = num1/num2;
92              out.println("<big><font color='#FFFFFF'>Ergebnis: "+result+"</font></big>");
93            }
94            else
95            {
96            }
97
98          }
99          catch(ArithmeticException ae)
100         {
101          out.println("<big>Fehler</big>");
102         }
103       }
104       else
105       {
106         out.println("<font color='#FFFFFF'>Gültige Operatoren: +, -, *, / .</font>");
107       }
108
109     }
110     out.println("</center></body></html>");
111   }
112
113   public void doPost(HttpServletRequest request, HttpServletResponse response)
114   throws ServletException, IOException
115   {
116       doGet(request,response);
117   }
118
119 }
```

Quelltext 9.5: JAVA Servlets: CalcServlet.java

9.3.5. Deployment auf dem Tomcat–Server

Um die Servlets im Browser aufzurufen, müssen sie zunächst auf dem Tomcat–Server installiert werden. Dies geschieht immer mit der Hilfe eines sog. Deployment–Descriptors: der aus dem Tomcat–Abschnitt bekannten `web.xml`.

Im Tomcat–Projekt „Servlet_Beispiele“ wird nun im Ordner `WEB-INF` die Datei `web.xml` angelegt und folgender Inhalt hineingeschrieben:

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4 <web-app>
5   <display-name>Beispiel-Servlets</display-name>
6
7   <servlet>
8     <servlet-name>TestSrv</servlet-name>
```

```
9       <servlet-class>testpack.TestSrv</servlet-class>
10    </servlet>
11
12    <servlet>
13      <servlet-name>CalcServlet</servlet-name>
14      <servlet-class>testpack.CalcServlet</servlet-class>
15    </servlet>
16
17    <servlet-mapping>
18      <servlet-name>TestSrv</servlet-name>
19      <url-pattern>servlet/TestSrv</url-pattern>
20    </servlet-mapping>
21
22    <servlet-mapping>
23      <servlet-name>CalcServlet</servlet-name>
24      <url-pattern>servlet/CalcServlet</url-pattern>
25    </servlet-mapping>
26
27  </web-app>
```

Quelltext 9.6: JAVA Servlets: web.xml

Es gibt zwei Möglichkeiten um mit dem Tomcat–Plugin die Webanwendung dem Tomcat–Server bekannt zu machen:

Kontext in der server.xml anlegen

Läuft der Tomcat zusammen mit Eclipse auf einem Rechner, kann der Kontext des Projekts in die `server.xml` des Tomcats eingetragen werden. Dies geschieht per Rechtsklick auf das Projekt → „Tomcat Projekt“ → „Kontext in Tomcat aktualisieren“. Folgender Eintrag wird dadurch von Eclipse am Ende der `server.xml` vorgenommen:

```
1 <Context path="/Servlet_Beispiele" reloadable="true"
2   docBase="D:\Entwurf\eclipse\workspace\Servlet_Beispiele"
3   workDir="D:\Entwurf\eclipse\workspace\Servlet_Beispiele\work" />
```

Quelltext 9.7: JAVA Servlets: server.xml

Somit sind die Pfade in dem die benötigten Dateien liegen dem Tomcat bekannt und er kann die Webanwendung ausführen. Der Eintrag `reloadable=''true''` bewirkt, dass Tomcat nach Änderungen automatisch die neuste Version der Webanwendung aktiviert.

Das Taschenrechner–Servlet und das HelloWorld–Servlet sind jetzt unter

- http://localhost/Servlet_Beispiele/servlet/TestSrv

  bzw.

- http://localhost/Servlet_Beispiele/servlet/CalcServlet

zu erreichen.

war–Datei erzeugen

Die andere Möglichkeit des Deployments besteht darin eine so genannte `war`–Datei zu erzeugen. `WAR` steht für Web Application Archiv und stellt ein Archiv dar, welches alle Dateien und Verzeichnisse einer Web–Applikation bündelt.

Dazu muss zuerst in den Einstellungen (Properties) des Tomcat–Projekts unter „Tomcat“ der Reiter „Einstellungen für den Export in eine war–Datei“ gewählt und dort eine `war`–Datei festgelegt werden. Die Endung „`.war`“ darf nicht fehlen, z.B. „`beispiel.war`“. Mit einem Rechtsklick auf das Projekt und der Auswahl von „Tomcat Projekt“ → „Exportiere in die in den Projekteinstellungen gewählte `war`–Datei“ wird das Projekt in die angegebene `war`–Datei exportiert.

Diese Datei kann man nun entweder auf dem Tomcat–Server in das Verzeichnis %TOMCAT_HOME%\webapps kopieren oder mit der Manager–Applikation des Tomcat unter http://localhost/manager/html installieren. Die `war`–Datei wird bei Bedarf vom Tomcat automatisch entpackt.

Abbildung 9.21.: JAVA Servlets: Applikations–Installation mit dem Tomcat–Manager

Das Taschenrechner–Servlet und das HelloWorld–Servlet sind jetzt unter

- http://localhost/beispiele/servlet/TestSrv
- bzw.
- http://localhost/beispiele/servlet/CalcServlet

zu erreichen.

Die Erstellung der Web–Applikation und der `war`–Datei kann auch auf andere (manuelle) Weise geschehen. Der hier aufgezeigte Weg mit Eclipse und dem Tomcat–Plugin ist natürlich nicht bindend.

9.4. Quellenangaben

- *Java Servlets und JSP mit Tomcat 4x – Die neue Architektur und moderne Konzepte für Webanwendungen im Detail*
  Peter Roßbach(Hrsg.), Andreas Holubek, Thomas Pöschmann, Lars Röwekamp, Peter Tabatt
  Software & Support Verlag, 2002, 2., unveränderte Auflage

- *Web Services – Grundlagen und praktische Umsetzung mit J2EE und.NET*
  Andreas Eberhart, Stefan Fischer
  Carl Hanser Verlag, 2003
- *Java Server und Servlets – Portierbare Web–Applikationen effizient entwickeln*
  Peter Roßbach, Hendrik Schreiber
  Addison Wesley Verlag, 2000, 2., aktualisierte Auflage
- http://jakarta.apache.org/tomcat/index.html

10. JSP

10.1. Einleitung

In der ersten Generation von Internet-Seiten war jede Seite statisch auf dem Web-Server abgelegt und durch einen eindeutigen Namen identifiziert. Doch dies reichte für viele Anwendungen nicht aus und beschränkte die Interaktionsfähigkeit. Es gibt mehrere gute Gründe, warum Web-Inhalte dynamisch generiert werden sollten:

- **Die Seite ist abhängig von Benutzereingaben**
  Wenn ein Kunde sich beispielsweise für ein Produkt und dessen Preis interessiert hat, dann war es kaum möglich, für jedes Produkt eine aktuelle statische Web-Seite bereitzustellen. Zudem sieht ja jede Seite anders aus, und so gäbe es sehr viele Seiten. Falls sich die Produktbeschreibung ändert, müsste der Benutzer immer eine aktuelle Seite sehen. In diesem Fall ist es günstig, die Web-Seiten bei Bedarf zu erzeugen. Für Einkaufssysteme kommt noch eine weitere Eigenschaft hinzu: Der Benutzer bewegt sich über mehrere Seiten und verwaltet einen Warenkorb, der anwachsen oder schrumpfen kann.
- **Daten ändern sich oft**
  Eine weitere Anwendung ergibt sich, wenn sich die Seiteninformationen ändern. Wie könnten wir die Anzahl der Benutzer, die bis dato auf eine Seite zugegriffen haben, darstellen? Oder was wollen wir machen, wenn Nachrichten oder Börseninformationen von einer Datenbank kommen und auf einer Web-Seite aktuell gehalten werden sollen? Dies würde mit statischen Seiten nur unter großen Verrenkungen möglich sein.

Aus diesen Gründen wurden Schnittstellen definiert, wobei die bekannteste das **Common Gateway Interface** (kurz CGI) ist. Andere Hersteller haben für ihre Server eigene Schnittstellen definiert. So etwa Netscape NSAPI oder Microsoft ISAPI (Internet Information Server). Alle Schnittstellen erlauben dem Web-Server, externe Programme aufzurufen, die dann eine HTML-Seite generieren, die zurück zum Client geschickt wird. Mittels der CGI-Schnittstelle kann der Browser dem Server Daten übergeben, wie etwa ein Produkt, nach dem das Programm suchen soll.

10.1.1. Java Server Pages

1998 wurden von Sun Microsystems™ die JSP's spezifiziert als ein Pendant zu Microsoft´s Active Server Pages. Die Server-Seite erzeugt dynamisch - bei Anforderung der entsprechenden URL durch einen Browser - die anzuzeigende Seite. Aus den Bestandteilen der Server Page wird ein Servlet generiert, dessen Ausführung die Resultatseite erzeugt.

Eine Java Server Page besteht aus

- HTML-Elementen
- Scriptlets: Skript-Elementen in Java, deren Quelltext unmittelbar auf der Seite steht, ähnlich wie JavaScript. Die Java Skriptelemente dienen aber der Erzeugung einer Seite auf dem Server, nicht der Interaktivität auf dem Client.
- Java Servlets, die von der Seite aufgerufen werden und einen Teil zur Ausgabe beitragen
- Java Beans, die ebenfalls dynamisch zum Inhalt der Seite beitragen können.

SUN ist damit auf den Zug aufgesprungen, bei der Seitenerzeugung statische HTML-Elemente mit dynamischen Skriptelementen und anderen dynamischen Elementen zu mischen. Größere Stücke von Anwendungslogik sollten in Beans oder Servlets auslagert werden, statt extensiv Scriptlets auf der Seite selbst zu verwenden. Folgende Vorteile entstehen bei dieser Auslagerung:

- bessere Wartbarkeit der Seite durch Nichtprogrammierer
- Wiederverwertung von Code durch zentrale Codekomponenten
- Erleichterung der Wartung des Codes

10.1.2. Java Servlets

Da die Java Server Pages die Java Servlets verwenden können dessen Möglichkeiten ebenfalls verwendet werden.

- „server side applets, without a face“
- „elements of web applications“
- „bieten die Funktionalität von CGI-Skripts“
- bieten ein Session-Konzept

- bieten über einen sogenanten Kontext ein Konzept einer Web-Application, die sich aus mehreren Servletts, HMTL-Seiten und Java Server Pages zusammensetzen kann
- können Teil einer Java Server Page sein und einen Teil der erzeugten Seite generieren
- „extend the functionality of the server“: können als Filter für bestimmte Dateitypen installiert werden.

SUN nennt folgende Vorteile

- component based
- platform independent, unlike Netscape Server API or Apache modules
- without the performance limitations of CGI programs (process versus thread)
- „crash protection“ of the Java language
- access to all Java APIs, including JDBC API for database access

10.2. Architektur der Java Server Pages

Java Server Pages benötigen eine JSP Engine die eine Umsetzung der JSP-Seite in ein Servlet durchführt und die Ergebnisseite als HTML (oder glw.) zurückliefert.

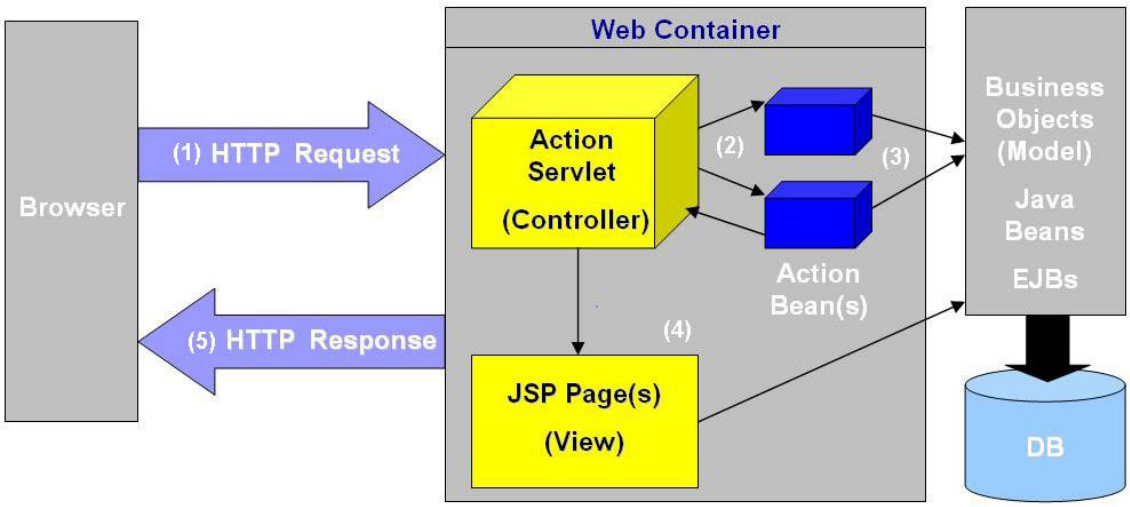


Abbildung 10.1.: JSP: Architektur

1. Der Client-Browser setzt einen HTTP-Request an die Applikation ab
2. Die Controller-Komponente empfängt die Anforderung. Sie entscheidet abhängig von den in ihr codierten Geschäftsregeln, wie die Verarbeitung fortgesetzt werden soll.
3. Die Model-Komponeneten (hier Action Beans und die Buissiness Objects) kümmern sich um die Zusammenarbeit mit den persistenten Datenspeichern oder entfernten Systemen, für die sie zuständig sind.
4. Abhängig von den Verarbeitungsergebnissen und den von den Model-Komponenten zurückgegebenen Daten, bestimmt der Controller, welche View-Komponente für die Anzeige der Daten verwendet werden soll. Daten werden auf die Darstellung durch das View-Objekt vorbereitet
5. Die ausgewählte View-Komponente stellt die HTTP-Antwort dar, die an den Benutzer gesendet wird.

10.3. Ziel der Java Server Pages

Heutzutage müssen ganze Geschäftsprozesse eines Unternehmens im Web präsentiert, überwacht und gesteuert werden. Dadurch entstehen große verteilte und skalierbare Anwendungen, die oft eine immense Komplexität erreichten. Wieder verwendbare Komponenten aus der Logik, die in solch einer Seite enthalten sind, müssen eventuell von verschiedenen anderen Anwendungen gleichzeitig verwendbar sein. Daher ist es notwendig die Präsentation von der Business- und Datenzugriffslogik zu trennen. So kann ein Entwicklerteam die unterschiedlichen Techniken voneinander getrennt entwickeln und testen.

Die Java Server Pages bieten genau diese Möglichkeit der Trennung der Präsentationsebene von der logischen Ebene. Dadurch kann sich ein Designer auf die Darstellung konzentrieren ohne Java beherrschen zu müssen und die Programmierer der Logik brauchen sich nicht Gedanken über die spätere Darstellung der Daten zu machen.

Weiterer Vorteil der Trennung ist die einfachere Wartung, Kontrolle und fortwährende Anpassung des Webauftrittes an die Erfordernisse des Unternehmens.

10.4. JSP und Tomcat

Bei Tomcat werden die Projekte von Webseiten im Unterorder `webapps` abgelegt. Als beispiel habe ich „jsp_1“ angelegt. Dort werden die `.jsp` Dateien hinzugefügt.

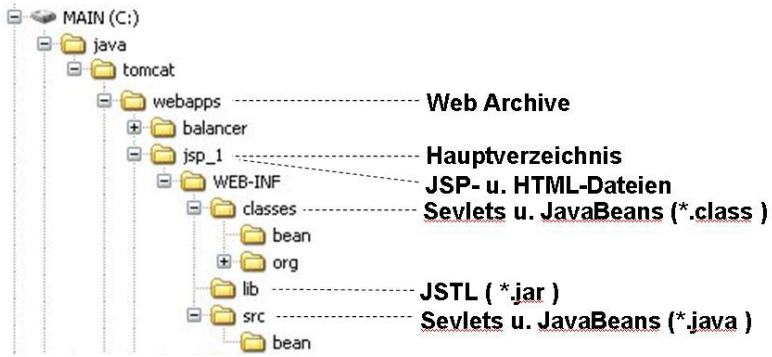


Abbildung 10.2.: JSP: JSP und Tomcat

Die abgelegte Webseite z.B. helloworld.jsp ist dann mit dem Browser mit der Eingabe von http://localhost:8080/jsp_1/helloworld.jsp in der Navigationsleiste erreichbar.

10.5. Anatomie einer Java Server Page

Eine Java Server Page bietet die Möglichkeit dynamischen Inhalt mit statischem Inhalt einer HTML Seite zu mischen. Hierbei werden die dynamischen Teile vom statischen Code durch gesonderte Tags getrennt. HTML-Code wird dabei Template-Text und der JSP-Code als JSP-Element bezeichnet.

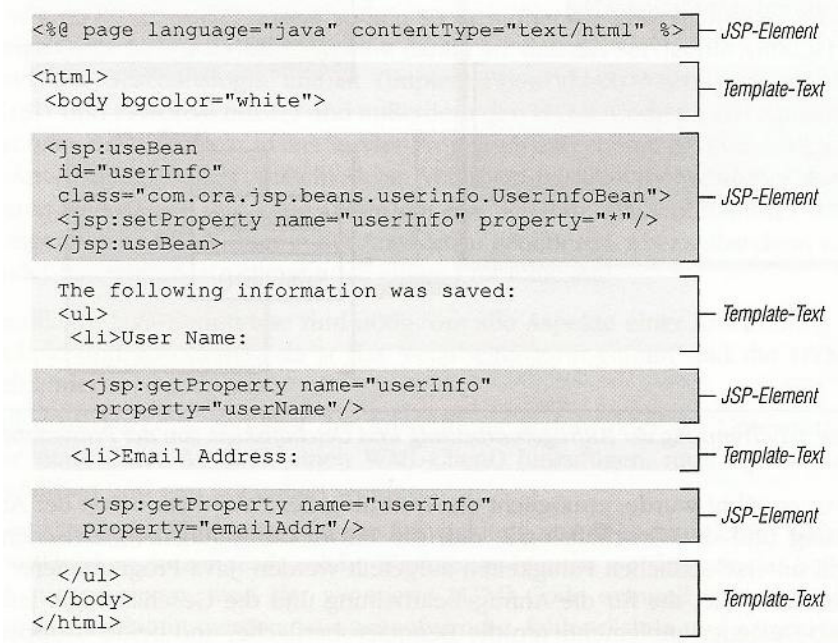


Abbildung 10.3.: JSP: Anatomie

10.6. JSP - Elemente

10.6.1. Vordefinierte Variablen

Die JSP-Spezifikation stellt einige nützliche Variablen bereit.

- **request**
  ermöglicht einen Zugriff auf die Request Parameters, den Request Typ und die Request Headers.
- **response**
  dient zur Rücksendung der Daten an den Client
- **out**
  ist der PrintWriter um die Daten an den Client zu schicken, Größe der Buffers sind veränderbar, dieser werden mit der page directive attribute buffer verändert.
- **session**
  dient um eine Verbindung als eindeutig zu definieren (HTTP ist zustandslos) , diese Variable wird immer erzeugt außer man verwendet die Page directive attribut session und dreht diese ab.
- **application**
  ist die Variable für ServletContext sie beinhaltet persistente Daten und kann von allen Servlets zugegriffen werden.
- **config**
  diese Variable beinhaltet das ServletConfig object für die Seite.
- **pageContext**
  ist die Variable für das PageContext Object, speziell die Java Beans sollte in diesem Object abgelegt sein um einen Zugriff mit der getAttribute Methode jederzeit während eines request zu gewährleisten.
- **page**
  ein Platzhalter für eventuell folgende Scriptsprachen (this)

10.6.2. Comments

In einer Server-Seite können zwei Arten von Kommentaren eingesetzt werden. Einmal Kommentare, die nicht im HTML-Text umgebaut werden und folglich auch nicht zum Browser wandern. Sie haben eines der beiden Formate:

```
1 <%-- Kommentar --%> oder  <% /** Kommentar **/ %>
```

JSP-Kommentare lassen sich, genauso wie normale Kommentare, nicht schachteln.

Der zweite Typ Kommentar ist ein HTML-Kommentar, der auf der Client-Seite sichtbar ist. Er hat folgendes Format:

```
<!-- Kommentar
-->
```

Das Elegante dieser Schreibweise ist, dass in den Kommentar noch Ausdrücke eingesetzt werden können. Um zum Beispiel eine Versionsnummer, die in der Variablen 'version' gehalten wird, zu schreiben, reicht Folgendes:

```
<!-- <%= version %>
-->
```

10.6.3. Expressions

JSP-Ausdrücke sind eine Abkürzung für `out.println()` innerhalb von Scriptlets. Innerhalb der Tags steht ein gültiger Java-Ausdruck ohne abschließendes Semikolon (da es automatisch in ein `print()` gesetzt wird, und da steht ja auch kein Semikolon vor der zweiten Klammer).

```
<% double w=2; %> Die Wurzel von <%= w %> ist <%= Math.sqrt(w) %>
<p>
Die aktuelle Zeit ist <%= new java.util.Date() %>
<p>
Hallo <%= request.getRemoteHost() %>
```

Quelltext 10.1: JSP: Beispiel: Expressons.jsp

Die Ausgabe ist:

```
Die Wurzel von 2.0 ist 1.4142135623730951
Die aktuelle Zeit ist Sun Dec 12 21:11:07 GMT+01:00 2004
Hallo 127.0.0.1
```

Einige Request-Beispiele:

JSP Request Method:	`GET`	`<%= request.getMethod() %>`
Request URI:	`/jsp-examples/snp/snoop.jsp`	
Request Protocol:	`HTTP/1.1`	`<%= request.getProtocol() %>`
Servlet path:	`/snp/snoop.jsp`	
Path info:	`null`	`<%= request.getContentLength() %>`
Server name:	`localhost`	`<%= request.getServerName() %>`
Server port:	`80`	`<%= request.getServerPort() %>`
Remote address:	`127.0.0.1`	
Remote host:	`127.0.0.1`	
Authorization scheme:	`null`	
Locale:	`de`	

Das Beispiel ist zu erreichen über:
http://localhost/jsp-examples/snp/snoop.jsp

Der Quelltext dazu:
http://localhost/jsp-examples/snp/snoop.jsp.html

10.6.4. Scriplets

Scriptlets ermöglichen komplizierte Ausdrücke auszuwerten und mit `out.println` an den Outstream zu übergeben. Alle in Java verwendbaren Konstrukte können auch hier verwendet werden.

Scriptlets liegen zwischen den Tags. Zwischen ihnen kann beliebiger Java-Quellcode eingebettet werden.

```
1 <% if ( Math.random() > 0.5 ) { %>
2   Wow, bist du gut drauf.
3 <% } else { %>
4   Du bist ja ein super Hecht.
5 <% } %>
```

Quelltext 10.2: JSP: Beispiel: WieFuehlIchMich.jsp

Auf die Klammern könnte im Prinzip verzichtet werden, doch ist es eine gute Idee, Probleme zu umgehen und die Blöcke deutlich zu machen. Eine oft genannte und berechtigte Kritik ist, dass JSP keine Tags zur einfachen Bedingungswahl zur Verfügung stellt. Dies kann mit selbst definierten Tags abgefangen werden.

10.6.5. Declarations

Die bisherigen Einbettungen waren lokal in der `service()`-Methode. Zwischen den Tags lassen sich nun Objektvariablen oder Methoden deklarieren. Sie werden außerhalb der Methode `service()` definiert. In Ausdrücken können wir dann die Methoden aufrufen und die Variablen nutzen.

```
1 <%! double d; %>
2 <%! java.awt.Point p = new java.awt.Point(2,3),
3                     q = new java.awt.Point(5,8); %>
4 <%! public java.awt.Point zufall(
5          java.awt.Point p, java.awt.Point q ) {
6     return ( Math.random() > 0.5 ) ? p : q;  } %>
7 <%= zufall(p,q) %>
```

Quelltext 10.3: JSP: Beispiel: Points.jsp

Der Gültigkeitsbereich der Eigenschaften umfasst die aktuelle Datei sowie möglicherweise zusätzlich geladene Dateien. Häufig wird so in Ausdrücken oder Scriptlets auf diese Eigenschaften verwiesen.

10.6.6. Direktiven

Eine Direktive gibt dem JSP-Container besondere Informationen darüber mit, wie er die Seite bearbeiten soll. Sie ergeben keine direkt sichtbare Ausgabe. Direktiven werden in den Tags eingeschlossen, und die allgemeine Form lautet:

```
1 <%@ direktivnname attribut="wert" attribut2="wert" ... %>
```

Jede Direktive besitzt einen Namen. Es gibt sie für die Einbettung von fremden Seiten (`include`), die Definition der Seitenattribute (`page`), die Startinformationen für die Servlet-Umgebung definiert, und weitere.

XML-Form:

```
1 <jsp:directive direktivnname attrib1="wert" attrib2="wert" ... />
```

Direktiven ermöglichen eine bessere Strukturierung in einer JSP Seite. Diese werden in 3 Typen unterschieden:

1. Die `page`-Direktive ist dafür da die Imports der einzelnen Klassen zu kontrollieren
2. Die `include`-Direktive ermöglicht das einfügen von Files in das Servlet zur Übersetzungszeit. Sie wird platziert am Platz wo das File eingefügt werden soll.
3. Die `taglib`-Direktive kann benutzt werden um Custom Tags zu definieren

Eine `page` Direktive beinhaltet ein oder mehrere Attribute die verwendet werden können.

- **import**
  gibt an welche Klassen importiert werden sollen vom generierten Servlet
- **contentType**
  gibt den Content Typ des Response Headers an
- **isThreadSafe**
  gibt an ob ein Servlet multiplen Zugriff gewährt oder nicht
- **session**
  gibt an ob `HttpSession` verwendet wird oder nicht
- **buffer**
  gibt die Größe des `out` Buffers an welcher vom JspWriter verwendet wird
- **autoflush**
  gibt an ob der Buffer automatisch geschickt werden soll oder nicht
- **extends**
  gibt die Superklasse für das generierte Servlet an
- **info**
  mit diesem Attribut kann man eine Info eingeben über das Servlet, diese kann mit `getServletInfo()` abgerufen werden.
- **errorPage**
  ermöglicht eine Seite mit der Bearbeitung einer Exception zu beauftragen.
- **isErrorPage**
  gibt bekannt ob eine Seite als Errorpage für eine andere Seite agieren kann.

10.7. JavaBeans

JavaBeans wurden von Sun Microsystems™ ursprünglich für die Entwicklung von graphischen Oberflächenelementen (GUI-Komponenten) spezifiziert. Die JavaBeans repräsentieren ein Design Pattern das auch für nicht GUI-Komponenten angewendet werden kann. JavaBeans sind Klassen die mind. folgende Kriterien erfüllen:

- Persistenz (Serializable-Interface)
- Get-/Set-Methoden
- Namenskonvention zwischen JavaBeans Attributen und Get-/Set-Methoden

```
1  package bean;
2
3  public class MyBean implements java.io.Serializable {
4    // Attribut des JavaBeans
5    private String strName;
6
7    // Get-/Set-Methoden
8    public String getName() {
9      return strName;
10   }
11
12   public void setName(String pstrName) {
13     this.strName = pstrName;
14   }
15 }
```

Quelltext 10.4: JSP: Beispiel: MyBean.java

Aufruf eines JavaBeans im XML-Format:

```
1  <%@page language="java"%>
2  <!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
3  <jsp:useBean id="myBean" class="bean.MyBean" scope="session">
4    <jsp:setProperty name="myBean" property="strName" value="Bean-Test"/>
5  </jsp:useBean>
6  <html>
7    <body>
8      <h1><b><jsp:getProperty name="myBean" property="strName"/></b></h1>
9    </body>
10 </html>
```

Aufruf als Scriptlet:

```
1  <%@page language="java" contentType="text/html" %>
2  <%@page import="bean.*"%>
3  <!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
4  <jsp:useBean id="MyBean" class="MyBean" scope="session"></jsp:useBean>
5  <% MyBean.setName("Bean-Test");%>
6  <html>
7    <body>
8      <h1><b><%= MyBean.getName()%></b></h1>
9    </body>
10 </html>
```

10.7.1. CustomTags

Seit Version 1.1 unterstützt der JSP Standard die so genannten Tag Extension oder Custom Tags. Darunter versteht man ein Feature, das es dem Entwickler ermöglicht, Logik in eigene Klassen auszulagern und innerhalb einer JSP über XML-konforme Tags zu nutzen. Dadurch wird nicht nur die Wartbarkeit einer Web-Anwendung, sondern auch die Wiederverwendung des bestehenden Quellcodes erhöht, da die Logik nicht mehr in Form von Scriptlets in der JSP direkt steht. Die Definition und Konfiguration der Tags erfolgt in einer speziellen XML-Datei, die als Taglibrary-Descriptor bezeichnet wird. Diese bildet zusammen mit den einzelnen Klassen die eigentliche Taglibrary, kurz Taglib.

Taglibraries bieten ein sehr breites Anwendungsgebiet, welches von der einfachen Formatierung von Ausgaben, über die Iteration über Collections, bis hin zum komplexen Rendering des JSP Outputs per XSLT reicht. Detaillierte Informationen über Tag Extensions gibt es auf der Website von Sun: http://java.sun.com/products/jsp/taglibraries.html

Custom Tags beinhalten Geschäftslogik in einer für den Internetseiten Autor bekannten Form (HTML Style). Custom Tags basieren auf dem Tag Extension Mechanismus (Vordefinierte Klassen und Interfaces) Custom Actions (Tag Handler) sind JavaBeans mit Attributen und Get-/Set-Methoden, die von der Klasse TagSupport abgeleitet worden sind. Eine Tag Library ist eine Menge von Custom Tags, jede Tag Library braucht einen Tag Library Descriptor (TLD) in dem das Mapping zwischen Tag Namen und den Tag Handlern definiert wird.

Der TLD liegt in dem Verzeichnis `/WEB-INF/tlds/` und die Java Klassen für die Tag Library liegen in einem Unterverzeichnis das nach dem Verzeichnis `/WEB-INF/classes/` kommt (z.B.: `/WEB-INF/classes/tags/html/`)

10.7.2. Beispiel

```
1  package bean;
2
3  import java.io.*;
4  import javax.servlet.jsp.*;
5  import javax.servlet.jsp.tagext.*;
6
7  public class MyTag extends TagSupport {
8
9    // Attribut des Tag Handlers
10   private String m_strName;
11
12   // Set-Methode
13   public void setName(String pstrName) {
14     this.m_strName = pstrName;
15   }
16
17   public int doEndTag() {
18     pageContext.getOut().println(m_strName);
19     return(EVAL_PAGE);
20   }
21 }
```

Quelltext 10.5: JSP: Tag Handler: MyTag.java

```
1  <?xml version=1.0" encoding="ISO-8859-1" ?>
2  <!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
3     http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd>
4  <taglib>
5    <tlibversion>1.0></tlibversion>
6    <jspversion>1.1</jspversion>
7    <shortname>mytag</shortname>
8    <tag>
9      <name>mytag</name>
10     <tagclass>de.jspdevelop.tools.MyTag</tagclass>
11     <bodycontent>empty</bodycontent>
12     <attribute>
13       <name>m_strName</name>
```

```
14      </attribute>
15    </tag>
16  </taglib>
```

Quelltext 10.6: JSP: Tag Library Descriptor

```
1  <%@ taglib uri="/WEB-INF/mylib.tld" prefix="MyTagLib"%>
2  <html>
3  <head>
4    <title>Anwendung einer Tag Library</title>
5  </head>
6  <body bgcolor="white">
7    <MyTagLib:mytag m_strName="MyCustomTag"/>
8  </body>
9  </html>
```

Quelltext 10.7: JSP: Custom Tag – Aufruf

10.7.3. Java Standard TagLibs

Unter den vielen schon existierenden Tag-Libraries (kurz Taglibs) standardisiert die JSR-052 Expert Group eine Taglib unter dem Namen JSTL-Tags. Diese Taglib definiert eine Reihe von Paketen für alle Aufgaben, mit denen es JSP-Entwickler aufnehmen müssen. Darunter fallen etwa Datenbankverbindungen, Zeichenkettenverarbeitung, Iterationen oder Datumsformatierungen. Neben dieser Standard Taglib bietet die Apache-Gruppe eine Reihe weiterer Taglibs an.

Jakarta Taglibs:
http://jakarta.apache.org/taglibs/index.html

JSP Standard Tag Library:
http://java.sun.com/products/jsp/jstl/index.html

Taglib	Beschreibung
`Application`	Gibt Zugriff auf das ServletContext-Objekt.
`Benchmark`	Möglichkeiten zur Messung der Performance
`BSF`	Das Bean Scripting Framework (BSF) erlaubt das Einbetten von Scipt-Sprachen wie JavaScript, VBScript, Perl, Tcl, Python in Java.
`DateTime`	Formatieren von Datumsausgaben, Lokalisierung, Eingabe aus HTML-Formularen
`DBTags`	Zugriff aus SQL-Datenbanken
`I18N`	Hilfe bei der Internationalisierung
`Input`	Repräsentiert FORM-Elemente.
`IO`	Ermöglicht HTTP, HTTPS, FTP, XML-RPC oder SOAP Anfragen aus der JSP, um externe Daten einzubinden.
`JNDI`	Erzeugt einen javax.naming.Context.
`Log`	Logging mit log4j in Dateien, JMS, RPC ...
`Mailer`	E-Mail-Versand
`Page`	Zugriff auf den PageContext
`Random`	Generiert zufällige Zeichenketten und Zahlen.
`Regexp`	Reguläre Ausdrücke
`Request`	Zugriff auf den HTTP-Request der JSP
`Response`	Zugriff auf den HTTP-Response
`Scrape`	Entnimmt Teile aus Web-Seiten und bindet sie in die eigene JSP ein.
`Session`	Zugriff auf HttpSession
`Standard`	Die Standard-Taglib, die im Java Community Process JSR-052 beschrieben ist
`String`	Verarbeitung von Zeichenketten
`Utility`	Utilities (Hallo-Tag grüßt die Welt, Validierung eines FORMs, Quellcode anzeigen)
`XSL`	Erlaubt XSL Transformations.
`XTags`	XSLT und Verweise mit XPath

11. SOAP / AXIS

11.1. Einleitung

SOAP[1] soll eine verteilte Anwendung über das Internet ermöglichen. Um dies zu erreichen, verknüpft SOAP HTTP und XML. Durch die Kombination von XML und HTTP gelingt es SOAP sprach– und plattformunabhängig zu sein. Mit Hilfe von HTTP als Transportprotokoll kann SOAP Firewalls überwinden und somit auch über das Internet eingesetzt werden. Demzufolge ist es möglich Dienstleistung (Web Services) über das Web zur Verfügung zu stellen. Diese Web Services können direkt aus Programmen heraus gestartet werden, ohne das der Browser dazu geöffnet werden muss.

11.2. Entstehung und Norm

Im Jahre 1998 wurde die erste SOAP–Spezifikation von Microsoft, DevelopMentor und UserLand Software geschrieben. Der Hauptautor war Don Box von DevelopMentor (einer der Autoren des XML–Manifests). Es war das Ziel, RPCs mit Hilfe von XML über HTTP zu ermöglichen.

Der erste große Durchbruch von SOAP kam mit der Version 1.1, als es im Mai 2000 beim W3–Consortium (W3C) eingereicht wurde. Zudem beteiligte sich auch IBM bei der neuen Spezifikation, so dass SOAP den Ruf verlor eine von Microsoft bestimmende Technologie zu sein. Dadurch beteiligten sich auch noch mehrere anderer großer Firmen an SOAP wie zum Beispiel SUN. Weitere wichtige Gründe für den Aufstieg von SOAP in der Version 1.1 waren noch größere Flexibilität und Erweiterbarkeit, wie zum Beispiel, dass die Einschränkung HTTP als einziges Transportprotokoll fallen gelassen wurde.

Die zur Zeit neueste Version 1.2 von SOAP wurde am 24. Juni 2003 vom W3C standardisiert und wir seitdem nicht mehr als Akronym verwendet. Die XML Protocol Working Group ist eine Arbeitsgruppe vom W3C, die sich mit der Standardisierung von SOAP beschäftigen.

Die heutigen Normen von SOAP befinden sich in den folgenden beiden Dokumenten:

- „SOAP Version 1.2 Part 1: Messaging Framework“

[1] SOAP – Simple Object Access Protocol

- „SOAP Version 1.2 Part 2: Adjuncts“

Im ersten Teil der Norm wird der Envelope beschrieben, welcher ein XML–Konstrukt für die Darstellung des Inhalts einer SOAP–Nachricht ist. Des Weiteren wird definiert, welcher Empfänger für welche Teile der Nachricht zuständig ist und ob die Bearbeitung der an einen bestimmten Empfänger adressierten Teile der Nachricht optional oder zwingend notwendig ist. Darüber hinaus definiert dieser Teil die Art und Weise, wie Spezifikation für die Bindung von SOAP an ein bestimmtes Transportprotokoll geschrieben werden kann. Der zweite Teil definiert das SOAP–Datenmodell, eine Kodierung von Datentypen zur Verwendung bei RPCs sowie eine konkrete Bindung eines Transportprotokolles (HTTP).

Neben diesen beiden Dokumenten existiert ein sogenannter Primer, der eine Einführung in SOAP anbietet, jedoch kein Standard ist.

11.3. Ziele der SOAP–Entwickler

SOAP sollte ein möglichst einfaches und erweiterbares Protokoll sein. Die Entwickler wollten nicht ein Protokoll erschaffen, welches eine möglichst hohe Funktionalität bietet, sondern welches schlank und leicht zu erlernen ist. Zudem sollte SOAP auf bestehende Standards und Techniken (XML und HTTP) aufsetzen. Dieses Protokoll sollte den Grundstein für die Interoperabilität (Sprach– und Plattformunabhängigkeit) von verteilten Anwendungen legen und sogar durch Firewalls hindurch kommen. Aus diesem Grund spielt die Erweiterbarkeit eine große Rolle, denn damit ist gewährleistet, dass das Protokoll an verschiedene Anwendungsmöglichkeiten angepasst werden kann.

11.4. Aufgaben von SOAP

Das Ziel von SOAP ist es, den Austausch von strukturierter Information zwischen verteilten, dezentralisierten Rechnern zu standardisieren. Dies geschieht mit Hilfe eines Protokolls, welches auf XML basiert. Folgende drei Bereiche beinhaltet SOAP:

- einen Serialisierungs–Mechanismus, welcher die Umwandlung von Datentypen in XML regelt
- es ist eine flexibler Aufbau von der SOAP–Nachricht definiert, welcher als „Envelope“ bezeichnet wird
- es ist möglich entfernte Methoden (Remote Procedure Calls = RPCs) aufzurufen

Dabei können anwendungsspezifische Daten in einer SOAP–Nachricht transportiert werden, obwohl das Nachrichtenformat anwendungsunabhängig ist, d.h. das es durch eine

SOAP–Implementation möglich ist, einen Methodenaufruf in eine SOAP–Nachricht umzuwandeln und über das Internet zu einem Rechner zu schicken, welcher die Methode ausführt und eine SOAP–Nachricht mit der Antwort zurückschickt. SOAP detailliert nicht nur die SOAP–Nachricht, sondern auch mit Hilfe eines Transportprotokolls deren Transport. Grundsätzlich soll SOAP ein einfaches, aber erweiterbares Protokoll sein. Aus diesem Grund sind keine Vorgaben gemacht worden. Es ist aber kein Problem SOAP um diese Themen zu erweitern. Bei der Kombination von SOAP–Nachrichten (oder „Message Exchange Pattern“ = MEP) ist die Spezifikation auch sehr offen. Es sind nur zwei MEPs definiert, obwohl es möglich ist, beliebig viele SOAP–Nachrichten miteinander zu kombinieren.

11.5. SOAP–Architektur

In der folgenden Abbildung werden alle Zwischenschritte verdeutlicht, die eine SOAP Nachricht zu durchlaufen hat, ehe diese beim Empfänger ankommt.
Zunächst wird eine Nachricht vom Empfänger erzeugt. Diese wird vom SOAP–System weitergeleitet, indem der Inhalt in XML serialisiert bzw. encodiert wird. Nach diesem Schritt erhalten wir eine sogenannte SOAP–Nachricht, die ein bestimmtes Format besitzt, um sich von einem herkömmlichen XML Dokument zu unterscheiden.
Im nächsten Schritt Packaging bzw. Retrieving, findet die Abbildung der SOAP–Nachricht auf das gewählte Transportprotokoll statt. Für die Übermittlung mit diesem Protokoll wird eine entsprechende Einheit, Underlying Protocol Support, benötigt, welche dieses Protokoll implementiert. Falls HTTP als Transportprotokoll verwendet, wird an dieser Stelle Webserver verwendet. Auf der Empfängerseite wird die SOAP–Nachricht auf gleiche Weise, aber in umgekehrter Reihenfolge verarbeitet und schließlich dem Empfänger weitergeleitet.

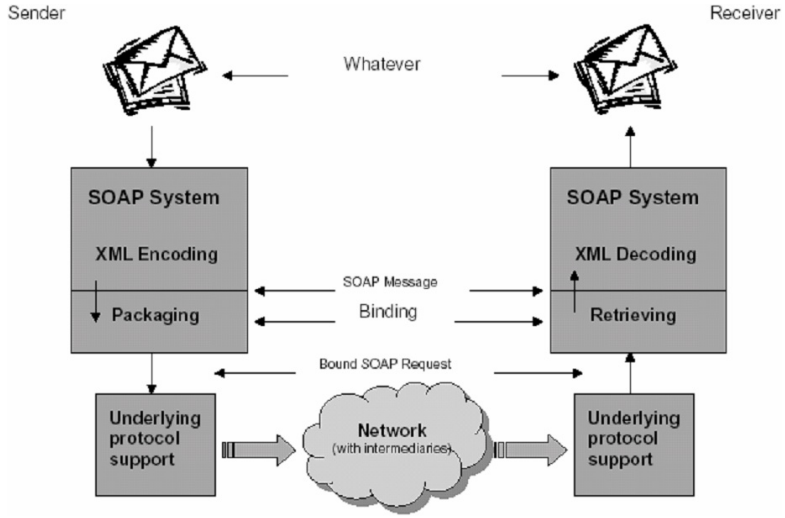


Abbildung 11.1.: SOAP: Kommunikation

11.6. RPCs mit SOAP

Um Berechnungen auf mehreren Computern durchführen zu können, gibt es seit 1984 den Ansatz der entfernten Funktionsaufrufe (RPC; engl. remote procedure call), der von den Amerikanern Birrell und Nelson ersonnen wurde. Das zugrundeliegende Konzept ist relativ einfach: Ein Programm X auf Rechner A kann ein Unterprogramm X1 auf einem anderen Rechner B aufrufen und dabei evtl. auch Parameter übergeben. Programm X wartet nun, bis das Ergebnis von dem Unterprogramm nach Abschluß der Berechnungen zurückgegeben wird.

Für die Parameterübergabe packt das Programm auf dem anfragenden Rechner (Client) die Parameter ein, was durch einen sog. Client–Stub realisiert wird. Dabei sind zunächst einmal nur Call–By–Value–Parameterübergaben möglich, da ja beide Prozesse auf verschiedenen Rechnern und damit in unterschiedlichen Adressräumen laufen. Ein Server–Stub auf dem Rechner, an welchen die Anfrage gerichtet ist, packt die Parameter wieder aus und ruft die jeweilige Methode auf.

SOAP ist ein Protokoll für RPCs. Damit wird erreicht, dass Funktionen auf entfernten Rechnern mit den dazugehörigen Parametern aufgerufen und deren Resultat empfangen werden können. Es ist möglich, mit einem beliebigen Rechner im Internet auf eine einfach, standardisierte Weise auf einen anderen Rechner zuzugreifen und seine Dienste nutzen.

RPC Protokolle wie DCOM oder CORBA ermöglichen verteilte Anwendungen innerhalb eines Intranets, d.h. dass Programme auf verschiedene Computer aufgeteilt werden können. Jedoch benötigen diese für die Kommunikation eigene Protokolle und spezielle Ports. Über das Internet funktioniert die Kommunikation mit DCOM oder CORBA nur selten, da die Firewall die Protokolle abblockt.

Aus diesem Grund muss auf ein Transportprotokoll zurückgegriffen werden, welches ungehindert durch Firewalls kommt. Daneben wird ein plattform– und programmiersprachenunabhängiges Format zur Beschreibung und Transformation benötigt. Für diesen Zweck eignet sich XML sehr gut. Außerdem ist XML viel flexibler und einfacher erweiterbarer als andere Formate. Die eben aufgezählten Dinge vereinigt SOAP.

Beim Nutzen des HTTP für Übertragungszwecke fügt sich ein RPC auf natürliche Weise in das Request/Response–Schema ein. Dabei wird der RPC als HTTP–Request übermittelt, das Ergebnis der Berechnung als HTTP–Response.

Um eine entfernte Methode aufzurufen, werden im Allgemeinen die folgenden Informationen veröffentlicht:

- Der URI des Zielobjektes
- Der Methodenname
- Evtl. vorhandene aktuelle Parameter der Methode.

Zusätzlich können weitere Informationen im SOAP–Header transportiert werden.

Für das Serialisieren eines RPC in einer SOAP Nachricht wurden in der Spezifikation relativ strikte Regeln definiert, um die gewünschte Interoperabilität zwischen verschiedenen System sicher zu stellen. SOAP ist also ein Protokoll, welches eine bisher große und unbesetzte Lücke schließt.

Die untere Abbildung soll den eben beschriebenen Vorgang verdeutlichen:

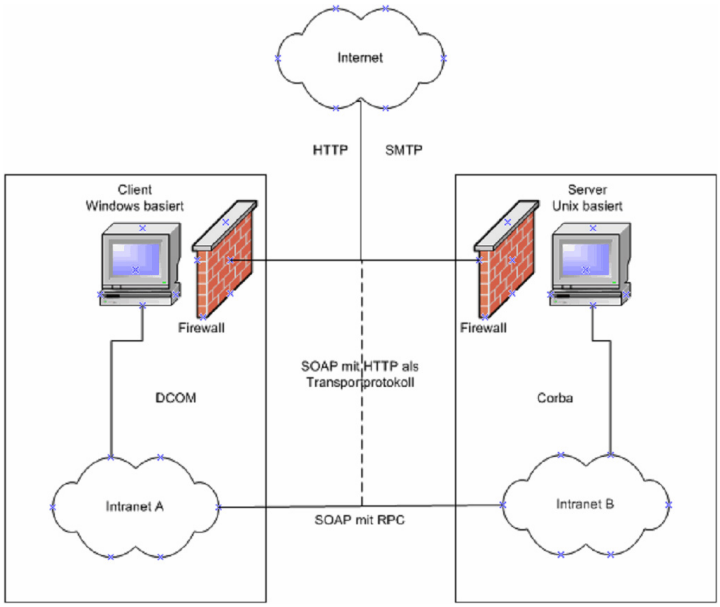


Abbildung 11.2.: SOAP: RPCs mit SOAP

11.7. Aufbau einer SOAP–Nachricht

Jede SOAP–Nachricht besteht aus einem Envelope, der optional einen Header und einen zwingend erforderlichen Body enthält. Im Body werden die eigentlichen Daten übertragen. Der Header enthält optionale Daten, wie zum Beispiel für Authentifikation oder Transaktionsverarbeitung. Diese Informationsblöcke beziehen sich da drauf, wie die Nachricht verarbeitet werden soll. Ist ein Header vorhanden, so muß dieser direkt auf den Envelope folgen.

11.7.1. Envelope

Envelope bedeutet auf Deutsch Umschlag. Diese Wortwahl wurde gewählt, weil der Envelope Tag das oberste Element des XML–Dokuments ist und die Nachricht wie ein Umschlag umschließt. Der Envelope enthält genau einen Body, kann aber so viele Kind–Knoten besitzen wie nötig sind. Zudem enthält der Envelope einen optionalen Header.

11.7.2. Header

Der Header darf nur ein einziges mal im Envelope erscheinen und muß als erstes Kind des Envelope auftauchen. Er kann jeden beliebigen gültigen und namensraumqualifizierten XML–Code enthalten. Die Informationen, die im Header angegeben werden können, sind nicht direkt als Anwendungsdaten zu klassifizieren. Der Header ermöglicht somit, einer SOAP–Nachricht noch zusätzliche Informationen mitzugeben, die in der eigentlich Nachricht (im Body) nicht enthalten sind, wie zum Beispiel ein Passwort oder eine Benutzer–ID. Der Zweck eines Headers ist also Kontextinformationen mitzuteilen, die für die Verarbeitung einer SOAP–Nachricht relevant sind.

11.7.3. Body

Der Body kann drei verschiedene Funktionen übernehmen. Bei einem Aufruf einer SOAP-Nachricht enthält dieser die eigentliche SOAP–Nachricht, wobei die meisten ein Methodenaufruf mit Parametern ist. Erfolgt nach dem Aufruf eine Antwort, so enthält dieser das Resultat des Aufrufes. Kommt es aber beim Abarbeiten der SOAP–Nachricht zu einem Fehler, wird dies im Body durch eine Fehlermeldung signalisiert.

Faults

Ein spezieller Nachrichtentyp ist ein SOAP–Fault. Dieser erlaubt es, Informationen über Fehler beim Verarbeiten einer SOAP–Nachricht anzuzeigen. Folgende Informationen werden übermittelt:

- `Fault-Code`
  Identifikation des augetretenden Fehlertyps
- `Fault-String`
  Erklärung des Fehlers
- `Fault-Actor`
  Bezeichner des Nachrichtenverarbeitungsknoten bei dem der Fehler aufgetreten ist
- `Fault-Details`
  Anwendungsspezifische Einzelheiten über den ausgetretenden Fehler

Dabei gibt es vier Standardtypen von SOAP–Faults:

- `VersionMismatch`
  SOAP–Envelope verwendet einen ungültigen Namensraum für das SOAP–Element Envelope.

- `MustUnderstand`
  Ein Headerblock, der das Flag `mustUnderstand=true` enthält, wurde vom Empfänger nicht verstanden.

- `Server`
  Es ist ein Fehler aufgetreten, der nicht unmittelbar mit der Verarbeitung der Nachricht in Verbindung gebracht werden kann.

- `Client`
  In der Nachricht tritt ein Problem auf.

Wenn die Standardtypen von SOAP–Faults nicht ausreichen, können Faults auch selbst definiert werden.

11.7.4. Attribute

Das SOAP–Verarbeitungsmodell besteht aus mindestens zwei Knoten. Es besteht die Möglichkeit, dass eine Nachricht über einen dritten Knoten zum Endknoten geleitet wird. Was ein Knoten zu tun hat, wenn er eine SOAP–Nachricht erhält, ergibt sich aus den SOAP spezifischen Teilen des Header. Wenn ein Header existiert, müssen die im Folgenden beschriebenen drei Attribute für den Knoten eine Rolle spielen.

Attribut role

Dadurch das es Zwischenknoten gibt, kann jeder von diesen eine bestimmt Aufgabe übernehmen. Damit dieser weiß, was er zu tun hat existiert das `role` Attribut. Falls das Attribute `role` vorhanden ist, beschreibt es dem Knoten, ob der Header für diesen Knoten relevant ist. Dabei kann `role` drei verschiedene Werte annehmen.

Knoten	**Keine Angabe**	**none**	**next**	**ultimate Receiver**
Initiator	—	—	—	—
Zwischenknoten	nicht relevant	nicht relevant	relevant	nicht relevant
Endgültiger Empfänger	Relevant	nicht relevant	relevant	relevant

Tabelle 11.1.: SOAP: Attribut role

Attribut mustUnderstand

Enthält ein Header–Block den Wert `true` für das `mustUnderstand`, so ist die Bearbeitung dieses Header–Blocks obligatorisch. Kann der Zielknoten diesen nicht „verstehen“, so muss die ganze Bearbeitung der SOAP–Nachricht abgebrochen werden. Zudem wird

eine Fehlermeldung an den Versender geschickt.
Weiterhin muss beachtet werden, dass der Wert dieses Attributs sich über die Zwischenknoten verändern kann.

Attribut relay

Eine weitere Verfeinerung bei der Zuordnung eines Headers zu einem SOAP–Knoten bietet das `relay` Attribut. Dieses zeigt an, ob der Header–Block von einem bestimmten Zwischenknoten weitergeleitet werden muss, wenn er nicht bearbeitet werden konnte. Falls das `relay` Attribute auf `true` gesetzt ist, so wird der betroffene Header–Block weitergeleitet. Andernfalls wird der Header–Block entfernt.

11.8. SOAP in der Java–Welt

Im Rahmen des Projektes verwenden wir den Java–Applikationsserver Tomcat, auf dem ein Webservice den verschiedenen Clients zur Verfügung gestellt werden soll. Um das Zusammenwirken der Technologien zu verdeutlichen, soll nachfolgend eine Implementierung von jeweils einer Server– als auch einer Client Java–API beschrieben werden.
Die Verwendung des Protokolls für Anfragen (Requests) und Antworten (Responses) ist SOAP.

11.8.1. Web Service

Für das Beispiel haben wir eine Addiermaschine gewählt, die zwei ganze Zahlen addiert und als Ergebnis die Summe wiederum als ganze Zahl zurückgibt.

11.8.2. Programmierung

Zunächst ist ein Java–Package zu erstellen, in dem dann die Klasse `Calculator` erzeugt wird. Die Methode `add` führt die Addieroperation aus. Als Parameter erwartet `add` die Integer–Werte `param1` und `param2`, die sie dann nach entsprechender Addition zurückgibt:

```
1 package javaSoapExample;
2
3 public class Calculator {
4
5   public int add(int param1, int param2){
6     return (param1 + param2);
7   }
8 }
```

Quelltext 11.1: SOAP: Calculator.java

11.8.3. Deployment

Um den Webservice auf dem Applikationsserver verfügbar zu machen, ist ein entsprechendes Deployment notwendig.
Die Deployment–Prozedur ist kein Bestandteil der SOAP–Spezifikationen, sodass hier verschiedene Ansätze möglich sind. Für das Projekt betrachten wir die Prozeduren unter Apache SOAP und AXIS (Apache Extensible Interaction System) näher. Zuvor sollen diese beiden Frameworks kurz vorgestellt werden.
Sowohl Apache SOAP als auch AXIS sind Apache–Jakarta Subprojekte. AXIS gilt dabei als designierter Nachfolger von Apache SOAP und zeichnet sich vor allem durch die verbesserte Architektur aus. Weiterhin existieren erhebliche Performance–Unterschiede: Apache SOAP basiert auf dem langsamen Document Object Model (DOM), während AXIS mit dem wesentlich schnelleren Simple API for XML (SAX) ausgestattet ist. Das vollständig in Java implementierte AXIS–Framework bietet zudem einige hilfreiche Tools, wie z.B. Codegeneratoren.

Deployment mit Apache SOAP

Nach erfolgreicher Apache SOAP–Installation unter Tomcat muss zunächst das Package–Verzeichnis (hier: `javaSoapExample`) mit dem compilierten Web Service in das Tomcat–Verzeichnis `%TOMCAT_HOME%\webapps\soap\WEB-INF\classes` kopiert werden.
Über einen Deployment–Descriptor (XML–Datei) teilen wir dem Applikationsserver die Informationen über den Webservice mit:

```
1 <isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
2       id="urn:CallCalculator">
3   <isd:provider type="java" scope="Application" methods="add">
4     <isd:java class="javaSoapExample.Calculator" static="false"/>
5   </isd:provider>
6   <isd:faultListener> org.apache.soap.DOMFaultListener</isd:faultListener>
7 </isd:service>
```

Quelltext 11.2: SOAP: CallCalculator.dd

Die Datei kann beispielsweise im Workspace unter `CallCalculator.dd` abgespeichert werden.

Das äußerste XML–Element heißt Service und wird durch den Identifier `isd` und den Namespace `http://xml.apache.org/xml-soap/deployment` repräsentiert. Das `id`–Attribut stellt den eindeutigen Namen des Web Services dar und kann durch die Bezeichnung `urn:` erweitert werden. (Anmerkung: Ein `urn`–Zusatz ist common practice – aber nicht zwingend)

Das erste Child–Element heißt Provider und beschreibt die Implementation des Services:

`type`	Apache SOAP erlaubt neben der hier verwendeten Java–Implementation auch andere, die hier aber nicht weiter behandelt werden sollen.
`scope`	Spezifiziert das jeweilige Service Activation Model und bestimmt die Lebenszeit des Service–Objektes. Man unterscheidet in HTTP–basierten SOAP–Systemen generell die folgenden Werte: `request`: Für jede Anfrage (request) wird jeweils eine neue Instanz des Service–Objektes angelegt, das lediglich die Dauer einer Anfrage überlebt. `application`: Eine einzelne Instanz läßt Methodenaufrufe während der gesamten Lebensdauer der Service–Applikation zu. `session`: Eine einzelne Instanz läßt Methodenaufrufe während der gesamten Lebensdauer der Sitzung zu, die beispielsweise durch den Einsatz von Cookies bestimmt werden kann. Zwangsläufig ist hier eine Erweiterung um Headerinformationen notwendig.
`methods`	Hier sind die Namen der Methoden aufgeführt, die vom Service dargelegt werden.

Tabelle 11.2.: SOAP: Deployment Descriptor

Das nächste Element Class ist dem `type`–Attribut „Java“ zugehörig und beschreibt die Java–Klasse, die den Service implementiert. Beim Verwenden von Packages muss darauf geachtet werden, dass hier der gesamte Pfad angegeben wird.
Mit dem `static`–Attribut müssen die gleichnamig deklarierten Methoden der Klasse indiziert werden.

Das `faultListener`–Element ist verwantwortlich für die Fehlerbehandlung; hier wurde die von ApacheSOAP zur Verfügung gestellten Standard–Klasse verwendet:
`org.apache.soap.server.DOMFaultListener`

Letztlich muss der Deployment–Descriptor ausgeführt werden. Hierzu sollen zwei Varianten beschrieben werden:

Alternative 1
Die Ausführung über die Kommandozeile:

```
1 java org.apache.soap.server.ServiceManagerClient
2 http://localhost:8080/soap/servlet/rpcrouter
3 deploy CallCalculator.dd
```

Der erste Parameter ist die URL des Apache SOAP RPC–Routers, der hier in diesem Beispiel über die lokale Loopback–Adresse `localhost` mit dem Port 8080 erreichbar ist. Die Parameter `deploy` und `CallCalculator.dd` sind die Anweisung, die entsprechende Datei zu deployen. (Anmerkung: Ein häufiger Fehler ist, dass die Datei `CallCalculator.dd` nicht gefunden wird. Bei der Ausführung sollte darauf geachtet werden, dass man sich entweder im selben Verzeichnis befindet, in der auch die CallCalculator–Datei abgelegt ist oder eine entsprechende Erweiterung des obigen Befehls um den kompletten Pfad vorgenommen wird.)

Wenn bei der Ausführung keine Fehlermeldungen auftreten, sollte der Service jetzt erfolgreich deployed worden sein. Hierzu kann man sich eine Liste der deployten Services anzeigen zu lassen:

```
java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter list
```

Ein entsprechendes Undeployment für den Service `urn:CallCalculator` kann mit folgendem Befehl vorgenommen werden:

```
java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter
undeploy urn:CallCalculator
```

Alternative 2
Die Ausführung über das Admin–Webinterface von ApacheSOAP:

Im Browser kann über die folgende URL http://localhost:8080/soap/admin (ggf. durch eigene SOAP–URL ersetzen) die Administrations–Seite aufgerufen werden:

Abbildung 11.3.: SOAP: ApacheSOAP Admin–Oberfläche

Der List–Button zeigt eine Liste der deployten Services. Klickt man auf einen Service, so erhält man weitere Informationen aus der Deployment–Descriptor–Datei.

Das Deployment kann mit dem Button Deploy durchgeführt werden. Hier müssen die einzelnen Eingabefelder mit den Angaben aus der Deployment–Descriptor–Datei vorgenommen werden.

Ein Undeployment ist letztlich über den Button „Un–deploy“ und der Auswahl des Services möglich.

Deployment mit AXIS

Nach erfolgreicher AXIS–Installation (siehe Anhang) unter Tomcat muss zunächst die `Calculator.java` Datei in das Verzeichnis `%TOMCAT_HOME%\webapps\axis` kopiert und in `Calculator.jws` umbenannt werden. AXIS bietet ein Auto–Deployment an, welches nach einem Neustart des Applikationsservers selbständig durchgeführt wird. Die compilierte Klasse wird im Verzeichnis `WEB-INF\jwsClasses` abgelegt.

Ob der Service erfolgreich deployed wurde, kann im Browser mit der folgenden URL (ggf. durch eigene AXIS–URL ersetzen) überprüft werden:
http://localhost:8080/axis/Calculator.jws

Anmerkung: Der Webservice wird bei AXIS durch die WebService–Definition–Language (WSDL) beschrieben. Das WSDL–Dokument beschreibt im Wesentlichen, welche Methoden der Webservice anbietet, mit welchen Parametern sie aufzurufen sind und was sie zurückliefern. Die in XML formulierte Datei kann man sich ebenfalls im Browser unter Erweiterung der obigen URL um die Bezeichnung `?wsdl` anzeigen lassen.

Eine Liste der deployten Services kann man sich in der Konsole über den Befehl

```
java org.apache.axis.client.AdminClient list
```

anzeigen lassen.

Das Undeployment wird durch eine entsprechende WSDL–Datei vorgenommen:

```
<undeployment xlmns="http://xml.apache.org/axis/wsdd/">
  <service name="CalculatorService"/>
</undeployment>
```

Quelltext 11.3: SOAP: undeploy.wsdl

```
java org.apache.axis.client.AdminClient undeploy NameDerUndeployDatei.wsdl
```

11.8.4. Aufruf des Webservices

Nachdem wir den Webservice erfolgreich deployed haben, wollen wir ihn jetzt nutzen bzw. testen. Hierzu muss jeweils eine Client–API programmiert werden. Die Implementierung wird wie beim Service in Java vorgenommen (ist aber nicht zwingend notwendig).

ApacheSOAP–Client

Zunächst müssen die SOAP–Libraries zum Projekt hinzugefügt werden. In unserem Package ist dann die Klasse `CalculatorInvoke` zu erstellen. Bevor wir mit der Programmierung dieser Klasse anfangen, sollten die packages `java.net.*`, `java.util.*`, `org.apache.soap.*` und `org.apache.soap.rpc.*` importiert werden. Die `main`–Methode ist um die Erweiterung `throws Exception` zu erweitern. Damit sind die Vorarbeiten abgeschlossen, sodass wir uns jetzt dem Quelltext widmen können:

```
package javaSoapExample;

import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class CalculatorInvoke {
```

```
9
10    public static void main(String[] args) throws Exception {
11      URL url = new URL("http://localhost:8080/soap/servlet/rpcrouter");
12      Call myCall = new Call();
13      myCall.setTargetObjectURI("urn:CallCalculator");
14      myCall.setMethodName("add");
15      myCall.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
16      Vector myParams = new Vector();
17      myParams.addElement(new Parameter("param1", Integer.class, "427", null));
18      myParams.addElement(new Parameter("param2", Integer.class, "7142", null));
19      myCall.setParams(myParams);
20      Response resp = myCall.invoke(url, "");
21      if (resp.generatedFault()) {
22        Fault fault = resp.getFault();
23        System.out.println("Fault-Code: " + fault.getFaultCode());
24        System.out.println("Fault-String: " + fault.getFaultString());
25      } else {
26        Parameter ret = resp.getReturnValue();
27        Integer value = (Integer) ret.getValue();
28        System.out.println("Result is " + value);
29      }
30    }
31  }
```

Quelltext 11.4: SOAP: CalculatorInvoke.java

Zunächst wurde eine Instanz der Klasse `java.net.URL` mit dem Apache SOAP RPC–Router unseres SOAP–Servers erstellt. Die URL des Routers lautet in unserem Beispiel analog zum bereits oben beschriebenen Deployment wie folgt:
http://localhost:8080/soap/servlet/rpcrouter.
Danach wurde die Instanz `myCall` der Klasse `org.apache.soap.rpc.Call` erzeugt. Die `Call`–Klasse vereint die SOAP–RPC Methoden.
Mit `setTargetObjectURI()` und `setMethodName()` spezifizieren wir den Namen des Service–Objektes und die Service–Methode, die wir nutzen wollen.
Die Konstante `Constants.NS_URI_SOAP_ENC` ist ein vordefinierter Wert für den Standard–Namespace `http://schemas.xmlsoap.org/soap/encoding/`. Sie wird mit der Methode `setEncodingStyleURI()` unserer `myCall`–Instanz zugewiesen. Die Parameter für die `add`–Methode unseres Addiermaschinen–Services werden mit `setParams()` übergeben.
`setParams()` erwartet einen `java.Util.Vector`, der wiederum die gewünschten Parameter enthält. In unserem Beispiel wurden in dem Vector die Integer–Werte `427` und `7142` angefügt.
Das Abschicken des myCall–SOAP–Requests erfolgt mit der Methode `invoke()`, die als Parameter die RPC–Router–URL und einen Leerstring übergeben bekommt. Die SOAP–Antwort wird in der Instanz `resp` der Klasse `java.org.apache.soap.rpc.Response` gespeichert.
Aufgetretene Fehler werden im dann folgenden 'If'–Teil mit Fehler–Code und –Bezeichnung ausgegeben. Der 'Else'–Teil gibt das in Integer gecastete Ergebnis auf dem Bildschirm aus. Ein Casting ist zwingend erforderlich, da die Apache SOAP RPC Methode lediglich Instanzen von Java–Objekten zurück liefert, jedoch keine Instanzen von Java–Primitivtypen.

AXIS–Client

Beim Erzeugen einer AXIS–Client–API können wir auf automatische Code–Generatoren zugreifen, die uns einen Großteil der Kommunikationslogik erzeugt. Bereits beim Deployment mit AXIS haben wir die Beschreibung eines Web Services mit WSDL angemerkt. Diese Funktionalität wollen wir im Folgenden nutzen, um daraus entsprechende Hilfsklassen und Interfaces zu erzeugen, die die Generierung einer Client–API wesentlich vereinfachen.

Anmerkung: WSDL wird ausführlich in einem eigenen Kapitel beschrieben, wir nutzen es hier lediglich für die automatische Codegenerierung.

Für die Programmierumgebung Eclipse existieren WSDL2Java–Plugins (z.B. „WSDL2Java Wizard“), die hier genutzt werden können. Alternativ dazu stellen wir die Möglichkeit vor, die Klassen über einen Aufruf in der Konsole zu erzeugen:

```
1 java -cp org.apache.axis.wsdl.WSDL2Java http://localhost:8080/axis/Calculator.jws?wsdl
```

(Selbstverständlich kann die URL durch den entsprechenden lokalen Pfad ersetzt werden, wenn die WSDL–Datei zuvor gespeichert/angepasst wurde.)

Bei erfolgreicher Generierung wird keine Erfolgsmeldung ausgegeben und die neuen Dateien sind entsprechend im aktuellen Verzeichnis zu finden:

`Calculator.java` (Interface, das den Web Service beschreibt)
`CalculatorService.java` (Interface, das einen Stub liefert (die Factory))
`CalculatorSoapBindingStub.java` (Klasse, bzw. der Stub, der Calculator implementiert)
`CalculatorServiceLocator.java` (Klasse, die die Factory implementiert)

Diese Dateien sind jetzt im Workspace der Entwicklungsumgebung (ggf. in einem seperaten Package) zu übernehmen, sodass sie referenziert werden können.

Als nächstes muss eine eigene Klasse erstellt werden, die wir in unserem Beispiel InvokeCalculator benannt haben. Die Programmierung der eigentlichen Client–API erfordert jetzt nur noch wenige Code–Zeilen:

```
1  package localhost.axis.Calculator_jws;
2
3  public class InvokeCalculator {
4
5    public static void main(String[] args) throws Exception {
6      CalculatorService myService = new CalculatorServiceLocator();
7      Calculator myPort = myService.getCalculator();
8      System.out.println("Die Summe lautet: " + myPort.add(10, 20));
9    }
10 }
```

Quelltext 11.5: SOAP: InvokeCalculator.java

Zunächst ist die `main()`–Methode wieder um `throws Exception` zu erweitern. Dann ist dem CalculatorService–Interface, `myService`, eine Instanz der Locator–Klasse zuzu-

weisen. Das Beschreibungs–Interface `myPort` bekommt die Methoden des Services über die Interface–Methode `getCalculator()`. Der Aufruf der Methode `add()` kann jetzt über `myPort` mit den gewünschten Integer–Parametern vorgenommen und ausgegeben werden.

Insbesondere der geringere Programmieraufwand und die bessere Struktur sind hier als Vorteile gegenüber der Implementierung mit Apache SOAP zu nennen.

12. WSDL

12.1. Web Services

WSDL[1], SOAP[2] und UDDI[3] sind drei Standards, die sich ergänzen und Web Services ermöglichen sollen. Alle drei Technologien sind dabei unabhängig voneinander, es ist also nicht zwingend notwendig WSDL mit SOAP und UDDI zusammen zu benutzen. Aufgrund der Unterstützung namhafter großer Firmen läuft jedoch alles auf diese Kombination hinaus. Dank Plattformunabhängigkeit und Unabhängigkeit von der Anbindung einer Programmiersprache ermöglichen diese Standards das Vergleichen und Kombinieren von Web Services.

12.1.1. WSDL, SOAP und UDDI

Eine Web Services-Architektur besteht typischerweise aus drei Beteiligten: Service Provider, Service Broker und Service Client. Der Service Provider, der einen Web Service anbieten will, legt dazu auf seinem Server eine WSDL-Datei ab. Um auf seinen Web Service aufmerksam zu machen, veröffentlicht er danach seinen Web Service bei einem Service Broker. Dieser soll als Vermittler zwischen Service Provider und Service Client agieren. In dem UDDI-Register des Service Brokers ist die Adresse der WSDL-Datei hinterlegt. Der Service Client wendet sich zuerst an den Service Broker, um einen gewünschten Web Service zu suchen. Hat er einen Web Service gefunden kann er anhand der angegebenen Adresse der WSDL-Datei diese Datei nun ausfindig machen und sie auswerten. Die Definitionen in der WSDL-Datei spezifizieren das Protokoll, die Adresse und andere Dinge, die es ihm ermöglichen, mit dem Service Provider in Verbindung zu treten, um den gewünschten Web Service zu benutzen.

12.2. Einleitung

WSDL ist ein Standard der von den Firmen Ariba, Microsoft und IBM entwickelt wurde. Er ist noch nicht fertiggestellt, sondern befindet sich noch mitten in der Weiterentwicklung.

[1] WSDL - WebService Description Language
[2] SOAP - Simple Object Access Protocol
[3] UDDI - Universal Description, Discovery and Integration

Die Spezifikation des Standards wurde dem W3C als Vorschlag zur Beschreibung von Web Services vorgelegt. Die Veröffentlichung des Standards durch das W3C gibt dem W3C selber aber keine Ermächtnis zur Änderung oder Weiterentwicklung des Standards.

12.2.1. Eigenschaften

`WSDL` ist eine `XML`-basierte Beschreibungssprache für Web Services. Beschrieben wird der Nachrichtenaustausch zwischen Netzwerkendpunkten, der sowohl dokument-orientierte wie auch prozedur-orientierte Informationen enthalten kann. Die Operationen und Nachrichten werden zuerst abstrakt beschrieben und dann an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden.

`WSDL` ermöglicht Erweiterbarkeit, da Beschreibungen von Nachrichten erlaubt werden, unabhängig von den Nachrichtenformaten und Netzwerkprotokollen, die zur Kommunikation benutzt werden. Bisher werden in der `WSDL`-Spezifikation Bindungen zu den Formaten `SOAP`, `HTTP GET/POST` und `MIME` unterstützt.

`WSDL` soll Teil eines Rezepts sein, das die Kommunikation von Anwendungen erleichtern und automatisieren soll.

12.3. Dokumentstruktur

Ein `WSDL`-Dokument kann man sich vorstellen als eine Ansammlung von Definitionen. Diese können in zwei Gruppen aufgeteilt werden:

- abstrakte Definitionen, in diesen werden Typen, Nachrichten und Sende- bzw Empfangsoperationen definiert.
- konkrete Definitionen, in diesen werden die oben genannten Definitionen einer Bindung und einer Netzwerkadresse zugeordnet.

Zu den abstrakten Definitionen gehören die Elemente `types`, `message` und `portType`. Zu den konkreten Definitionen werden die Elemente `binding` und `service` gezählt.

12.3.1. Aufbau

Die Datei `example.wsdl` enthält ein äußeres Element. Es heißt `definitions` und enthält die Elemente `types`, `message`, `portType`, `binding` und `service`. Diese wiederum enthalten weitere Unterelemente. Im Element `types` befindet sich das Element `schema`, im Element `message` das Element `part`. Die Elemente `portType` und `binding` haben jeweils ein Unterelement `operation` und das Element `service` enthält das Element `port`.

Weitere Unterelemente werden später vorgestellt. Alle Elemente werden nach bekannten `XML-Regeln` mit Anfangs- und Ende-Tag eingefügt.

Der Aufbau der Beispieldatei example.wsdl:

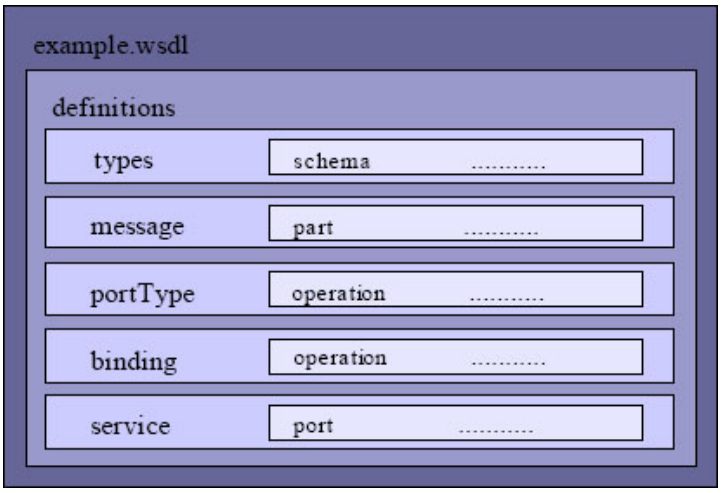


Abbildung 12.1.: WSDL: Beispiel

12.3.2. Wiederverwendbarkeit

Die Dokumentstruktur des obigen Abschnitts hat zwei Nachteile. Als erstes können Dateien sehr umfangreich und somit unübersichtlich werden. Des Weiteren ist eine Wiederverwendbarkeit einzelner Teile der Datei nicht möglich. Wäre Wiederverwendbarkeit möglich, könnte man z.B. das Nachrichtenelement `message` für mehrere unabhängige Web Services benutzen ohne dieses Element zweimal zu definieren.

`WSDL` unterstützt die Wiederverwendbarkeit durch Aufteilen der erforderlichen Elementdefinitionen auf mehrere Dateien. Die Referenz zwischen den Dateien kann durch ein `import`-Element hergestellt werden. Ein wenig Vorsicht ist geboten, damit keine Situation eintritt, in der ein Element mehrmals in verschiedenen Dateien definiert wird.

12.4. WSDL-Elemente

12.4.1. Element Definitions

Dieses Element ist das äußerste Element, das alle anderen Elemente beinhaltet. Es enthält ein Attribut `name`, das dem `definitions`-Element einen eindeutigen Namen verleiht. Das `targetNamespace`-Attribut gibt den Namensraum der Datei an und die `xmlns`-Attribute machen Angaben zu `XML`-Namensräumen. Im unten angegebenen Beispiel werden Angaben zu Namensräumen für `SOAP` mit `xmlns:soap`, einer `XSD`-Datei mit

`xmlns:xsd1`, einer `WSDL`-Datei mit `xmlns:tns` und `WSDL` selber gemacht. Die angegebene XSD-Datei ist eine Datei, in der sich Definitionen von Datentypen befinden, auf die in diesem Web Service zurückgegriffen wird.

In einer `WSDL-Datei` benötigt man öfters einen Mechanismus, um auf schon definierte Elemente zurückgreifen zu können. Dieser Mechanismus funktioniert, indem eine Referenz durch `tns:elementname` aufgebaut wird. Tns ist eine Abkürzung, die für `thisnamespace` steht. Diese Bezeichnung ist nicht zwingend vorgeschrieben, hat sich allerdings als Konvention durchgesetzt. Für das Funktionieren dieses Mechanismus benötigt man jetzt noch die Angabe des Namensraums für `tns`. Genau dies wird mit dem Attribut `xmlns:tns` im `definitions-Element` gemacht, wobei als Namensraum die eigene Datei angegeben wird.

```
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote.wsdl"
   xmlns:tns="http://example.com/stockquote.wsdl"
   xmlns:xsd1="http://example.com/stockquote.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
   ..........................
</definitions>
```

Quelltext 12.1: WSDL: Bsp Element Definitions

12.4.2. Element Import

Mit diesem Element werden `WSDL`-Elemente importiert, die in einer anderen Datei definiert worden sind. Dabei wird die Adresse der zu importierenden Datei mit einem Namensraum assoziiert.

```
<definitions ......................>
   <import namespace="http://example.com/stockquote/schemas"
      location="http://example.com/stockquote/stockquote.xsd" />
      ..................
</definitions>
```

Quelltext 12.2: WSDL: Bsp Element Import

12.4.3. Element Types

Das Element `types` enthält Definitionen von Datentypen. Bevorzugt wird dabei das `xsd`-Typsystem aufgrund seiner Plattformunabhängigkeit. In dem Element `types` wird zunächst ein Element `schema` benötigt, dass nochmals Angaben zu Namensräumen macht. Im `schema`-Element können nun Elemente definiert werden. Sie haben einen eindeutigen Namen und müssen genau einem Typ entsprechen. Dieser Typ kann auch ein komplexer Typ sein. In Bsp 12.3 werden zwei Elemente namens `TradePriceRequest` und `TradePrice` definiert. Das Erstere enthält ein Element namens `tickerSymbol` das vom Datentyp `string` ist, das Letztere ein Element namens `price` das von Datentyp `float`

ist. In Bsp 12.4 wird ein komplexer Typ gezeigt der aus Elementen unterschiedlicher Typen besteht.

```
<types>
   <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
         <complexType>
            <all>
               <element name="tickerSymbol" type="string"/>
            </all>
         </complexType>
      </element>
      <element name="TradePrice">
         <complexType>
            <all>
               <element name="price" type="float"/>
            </all>
         </complexType>
      </element>
   </schema>
</types>
```

Quelltext 12.3: WSDL: Bsp Element Types

```
<complexType name="Item">
   <all>
      <element name="quantity" type="int"/>
      <element name="product" type="string"/>
      <element name="price" type="string"/>
   </all>
</complexType>
```

Quelltext 12.4: WSDL: Bsp Komplexer Typ

12.4.4. Element Message

Das Element `message` besteht aus mehreren logischen Teilen. Diese logischen Teile sind die `part`-Elemente innerhalb des `message`-Elements. Sowohl das `message`-Element wie auch die `part`-Elemente erhalten einen eindeutigen Namen durch das `name`-Attribut. Die `part`-Elemente enthalten ein `element`-Attribut, in dem eine Referenz zu einem Datentyp hergestellt wird, und zwar zu einem jener Datentypen, die im vorangegangenen `types`-Element definiert worden sind.
Im folgenden Beispiel werden zwei Nachrichten mit jeweils einem `part`-Element definiert. Das erste `part`-Element stellt eine Beziehung mit dem Element namens `TradePrice-Input` her. `TradePriceInput` enthielt in der obigen Definition ein Element vom Datentyp `string`. Dementsprechend wird dem zweiten `part`-Element nun der Typ `float` zugeordnet.

```
<message name="GetLastTradePriceInput">
   <part name="One" element="tns:TradePriceRequest"/>
</message> <message name="GetLastTradePriceOutput">
   <part name="Two" element="tns:TradePrice"/>
</message>
```

Quelltext 12.5: WSDL: Bsp Element Message

12.4.5. Element PortType

Dieses Element besteht aus einem Set von Empfangs- und Sende-Operationen.

Bisher wurde nur gesagt wie viele Nachrichten ausgetauscht werden sollen und von welchem Datentyp sie sind. Jetzt wird einer Nachricht die Funktion einer Sende- oder Empfangsoperation zugeordnet.

Dabei gibt es vier primitive Kommunikationsformen:

- **One-Way**: Empfangen einer Nachricht
- **Request-Response**: Empfangen einer Nachricht, dann Senden einer Nachricht
- **Solicit-Response**: Senden einer Nachricht, dann Empfangen einer Nachricht
- **Notification**: Senden einer Nachricht

Das Empfangen einer Nachricht wird dabei als `input`-Element dargestellt, das Senden als `output`-Element. Die `input`- und `output`-Elemente werden zusammen in das `operation`-Element eingefügt, das ein Unterelement des `portType`-Elements ist. Sowohl das `portType`-Element wie auch das `operation`-Element besitzten einen eindeutigen Namen der durch das `name`-Attribut verliehen wird. Auch hier wird wieder der Mechanismus des Referenzierens benutzt. Dabei wird ein oben definiertes Nachrichtenelement genommen und als `input`- oder `output`-Nachricht spezifiziert. Zudem gibt es noch ein optionales `fault`-Element, das ebenfalls innerhalb des `operation`-Elements eingefügt werden kann. Es kann für den Fall eines Fehlers beim Austausch von Nachrichten herangezogen werden.

Die vier Möglichkeiten werden in den kommenden Beispielen aufgezeigt.

```
1 <portType name="StockQuotePortType">
2    <operation name="GetLastTradePrice">
3       <input message="tns:GetLastTradePriceInput"/>
4    </operation>
5 </portType>
```

Quelltext 12.6: WSDL: Bsp Element PortType: One-Way

```
1 <portType name="StockQuotePortType">
2    <operation name="GetLastTradePrice">
3       <input message="tns:GetLastTradePriceInput"/>
4       <output message="tns:GetLastTradePriceOutput"/>
5    </operation>
6 </portType>
```

Quelltext 12.7: WSDL: Bsp Element PortType: Request-Response

```
<portType name="............">
   <operation name="...........">
      <output message="............"/>
      <input message="............"/>
   </operation>
</portType>
```

Quelltext 12.8: WSDL: Bsp Element PortType: Solicit-Response

```
<portType name=".......">
   <operation name="............">
      <output message="................."/>
   </operation>
</portType>
```

Quelltext 12.9: WSDL: Bsp Element PortType: Notification

12.4.6. Element Binding

Im Element `binding` wir die Bindung zu einem Protokoll und zu einem Nachrichtenformat hergestellt. Als Protokolle kommen nach derzeitiger `WSDL`-Spezifikation `SOAP 1.1`, `HTTP Get/Post` und `MIME` in Betracht. Jedes `binding`-Element muss die Beziehung zu genau einem Protokoll herstellen.

In jedem `binding`-Element befindet sich ein `operation`-Element, dass wiederum ein `input`- und `output`-Element und ein `fault`-Element enthalten kann. Auch hier gilt, dass das `binding`- und das `operation`-Element wieder einen eindeutigen Namen erhalten. Diesmal wird die Referenz zu einem definierten `portType` im `type`-Attribut vorgenommen. Dieses `type`-Attribut ist im `binding`-Tag enthalten.

Im folgenden Beispiel sind alle Stellen, an denen konkrete Protokollinformationen stehen durch `extensibility`-Elemente gekennzeichnet, die im folgenden Kapitel vorgestellt werden.

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <-- extensibility element (1) -->
   <operation name="GetLastTradePrice">
      <-- extensibility element (2) -->
      <input>
         <-- extensibility element (3) -->
      </input>
      <output>
         <-- extensibility element (4) -->
      </output>
      <fault>
         <-- extensibility element (5) -->
      </fault>
   </operation>
</binding>
```

Quelltext 12.10: WSDL: Bsp Element Binding

12.4.7. Element Port

Das Element `port` spezifiziert für jedes `binding`-Element genau eine Netzwerkadresse. Möchte man mehrere Netzwerkadressen für die Benutzung eines Web Service anbieten, so kann man mehrere `port`-Elemente definieren. Im Bsp 12.11 steht das `extensibility`-Element für die Stelle, an der die Netzwerkadresse, abhängig vom verwendeten Protokoll, angegeben wird.

Wie gewohnt hat auch das `port`-Element ein `name`-Attribut. Die Referenz zu dem oben definierten `binding`-Element wird hier durch das `binding`-Attribut hergestellt.

```
<port name="StockQuotePort"
   binding="tns:StockQuoteSoapBinding">
   <-- extensibility element -->
</port>
```

Quelltext 12.11: WSDL: Bsp Element Port

12.4.8. Element Service

Das `service`-Element fasst alle `port`-Elemente zu einer Gruppe zusammen. Dabei dürfen die Ports nicht miteinander kommunizieren. Der Output eines Ports darf nicht der Input eines anderen Ports sein.

```
<service name="StockQuoteService">
   <port name="StockQuotePort1"
         binding="tns:StockQuoteSoapBinding1">
      ................
   </port>
   <port name="StockQuotePort2"
         binding="tns:StockQuoteSoapBinding2">
      ................
   </port>
</service>
```

Quelltext 12.12: WSDL: Bsp Element Service

12.4.9. Element Documentation

Dieses Element ist für den Benutzer gedacht, um den Quellcode zu dokumentieren. Es kann beliebig oft und an beliebiger Stelle eingefügt werden.

```
<service name="StockQuoteService">
   <documentation>
      Hier wird die Adresse der Bindung angegeben
   </documentation>
   <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
      ................
   </port>
</service>
```

Quelltext 12.13: WSDL: Bsp Element Documentation

12.5. SOAP-Bindung

Jede Bindung benötigt spezielle Elemente, die sich nach den Anforderungen der Bindung richten. Die aktuelle WSDL-Spezifikation beinhaltet diese speziellen Elemente für SOAP 1.1, HTTP Get/Post und MIME.

Alle Elemente werden in der folgenden Form eingefügt:

```
<soap:elementname ................. />  bei SOAP
<http:elementname ................. />  bei HTTP Get/Post
<mime:elementname ................. />  bei MIME
```

Quelltext 12.14: WSDL: SOAP Bindung

In diesem Kapitel wird nur die SOAP-Bindung behandelt, da sie im Rahmen dieses Projekts genutzt werden soll.

12.5.1. Element soap:binding

Dieses Element verdeutlicht die Bindung zum SOAP-Format mit seiner Struktur aus Envelope, Header und Body. Es wird in das binding-Element eingefügt.

```
<binding .... >
   <soap:binding transport="uri" style="rpc | document"/>
      ..............
</binding>
```

Quelltext 12.15: WSDL: Element soap:binding

Das transport-Attribut gibt den Transportmechanismus an. Hier kann z.B. HTTP, SMTP oder FTP verwendet werden. Das style-Attribut gibt Aufschluss darüber, ob eine RPC-orientierte oder eine dokument-orientierte Operation benutzt wird. Wenn keine Angabe gemacht wird, dann wird von einer dokument-orientierten Operation ausgegangen.

12.5.2. Element soap:body

Das soap:body-Element beinhaltet mehrere Attribute, mit denen das Erscheinungsbild der Nachricht im Body der SOAP-Nachricht beeinflusst werden kann. Es liegt im input-, bzw. output-Element innerhalb des operation-Elements im binding-Konstrukt.

Das Attribut parts ist optional. Wird es weggelassen, dann sind alle parts, also alle Nachrichtenteile, die zuvor im message-Element definiert worden sind angesprochen. Es kann also für jedes part-Element ein eigenes soap:body-Element definiert werden, wobei nmtokens der Name des parts ist.

Das use-Attribut gibt an, ob ein part eine encoding-Regel benutzt oder nicht. Das namespace-Attribut gibt wieder einen Namensraum an, und das encodingStyle-Attri-

but besteht aus einer Liste von URIs die Encodings repräsentieren, die in der Nachricht verwendet werden.

```
<binding ..............>
   <soap:binding ...........>
   <operation ..................>
      ........
      <input>
         <soap:body parts="nmtokens" use="literal | encoded"
            encodingStyle="uri-list" namespace="uri" />
      </input>
      <output>
         <soap:body parts="nmtokens" use="literal | encoded"
            encodingStyle="uri-list" namespace="uri" />
      </output>
      ........
   </operation>
</binding>
```

Quelltext 12.16: WSDL: Element soap:body

12.5.3. Element soap:header

Mit dem soap:header-Element ist es möglich, den Header der SOAP-Nachricht zu beeinflussen. Im Header können dabei beliebige Informationen wie z.B. eine User-ID eingefügt werden. Es ist am selben Platz gelegen wie das soap:body-Element. Die Attribute sind dieselben wie die des soap:body-Elements. Zusätzlich gibt es noch das message-Element, das den Namen der message angibt, die das part-Element enthält.

Neben der soap:header-Definition gibt es noch eine soap:headerfault-Definition die an derselben Stelle gelegen ist, die gleichen Attribute hat und für eine Fehlerbehandlung zuständig ist.

```
<binding ..............>
   <soap:binding ...........>
   <operation ..................>
      ........
      <input>
         <soap:body ............/>
         <soap:header message="qname" part="nmtokens"
            use="literal | encoded" encodingStyle="uri-list"
            namespace="uri" />
      </input>
      ........
   </operation>
</binding>
```

Quelltext 12.17: WSDL: Element soap:header

12.5.4. Element soap:operation

Das soap:operation-Element stellt Informationen für die Operation als Ganzes zur Verfügung. Es befindet sich ebenfalls im binding-Element.

Das Attribut `style` ist schon bekannt aus der Definition von `soap:binding`.
Das `soapAction`-Attribut spezifiziert einen Wert für den `SOAP-Action-Header` bei einer `SOAP-HTTP`-Anfrage. Dieser Wert, er muss kein `URI`, sondern kann ein beliebiger String sein, kann als Identifizierung einer Absicht verstanden werden, die eine `SOAP-HTTP`-Anfrage verfolgt. Nützlich kann dies z.B. für Firewalls sein, um bestimmte `SOAP`-Anfragen zu filtern.

```
<binding ..............>
   <soap:binding ...........>
   <operation ..................>
      <soap:operation soapAction="uri" style=rpc | document" />
      <input>
         ............
      </input>
      ........
   </operation>
</binding>
```

Quelltext 12.18: WSDL: Element soap:operation

12.5.5. Element soap:address

Das `soap:address`-Element spezifiziert die Adresse des Ports. Es wird im `port`-Element eingefügt, das ein Subelement vom `service`-Element ist. Die genaue Art der Adresse wird in dem `location`-Attribut angegeben. Dies können `HTTP`-, `SMTP`-, `FTP`-Adressen oder auch andere sein.

```
<service ..................>
   <port .... >
      <soap:address location="http://example.com/stockquote"/>
   </port>
</service>
```

Quelltext 12.19: WSDL: Element soap:address (1)

oder

```
<service>
   <port ......>
      <soap:address location="mailto:subscribe@example.com"/>
   </port>
</service>
```

Quelltext 12.20: WSDL: Element soap:address (2)

12.6. Erzeugung von WSDL-Dateien

In vielen Fällen wird ein Web Services-Entwickler keine `WSDL`-Datei schreiben, sondern auf andere Programmiersprachen wie `Java` zurückgreifen. Grund kann z.B. der relativ große Quellcode einer `WSDL`-Datei verglichen mit `Java` sein. Zum Erzeugen einer `WSDL`-Datei wird dann ein Toolkit benutzt, das aus der `Java`-Datei die `WSDL`-Datei generiert. Im Anschluß muss dann die `WSDL`-Datei evtl. noch manuell geändert werden um z.B. eine Netzwerkadresse anzugeben. Im Rahmen dieses Projektes wird dazu ApacheSOAP und AXIS genutzt.

12.7. Referenzen

- http://www.w3c.org/2002/ws
- http://www.w3.org/TR/wsdl
- http://www.geolizer.de/
- http://www.w3schools.com

13. UDDI

13.1. Einführung

Bisher gab es keine Möglichkeit zum Auffinden von Diensten, welche von verschiedenen Firmen zur Verfügung gestellt werden. Eine Suche über einen Webkatalog, wie z.B. Yahoo ist unzureichend, denn er bietet nur eine Kategorisierung der Firmen in Form von Firmenbeschreibungen und URL, aber keine Beschreibung der Dienste, die das Unternehmen bietet. Die Dienste können allgemeiner Natur sein, oder Web Services, und gerade für letztere gab es bisher keinen Katalog, der dieses auflistet, beschreibt und eine Link zu seiner technischen Anbindung offeriert. UDDI füllt genau diese Lücke und bietet einen plattformunabhängigen offenen Rahmen, um Services zu beschreiben (description), Unternehmen zu finden (discovery) und die Services zusammenzuschließen (integration).

UDDI kann man also als eine Art „Gelbe, Grüne oder Weiße Seiten“ verstehen, in denen Unternehmen ihre Dienste, die sie als Web–Services zur Verfügung stellen, registrieren können.

Die erste Spezifikation entstand innerhalb von ca. 6 Monaten aus der Zusammenarbeit von Microsoft, IBM und Ariba im September 2000. Mai 2001 wurde der erste UDDI Server gestartet, dieser wurde nur noch von Microsoft und IBM entwickelt. Später beteiligten sich weitere Firmen an diesem Projekt. Der UDDI Standard umfasst bis heute mehr als 300 Unternehmen, u.a. Ariba, Microsoft, SAP, Compaq/HP, Fujitsu, IBM, Verisign, Oracle und SUN. Die zurzeit aktuellste Version ist UDDI v3.0 Data Structure Reference. Der Zugriff auf ein UDDI Registry erfolgt über ein Web Interface oder ein Tool. Die Registry wird via SOAP in Form einer so genannten SOAP[1] Message angesprochen. Die Einträge werden über eine UBR[2] erstellt. UBRs werden von verschiedenen Firmen angeboten und funktionieren unabhängig voneinander. Die Funktionsweise erinnert an die Internet domain name services (DNS).

Die Registries werden regelmäßig, mindestens täglich, miteinander abgeglichen.

Um ein Unternehmen einzutragen, muss man sich bei einer Registry registireren (z.B.: http://uddi.hp.com oder http://uddi.microsoft.com). Man kann nur dort, wo man sich registriert hat, die Daten ändern. Es gibt öffentliche und interne Registries. Interne wer-

[1] SOAP – Simple Object Access Protocol
[2] UBR – UDDI Business Registry

den nur innerhalb eines Unternehmens benutzt, um die eigene Verwaltung zu erleichtern und gelangen nicht nach außen.

13.1.1. Encodieren von Informationen

Es gibt in UDDI drei Arten von Informationen:

- Die **Weißen Seiten** enthalten den Firmennamen und Kontaktdetails, den Webseitennamen und identifizierende Nummern.
- Die **Gelben Seiten** sind das Branchenverzeichnis. Sie enthalten die Branche der Firma, den Sitz, ihre Produkte und weitere Kategorisierungen (Taxonomies) der Firma.
- Die **Grünen Seiten** enthalten die technischen Daten über die Services. Zum Beispiel wie man mit Ihnen interagiert, Geschäftsprozessdefinitionen usw.

13.2. UDDI Datentypen

13.2.1. Struktur eines typischen UDDI Eintrags

```
<businessEntity businessKey= "..." operator="..." authorizedName="...">
   <name> ... </name>
   <discoveryURLs> ... </discoveryURLs>
   <description> ... </description>
   <contacts> ... </contacts>
   <businessServices>
      <businessService serviceKey="..." businessKey="...">
         ...
         <bindingTemplates>
            <bindingTemplate serviceKey="..." bindingKey="...">
               ...
               <accessPoint URLType="..."> ... </accessPoint>
               <tModelInstanceDetails>
                  <tModelInstanceInfo tModelKey="...">
                     ...
               </tModeInstaceDetails>
            </bindingTemplate>
         </bindingTemplates>
         ...
      </businessService>
   </businessServices>
   <identifierBag> ... </identifierBag>
   <categoryBag> ... </categoryBag>
</businessEntity>
```

Quelltext 13.1: UDDI: Typischer UDDI Eintrag

13.2.2. businessEntity

In der businessEntity sind die generellen Unternehmensinformationen encodiert. Die businessEntity bildet das Wurzelobjekt. Es beinhaltet alle Informationen über das Unternehmen, dessen Web Services beschrieben werden.

Weitere Strukturen der `businessEntity` werden im Folgenden erklärt:

- **businessKey**: eindeutiger identifizierender Schlüssel des Eintrages (UUID)
- **operator**: Name des UBR Operators, also des Unternehmens, das die UBR zur Verfügung stellt.
- **discoveryUrls**: Eine Liste von URLs, die auf weitere Dokumente verweist, die den Dienst beschreiben.
- **name**: Ein vorgeschriebenes sich wiederholendes Element, für von Menschen lesbare Bezeichnung für das beschriebene Unternehmen. Die Strings können über `xml:lang` auf die Sprache verweisen, in der sie verfasst sind.
- **description**: Eine oder mehrere kurze Beschreibungen des Unternehmens, dass seine Dienste publiziert.
- **contacts**: Eine Liste von Ansprechpartnern im beschriebenen Unternehmen.
  Die Kontakte selbst werden ebenfalls durch einen abstrakten Datentyp Contact repräsentiert. Dieser besteht aus:

  < **contacts** >
  – personName
  – useType
  – description
  – phone
  – email
  – address

- **businessServices**: Eine Liste von Beschreibungen der angebotenen Dienste.
- **identifierBag**: Eine Liste von Name–Wert Paaren, die genutzt werde können, um Schlüssel anzugeben, die in einer Suche verwendet werden können, z.B. die Handelsregisternummer.
- **categoryBag**: Eine Liste von Name–Wert Paaren, die verwendet werden, um das businessEntity mit spezifischen Kategorisierungsinformationen zu versehen

Die wichtigsten Elemente einer `businessEntity`–Struktur sind in der folgenden Struktur aufgeführt:

< **businessEntity** >
\+ businessKey
\+ operator
\+ name
– discoveryURLs
– description
– contacts
– businessServices
– identifierBag
– categoryBag

Dabei steht das + für obligatorische und das – für optionale Attribute.

13.2.3. businessService

Eine businessEntity kann mehrere businessServices enthalten. Das Element beinhaltet Informationen über die einzelnen Web Services, welche angeboten werden. Die Beschreibung beinhaltet, um welche Art von Web Service es sich handelt, in welche Kategorien von Taxonomien er gehört und wie die Anbindung an den Dienst erfolgt. Auch dieser Struktur wird ein eindeutiger Schlüssel, also eine UUID[3] zugeordnet. Dieser wird im serviceKey gespeichert. Im businessKey steht die UUID in der businessEntity, der Hauptstruktur. Dies ist sinnvoll, um den Zusammenhalt abzusichern. Der UUID ist ein eindeutiger Identifizierer des Eintrages. Dieser wird mit einem anspruchsvollen Algorithmus (ISO/IEC 11578:1996) erzeugt, der hinreichend sicherstellt, dass niemals zwei gleiche Schlüssel generiert werden.

Weitere Strukturen des businessService werden im Folgenden erklärt:

- **businessKey**: Eine Wiederholung der UUID des Eintrags, die nur dann notwendig ist, wenn die Struktur einzeln verarbeitet wird, also nicht in einem vollständigen business–Entity–Element enthalten ist.
- **serviceKey**: Der eindeutige Schlüssel des businessService–Elements. Dieses Element wird bei einem neuen Eintrag leer übergeben und vom Operator dann gefüllt. Beim Update wird die jeweilige ID mit angegeben.
- **name**: Mit den erforderlichen name–Elementen können Namen für den Dienst angegeben werden, die für Menschen bestimmt sind.
- **description**: Hier können entsprechend dem name–Element Kurzbeschreibungen des Dienstes abgelegt werden.

[3]UUID – Universal Unique IDentifier

- **bindingTemplate**: Diese Struktur beinhaltet die technische Dienstbeschreibung zu einer gegebenen Dienstefamilie.
- **categoryBag**: Hier kann eine Liste von Name–Wert–Paaren angegeben werde (Taxonomierung durch Kategorien), die für eine Suche entsprechende Klassifikationsinformationen, wie geographische Informationen, Industrie– oder Produktklassifizierungen, beinhaltet. Zum Beispiel:
  - `NAICS` (North American Industry Classification System 1997 Release)
  - `UNSPSC` (Universal Standard Products and Services Classification)
  - `ISO 3166 Geographic Taxonomy` – geographische Regionen, Länderkennungen etc.
  - `D-U-N-S` Number Identification System
  - Thomas Register Supplier Identification Code System

Die wichtigsten Elemente einer `businessService`–Struktur sind in der folgenden Struktur aufgeführt:

< **businessService** >
\+ name
\+ serviceKey
\+ bindingTemplates
\+ categoryBag
– businessKey
– description

13.2.4. bindingTemplate

Das bindingTemplate enthält die technische Dienstbeschreibung. Die Templates beschreiben den technischen Zugangspunkt (Kommunikationsschnittstelle) zum beschriebenen Dienst. Einer einzelnen businessService–Struktur können mehrere bindingTemplates zugeordnet werden.

Weitere Strukturen des bindingTemplate werden im Folgenden erklärt:

- **bindingKey**: enthält die UUID dieses bindingTemplates. Dies erlaubt mit den einzelnen Elementen losgelöst voneinander umzugehen und die Aktualisierung einzelner Elemente.
- **serviceKey**: enthält die UUID, die auf das zugeordnete businessService–Element verweist. Das Element ist nur dann notwendig, wenn die Struktur auf ohne das umschließende Element verwendet wird.

- **description**: Auch dieses Element verfügt über eine Kurzbeschreibung der Funktionalität.
- **accessPoint**: Dieses Element beinhaltet eine textuelle Adresse, die für den Zugriff auf den Web Service notwenig ist. Diese Adresse kann ein URL oder eine Email–Adresse oder eine andere Beschreibung eines Kommunikationspunktes sein. (mailto, http, https, ftp, fax, phone)
- **hostingRedirectory**: Falls kein accessPoint angegeben ist, muss der hostingRedirector angegeben werden. Wenn dieser Parameter vorhanden ist, dann ist dieses bindingTemplate nicht länger gültig und es sollte das Template geladen werden, auf das der hostingRedirector verweist.
- **tModelInstanceDetails**: Diese Struktur ist eine Liste von tModelInstanceInfo–Elementen, welche auf tModels verweisen.

Die Struktur hat folgende Elemente:

< **bindingTemplate** >
\+ bindingKey
\+ tModelInstanceDetails
+− accessPoint
+− hostingRedirectory
− serviceKey
− description

13.2.5. tModel

Das tModel ist eine technische Spezifikation der Web Service Schnittstelle. Es dient als Mechanismus, um Metadaten, also Daten über Daten in einer sehr generellen Form über einen Web Service auszutauschen, wie die Web Service Beschreibung oder einen Zeiger auf eine WSDL Datei. Das tModel enthält nicht die technische Beschreibung des Web Services selbst, sondern nur den Link dahin. Das tModel ist genauso eine Hauptstruktur wie die businessEntity.
Der Aufbau eines tModels sieht folgendermaßen aus:

```
<tModel modelKey="..." authorizedName="..." operator="...">
   <name>...</name>
   <description> ... </description>
   <overviewDoc> ... </overviewDoc>
   <categoryBag>
      ...
   </categoryBag>
   <identifierBag>
      ...
   </identifierBag>
</tModel>
```

Quelltext 13.2: UDDI: tModel

Weitere Strukturen des tModel werden im Folgenden erklärt:

- **tModelKey**: ist die UUID des tModels.
- **authorizedName**: Der Name des Erstellers des tModels, ebenfalls wie in businessEntity. Diese Information wird normalerweise automatisch vom Operator des UDDI–Verzeichnisses erzeugt und eingefügt.
- **name**: Name des tModel Eintrags. Muss gesetzt sein.
- **description**: Beschreibung des Eintrags.
- **overviewDoc**: Dieses Element wird benutzt um eine Referenz auf eine externe Quelle mit Informationen oder Instruktionen zum tModel Typ zu geben.
- **categoryBag**: Eine Liste von Name–Wert Paaren, die Klassifizierungsinformationen über das tModel beinhalten. Die Informationen werden ebenfalls im Zuge einer Suche nach tModel–Elementen verwendet.

Die Struktur hat folgende Elemente:

< **tModel** >
\+ name
\+ tModelKey
\+ authorizedName
– description
– categoryBag
– overviewDoc
– identifierBag

13.2.6. publisherAssertion

Die publisherAssertion ist eine Beziehungstruktur zwischen zwei businessEntity Strukturen.

Weitere Strukturen der publisherAssertion werden im Folgenden erklärt:

- **fromKey**: enthält die UUID zum ersten Unternehmen.
- **toKey**: enthält die UUID zum zweiten Unternehmen.
- **keyedReference**: enthält den Typ der Beziehung.

Die Struktur hat folgende Elemente:

< **publisherAssertion** >
\+ fromKey
\+ toKey
\+ keyedReference

13.2.7. Zusammenhänge der Datentypen

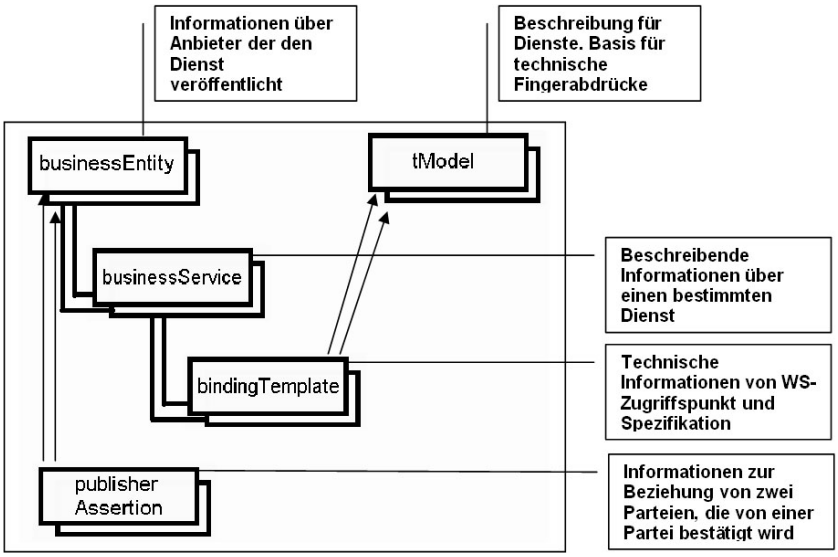


Abbildung 13.1.: UDDI: Struktur der UDDI–Datentypen

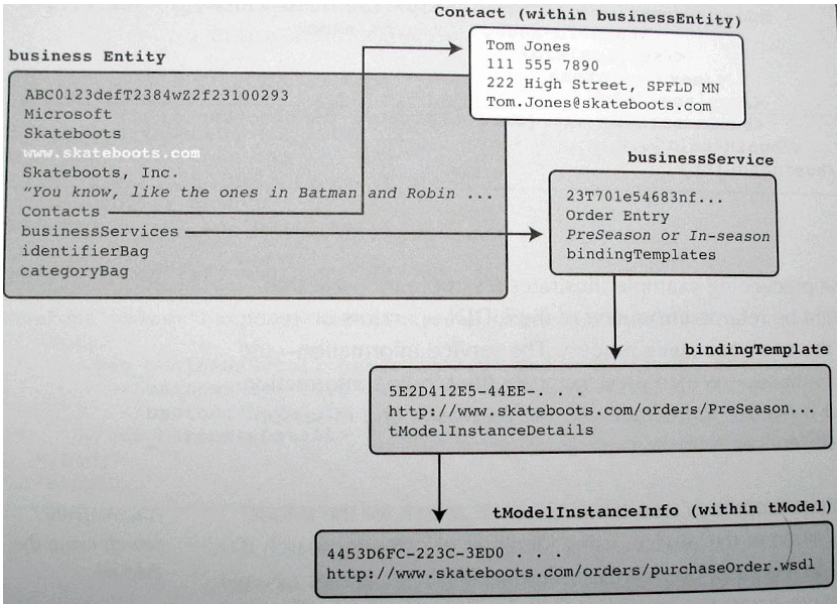


Abbildung 13.2.: UDDI: Praxisbeispiel Struktur der UDDI–Datentypen

13.2.8. Die UDDI – API

Für den Zugriff auf ein UDDI–konformes Verzeichnis sind Programmierschnittstellen definiert. Diese sind als SOAP Schnittstellen ausgelegt. Der UDDI Standard definiert zwei zentrale Komponenten, einmal die Struktur für ein universelles Verzeichnis in dem Unternehmen sich selbst und die angebotenen Dienstleistungen beschreiben und ablegen können und die Funktionen mit deren Hilfe Daten gesucht und modifiziert werden können. Hierbei handelt es sich um die beiden Schnittstellen, die in der UDDI (Programmierschnittstelle) API definiert sind, Publish und Inquire.

Die Publizierungsschnittstelle regelt die Interaktion zwischen dem Verzeichnis und Programmen von Anbietern. Hierzu stellt sie Funktionen zur Speicherung und zur Aktualisierung von Einträgen bereit. Ein Anbieter muss hierbei zunächst zu einem Operator eines geeigneten Verzeichnisses Verbindung aufnehmen.

Für die Gruppe der Anwender von Web Services ist eine Reihe von Funktionen definiert, welche den Zugriff auf die im Verzeichnis gespeicherten Informationen erlaubt. Die Abfrageschnittstelle definiert dabei drei prinzipielle Abfragemuster. Das erste dieser Muster dient der Suche bzw. dem Browsen in den Verzeichnisinhalten. Hierzu liefern Funktionen wie die find_xx–Aufrufe Überblickinformationen zu den im Verzeichnis vorhandenen Informationen zurück. Sind auf der Basis dieser Informationen die Ergebnisse weiter eingegrenzt, so kann mit dem nächsten Abfragemuster, dem so genannten, drill–down–Muster auf Details von vorausgewählten oder bekannten Einträgen zugegriffen werden. Hierbei kommen die get_xx–Aufrufe zum Einsatz. Ist dann schlussendlich der gewünschte Business–Partner mit dem gesuchten Dienstangebot gefunden, wird mit Hilfe des Aufrufmusters, dem letzten der drei Muster, auf die gewünschte Funktionalität zugegriffen. Hier kommen die bindingTemplates ins Spiel, die beschreiben wie mit einem gegebenen Dienst interagiert werden kann. Die entsprechende Funktion der Abfrageschnittstelle ist get_bindingDetails.

Im Folgenden die einzelnen Funktionen mit Erläuterungen:

13.2.9. Publishing API

Das Publishing Interface definiert 16 Operationen, um Informationen in der Registry zu verwalten:

- **get_authToken**: Erhalt eines Authorisationstokens (das Äquivalent zu einem Login)
- **discard_authToken**: Fallenlassen eines Authorisationstokens
- **save_business, save_service, save_binding, save_tModel**: Neu speichern oder Updaten der Strukturen

- **delete_business, delete_service, delete_binding, delete_tModel**: Löschen der Strukturen
- **add_publisherAssertions**: Zufügen von Relationsbeziehungen zwischen Unternehmen
- **delete_publisherAssertions**: Löschen der Relationsbeziehungen zwischen Unternehmen
- **get_publisherAssertions**: Liefert eine Liste aller Unternehmensbeziehungen
- **set_publisherAssertions**: Speichert eine Liste aller Beziehungen (Überschreibt eine bereits existierende Liste)
- **get_assertionStatusReport**: Liefert einen Status Report, der die Unternehmensbeziehungen und Status Information enthält
- **get_registeredInfo**: Liefert eine Zusammenfassung der Information

13.2.10. Inquiry API

Das Inquire Interface definiert 10 Operationen, um die Registry zu durchsuchen:

- **find_business**: Zum Auffinden von Informationen über Unternehmen
- **find_service**: Zum Auffinden von spezifischen Services
- **find_binding**: Zum Auffinden von spezifischen Bindungen innerhalb eines businessService
- **find_tModel**: Zum Auffinden von tModel Informationsstrukturen
- **find_relatedBusinesses**: Zum Auffinden von Informationen über businessEntity Registrierungen, die in Beziehung stehen zur UUID der übergebenen businessEntity
- **get_businessDetail**: Zum Erhalten der vollständigen Unternehmensinformation
- **get_businessDetailExt**: Zum Erhalten erweiterter Informationen über ein Unternehmen
- **get_bindingDetail**: Zum Erhalten der vollständigen Information über ein bindingTemplate
- **get_serviceDetail**: Zum Erhalten aller Details eines businessService
- **get_tModelDetail**: Zum Erhalten aller Details eines tModels

13.3. Die Funktionen von UDDI im Überblick

Die allgemeine Funktionsweise für UDDI lässt sich gut im Dreieck der beteiligten Parteien verdeutlichen. Diese sind zum einen der Anbieter irgendeines Web–Dienstes, den ein anderer Anwender oder Nutzer in Anspruch nehmen will. Damit diese beiden Parteien zusammenfinden, gibt es ein Verzeichnis, das dem Anbieter ermöglicht seine Dienste zu veröffentlichen und dem Nutzer ein Suchen nach Diensten und Anbietern erlaubt. Für das Verzeichnis sind dabei die Struktur der Daten und der Zugriff auf diese von großer Wichtigkeit. Die beschriebenen Zusammenhänge sind in folgender Grafik dargestellt.

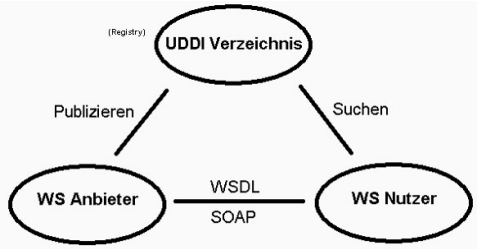


Abbildung 13.3.: UDDI: Funktionen von UDDI

Wie der Abbildung zu entnehmen ist, definiert UDDI zwei grundsätzliche Arten von Zugriffsfunktionen bzw. Schnittstellen. Eine dient dem Anbieter eines Dienstes zum Veröffentlichen von Informationen über seine Business–Tätigkeit und die angebotenen Dienstleistungen. Dieses wird in der Abbildung durch den Pfeil "Publizieren"dargestellt. Natürlich stellt das Publizieren von Informationen über ein Unternehmen oder einen Dienst eine bidirektionale Kommunikation dar und kein einseitiger Austausch von Informationen.

Die andere große Schnittstelle ist das Suchen bzw. Abfragen von Informationen, was hauptsächlich vom Nutzer von Web Services in Anspruch genommen wird. Hierbei ist zu beachten, dass Nutzer von Web Services hierbei im Normalfall nicht direkt Personen sondern Anwendungen sind.

Hat ein Nutzer einen passenden Dienst eines registrierten Anbieters im Verzeichnis gefunden, so erhält er eine Referenz zu der Beschreibung des Dienstes. Hier kommt dann die Beschreibungssprache WSDL ins Spiel. Auf diese Beschreibung kann nun per Internet zugegriffen werden und Details über den Zugriff und der Nachrichtenformate des angebotenen Dienstes abgerufen werden. In der Theorie kommt so vollständig dynamisch und automatisch eine programmgesteuerte Verbindung zwischen den beteiligten Applikationen zu Stande.

Ist jedoch per UDDI die Verbindung zwischen Anbieter und Nutzer hergestellt und hat der Nutzer über die WSDL–Beschreibung die Details für den Zugriff auf den gewünschten Dienst erhalten, dann kann die Kommunikation mit dem zentralen Baustein SOAP der Web Service–Architektur erfolgen. Die in der WSDL–Beschreibung definierten Nachrichten können ausgetauscht und so der jeweilige Dienst in Anspruch genommen werden.

13.3.1. Beispiel einer Suchanfrage und Suchantwort in SOAP

1. SOAP Request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <get_businessDetail xmlns="urn:uddi-org:api" generic="1.0">
      <businessKey>c7431408-9e9c-4186-aeb4-f5b45c74fdce</businessKey>
    </get_businessDetail>
</Body>
</Envelope>
```

Quelltext 13.3: UDDI: SOAP Request

2. SOAP Response:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <businessDetail generic="1.0" operator="Microsoft Corporation"
       truncated="false" xmlns="urn:uddi-org:api">
      <businessEntity businessKey="c7431408-9e9c-4186-aeb4-f5b45c74fdce"
         operator="Microsoft Corporation" authorizedName="Richard Roe">
        <discoveryURLs>
          <discoveryURL useType="businessEntity">
            http://uddi.microsoft.com/discovery?businesskey=c7431408-9e9c-4186-
            aeb4-f5b45c74fdce
          </discoveryURL>
        </discoveryURLs>
        <name>BizWeb</name>
        <description xml:lang="en">Business Web Services</description>
      </businessEntity>
    </businessDetail>
  </soap:Body>
</soap:Envelope>
```

Quelltext 13.4: UDDI: SOAP Response

13.4. UDDI in der Praxis

UDDI lässt sich in verschiedenen Kontexten nutzen. Aus einer reinen Anwendersicht betrachtet stellt UDDI eine Suchmaschine dar, in der unterschiedliche kommerzielle Dienste gefunden werden können. Die Verwendung von UDDI als Suchmechanismus bietet jedoch eine Reihe von Vorteilen im Vergleich zu z.B. Internet–Suchmaschinen.

Während die Informationen im Internet umstrukturiert und abstrakt abgelegt sind, da es kein verbindliches, einheitliches Format gibt, ist innerhalb einer UDDI–Registry das Format für jeden Zweck entsprechend vorgegeben. Andererseits jedoch sind die Informationen nicht unbedingt benutzungsfreundlich abgelegt, so dass eine weiterführende Aufbereitung mit dem Ziel einer besseren Lesbarkeit für den Endanwender von Nöten

ist, wenn diese direkt in einer UDDI–Registry browsen wollen. Zu diesem Zweck bieten einige Unternehmen entsprechende Portale an.

Neben der reinen Informationssuche für Anwender benutzen Entwickler die UDDI–Registry zum Publizieren und Auffinden von Diensten. Der Vorteil von UDDI liegt dabei in der Eingrenzbarkeit der Suchparameter und dem hohem Automationsgrad des Dienstes, so dass sich das Publizieren und Auffinden ohne manuelle Eingriffe realisieren lässt. So ist es möglich, dass verschiedene Dienste automatisch durch Software erkannt und angesprochen werden können. Dies macht Updates von Diensten sehr einfach, da keinerlei Anpassungsaufwand auf Seiten der Applikationen erforderlich ist.

Somit können sowohl Endanwender als auch Softwareentwickler Dienste in verschiedenen oder in der gleichen Registry ablegen und auffinden. Des Weiteren können auf diese Dienste über Endanwender–freundliche, Web–basierte Portale oder über Programmierschnittstellen zugegriffen werden.

Diesen Zusammenhang veranschaulicht die folgende Abbildung:

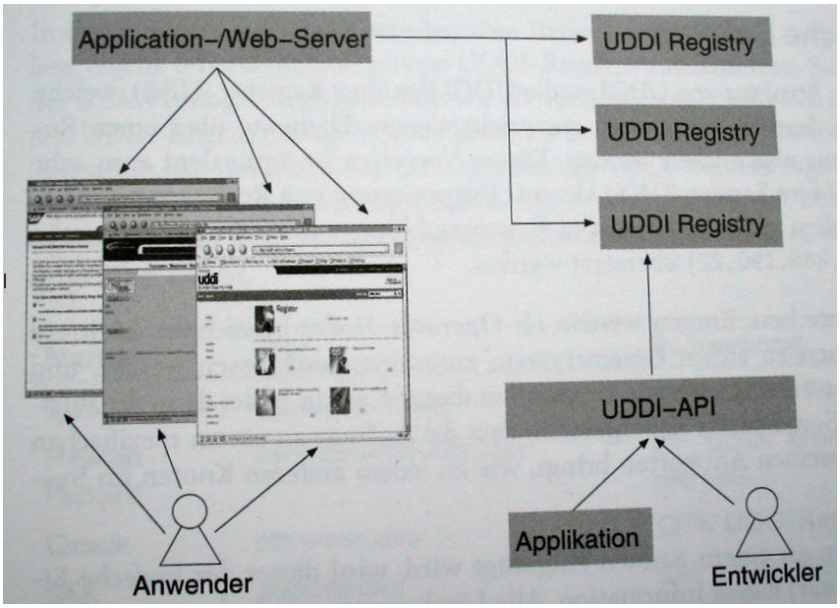


Abbildung 13.4.: UDDI: Praxisbeispiel Anwender/Entwickler UDDI

13.5. Anwendungsbeispiel

Wie bereits im theoretischen Teil erklärt wurde, sind die zentralen Aufgaben von UDDI das Suchen und Registrieren von erstellten Webservices (siehe Abb. 13.5)

Abbildung 13.5.: UDDI: Anwendungszenarien Publish und Inquire

Das Suchen und Registrieren von Webservices kann mithilfe von UDDI auf zwei unterschiedliche Arten erfolgen:

1. Über die Homepage eines der öffentlichen Verzeichnisdienstes
2. Mithilfe der einer JAVA API (z.B. UDDI4J)

Bei den Verzeichnisdiensten muss man zunächst zwischen öffentlichen und nichtöffentlichen Servern unterscheiden. Drei große Anbieter eines Verzeichnisdienstes sind:

- Microsoft → uddi.microsoft.com
- SAP → uddi.sap.com
- IBM → ibm.uddi.com

Anmerkung:
Hewlett Packard bietet ebenfalls einen Verzeichnisdienst an, der aber zum derzeitigen Zeitpunkt nicht erreichbar ist.

Nicht–öffentliche Server sind Verzeichnisdienste die zum Beispiel nur zum internen Gebrauch in einem Unternehmen aufgebaut werden.

Das Registrieren von einem WebService bei einem der genannten öffentlichen Server ist relativ trivial, da der Benutzer mit ein paar „klicks“, die in der Theorie angesprochenen Parameter eintragen kann. Wichtiger ist später das Auffinden von WebServices. Im Folgenden wird das Registrieren und Suchen auf einem öffentlichen Server sowie das Suchen eines registrierten Dienstes mithilfe der API UDDI4J erläutert.

13.6. Registrierung und Suche eines Dienstes

13.6.1. uddi.microsoft.com

Die Abbildung 13.6 zeigt die Eingangsseite des von Microsoft angebotenen Verzeichnisdienstes. Um einen Service registrieren zu können muss man sich zunächst bei Microsoft® Passport anmelden, diese Anmeldung ist kostenlos. Um einen Service zu suchen ist keine Anmeldung nötig.

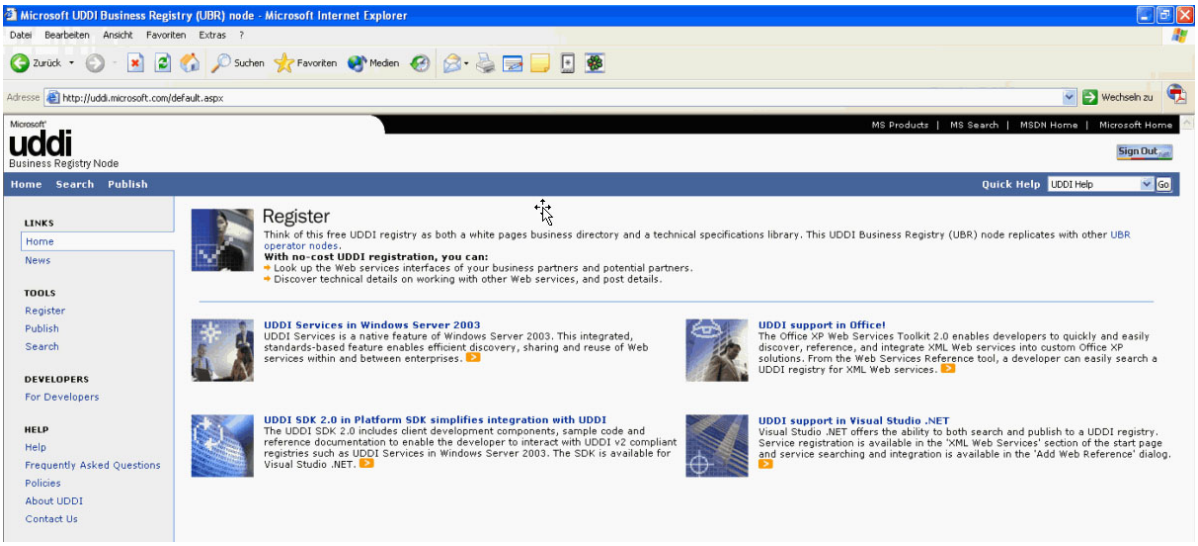


Abbildung 13.6.: UDDI: Microsoft Internetseite – uddi.microsoft.com (1)

Nach erfolgreicher Anmeldung kann nun die Registrierung von einem oder mehreren Services vorgenommen werden (siehe Abb. 13.7). Der Benutzer kann jederzeit die Angaben löschen, editieren oder einen neuen Service hinzufügen. Microsoft vergibt bei jedem hinzugefügten Element automatisch eine UUID (z.B. für den Service oder das tModel). Dies hat den Vorteil, dass später bei der Suche jedes Element eindeutig angesprochen werden kann.

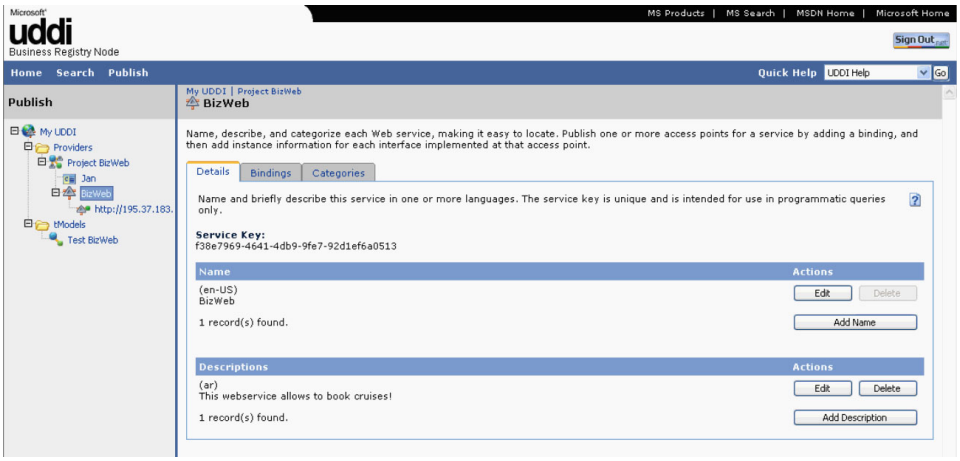


Abbildung 13.7.: UDDI: Microsoft Internetseite – uddi.microsoft.com (2)

Bei der Suche eines bereits existierenden Services hat der Benutzer folgende Möglichkeiten zu Suchen:

- Nach einer Kategorie
- Nach Name des Services
- Nach Anbieter
- Nach dem `tModel`

Abbildung 13.8.: UDDI: Microsoft Internetseite – uddi.microsoft.com (3)

Die Suchergebnisse werden am oberen linken Bildrand angezeigt, von hier aus kann der Benutzer alle relevanten Daten abrufen (wie zum Beispiel den Accesspoint des Webservices).

Da das Registrieren und Suchen von Webservices über einen öffentlichen Server, wie zum Beispiel der von Microsoft durch die einfach Menüstruktur so gut wie selbsterklärend sind, wird an dieser Stelle nicht weiter darauf eingegangen.

13.7. Suche eines Dienstes über API

In der Praxis sind zurzeit UDDI – Pakete im Einsatz, mit denen es möglich ist gewünschte Informationen eines Services einzutragen bzw. auszulesen. Bekannte API's sind die Klassenbibliotheken jUDDI, JAXR von JAVA Sun und das UDDI4J von uddi4j.org, diese wird von HP, IBM und SAP unterstützt.

Bei dem hier angeführten Beispiel wird mit der JAVA API UDDI4J gearbeitet, diese findet man zum kostenlosen Download auf der Seite:

http://uddi4j.org

Um einen Service mithilfe von JAVA Code auszulesen werden insgesamt folgende Pakete bzw. `jar` – Dateien benötigt:

- `uddi4j.jar` (uddi4j – Paket)
- `mail.jar` (javamail – Paket)
- `soap.jar` (soap – Paket)
- `activation.jar` (jaf – Paket)

Die entsprechenden `jar` – Dateien findet man meistens in dem entsprechenden `lib` – Verzeichnis des jeweiligen Paketes, diese müssen dann dem entsprechenden neuen JAVA – Projekt hinzugefügt werden.
Des Weiteren werden die entsprechenden `inquire` und `publish` Webadressen benötigt, um dem JAVA Programm mitzuteilen auf welchem Server er die gewünschte Anfrage durchführen soll. Eine entsprechende Übersicht befindet sich in der folgenden Tabelle:

Microsoft	http://uddi.microsoft.com/inquire http://uddi.microsoft.com/publish
SAP	http://uddi.sap.com/UDDI/api/inquiry/ http://uddi.sap.com/UDDI/api/publish/
IBM	http://uddi.ibm.com/beta/inquiryapi https://uddi.ibm.com/beta/publishapi

Anmerkung:
Befindet sich der Webservice auf einen nicht–öffentlichen Server, so muss entsprechend die inquire und publish Adresse dieses Verzeichnisdienstes angegeben werden.

Folgendes Programm sucht bei Microsoft nach einem Webservice, der das Wort „BizWeb" beinhaltet. Das Ergebnis wird in Form einer BusinessList ausgegeben und beinhaltet, den genauen Namen, die Service Infos, sowie den eindeutigen BusinessKey in Form der UUID.

```
1  import org.uddi4j.client.*;
2  import org.uddi4j.util.*;
3  import org.uddi4j.datatype.*;
4  import org.uddi4j.response.*;
5  import java.util.*;
6
7  public class UDDI4J {
8     public static void main(String[] args) throws Exception {
9        UDDIProxy proxy = new UDDIProxy();
10       proxy.setInquiryURL("http://uddi.microsoft.com/inquire");
11       Vector names = new Vector();
12       names.add(new Name("BizWeb"));
13       FindQualifiers findQualifiers = new FindQualifiers();
14       Vector qualifier = new Vector();
15       qualifier.add(new FindQualifier("caseSensitiveMatch"));
16       findQualifiers.setFindQualifierVector(qualifier);
17       BusinessList bl = proxy.find_business(names, null, null,
18          null, null, findQualifiers, 20);
19       Vector biv = bl.getBusinessInfos().getBusinessInfoVector();
20       for (int i = 0; i < biv.size(); i++) {
21          BusinessInfo bi = (BusinessInfo) biv.elementAt(i);
22          System.out.println(bi.getDefaultNameString());
23          System.out.println(bi.getServiceInfos());
24          System.out.println(bi.getBusinessKey());
25       }
26    }
27 }
```

Quelltext 13.5: UDDI: Test

Ausgabe des Programms:

BizWeb	(DefaultNameString)
org.uddi4j.response.ServiceInfos@1b4fad5	(ServiceInfo)
c7431408-9e9c-4186-aeb4-f5b45c74fdce	(BusinessKey)

Die Suchbegriffe werden in einem Vector „names“ gespeichert, dadurch kann der Benutzer jederzeit nach mehreren Begriffen gleichzeitig suchen. Suchwörter können über den Befehl

```
1  names.add(new Name ("Suchwort")
```

Quelltext 13.6: UDDI: Zusätzliche Suchwörter

hinzugefügt werden.

Unter 13.2.9 „Publishing API“ befindet sich ein Überblick über die möglichen Befehle, mit denen Informationen eingetragen werden können, bzw. mit denen Services eingetragen werden können.

13.8. Anfrage in SOAP

Sowohl das Eintragen und Suchen über die Homepage von Microsoft, als auf die Suchanfrage mithilfe der JAVA API UDDI4J stellen eine Anfrage in Form eines SOAP Dokumentes an den entsprechenden UDDI Server, dieser liefert ebenfalls eine Antwort in SOAP. Weder Microsoft noch das JAVA Programm liefern genauere Informationen zu dieser SOAP Anfrage. Hierzu ist ein weiteres Tool zum Mitschneiden dieser Anfrage nötig. Eine weitere Möglichkeit bietet die Seite www.soapclient.com.

Auf dieser Seite wird ein UDDI – Browser angeboten über denen Suchanfragen gestellt werden können.

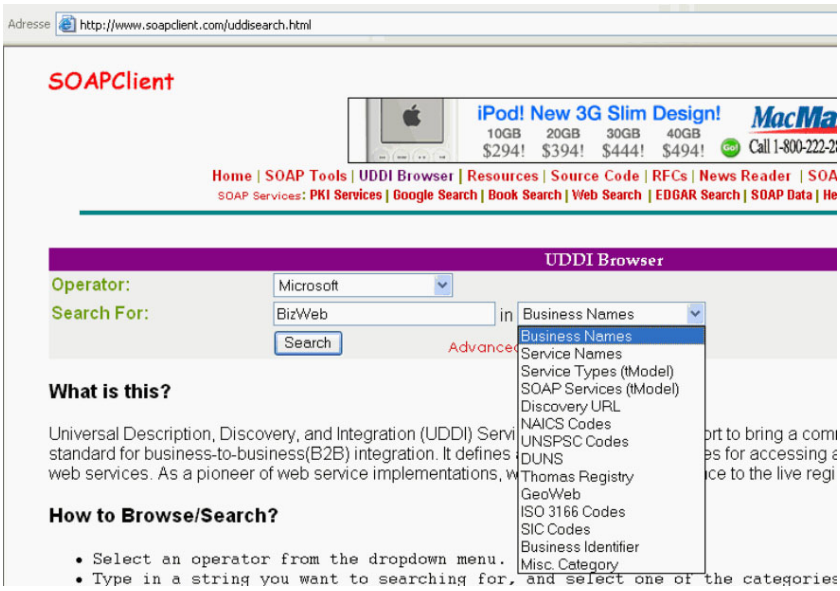


Abbildung 13.9.: UDDI: Anfrage in SOAP – www.soapclient.com (1)

Über SOAP Request und SOAP Response (siehe Abb. 13.10) kann nun die komplette Anfrage bzw. Antwort in SOAP angezeigt werden.

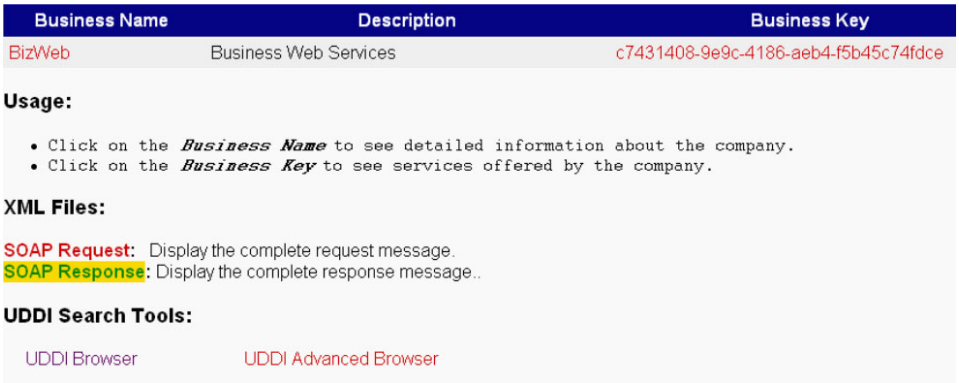


Abbildung 13.10.: UDDI: Anfrage in SOAP – www.soapclient.com (2)

Antwort des UDDI Servers von Microsoft:

```
<soap:Envelope>
  <soap:Body>
    <businessList generic="1.0" operator="Microsoft Corporation" truncated="false">
      <businessInfos>
        <businessInfo businessKey="c7431408-9e9c-4186-aeb4-f5b45c74fdce">
          <name>BizWeb</name>
          <description xml:lang="en">Business Web Services</description>
          <serviceInfos/>
        </businessInfo>
      </businessInfos>
    </businessList>
  </soap:Body>
</soap:Envelope>
```

Quelltext 13.7: UDDI: Anfrage in SOAP – www.soapclient.com (3)

13.9. Alternative WS – Inspection

Es gibt Alternativen zu UDDI. Eine ist die Web Service Inspection Language (WS–Inspection), welche von IBM und Microsoft ausgearbeitet wurde. Diese ist sehr einfach, denn es gibt keine Registrierung bei einem Server. Statt dessen wird eine Datei mit dem Namen Inspection.wsil im root Verzeichnis der Firmendomain gelagert. In dieser Datei sind einfach die Web Services inklusive Zeiger auf das WSDL Dokument gelistet.

Hier ein Beispiel so einer Datei:

```
<?xml version="1.0" ?>
<inspection
   xmlns="http://schemas/xmlsoap.org/ws/2001/10/inspextion/"
   xmlns:uddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/">
  <service>
    <abstract>Beispielservice</abstract>
    <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
       location="http://example.com/beispielservice.wsdl"/>
    </description>
  </service>
</inspection>
```

Quelltext 13.8: UDDI: Alternative WS – Inspection

WS – Inspection ist insbesondere dann nützlich, wenn man den Service Anbieter schon kennt, und nur noch die genauen Services wissen will, die angeboten werden.

13.10. Quellen

- ***Understanding Web Services*** – *XML, WSDL, SOAP, and UDDI*
  Eric Newcomer
  Addison–Wesley 2002

- ***Building Web Services with Java*** – *Making Sense of XML, SOAP, WSDL, and UDDI*
  Graham, Simeonov, Boubez, Davis, Daniels, Nakamura, Neyama
  SAMS Publishing 2002

- ***UDDI, SOAP, and WSDL*** – *The Web Service Specification Reference Book*
  Aaron E. Walsh
  Prentice Hall PTR, 2002

14. Lauffähige Umgebung einrichten

Kurzanleitung zur Einrichtung einer lokalen, lauffähigen Umgebung mit Tomcat und Axis

Die Pfadangaben orientieren sich an der Installation auf den Rechnern im SAP-Labor, können natürlich individuell angepasst/verändert werden.

1. Java SDK 1.4.2_04 nach `C:\Java\j2sdk1.4.2` installieren
2. Tomcat installieren nach `C:\Java\Tomcat`
3. Eclipse SDK M7 entpacken
4. Schreibschutz vom Verzeichnis entfernen
   (ich weiß nicht, ob es zwingend notwendig ist)
5. Eclipse starten (damit workspace angelegt wird)
6. Unter „Window“ → „Preferences“ → „Java“ → „Installed JREs“
   - das installierte j2sdk1.4.2_04 hinzufügen (add)
   - „JRE name“: beliebig (z.B. `j2sdk1.4.2_04`)
   - „JRE home directory“: `C:\Java\j2sdk1.4.2_04`
   - „Javadoc URL“: http://java.sun.com/j2se/1.4.2/docs/api
   - den neu angelegten Eintrag unter „Installed JREs“ aktivieren
7. Eclipse beenden.
8. Folgende PlugIns entpacken: (unter WinZip: „Entpacken nach...“)
   - `com.myspotter.wsdl2java.zip` (`C:\Java`
   - `editors.0.4.1.zip` **Achtung!** Das Verzeichnis „Update“ gibt es nicht; dessen Inhalt nach `C:\Java\eclipse`
   - `emf_2.0.0_20040127_1738SL.zip` (`C:\Java\eclipse`)
   - `GEF-runtime-I20040212.zip` (`C:\Java\eclipse`)
   - `lomboz.3m7.zip` (`C:\Java\eclipse`)

- `net.sf.solareclipse_0.4.1.bin.dist.zip` (`C:\Java\eclipse`)
- `tomcatPluginV221.zip` (`C:\Java\eclipse\plugins`)
- `uml2_200401170850.zip` (`C:\Java\eclipse`)
- `VE-runtime-I20040218.zip` (`C:\Java\eclipse`)
- `xmlbuddy_2.0.5.zip` (`C:\Java\eclipse\plugins`)
- `xsd-runtime-I200403250631.zip` (`C:\Java\eclipse`)

9. erneut Schreibschutz entfernen
10. Eclipse starten
    - Help → About → Manage PlugIns → gibt evtl. Fehlermeldung (macht aber nichts) → Fenster schließen
      Falls die neuen Projekte oder Ansichten nicht direkt unter „New“, „Project“ zu finden sing, einfach mal gucken unter: → „New“, „Project“, „Others“ sind die neuen Funktionen, bei den Ansichten unter „Window“ → „Show View“
11. Tomcat–Einstellungen unter „Window“ → „Preferences“ → „Tomcat“ einstellen:
    - „Tomcat Version“: Version 5.x
    - „Tomcat Home“: `C:\Java\Tomcat`
    - „Tomcat base“: `C:\Java\Tomcat`
    - „Konfigurationsdatei“: `C:\Java\Tomcat\conf\server.xml`
12. Tomcat starten (Button in Symbolleiste)
13. Tomcat testen mit http://127.0.0.1:8080
14. Port von 8080 auf 80 ändern in `D:\Informatik\Tomcat 5.0\conf\server.xml` (ca. Zeile 94) oder im Tomcat–Administrator unter:
    http://localhost/admin/frameset.jsp (Standard–User: `admin`, OHNE !! Passwort)
15. Tomcat neu starten (per Button möglich)
16. Tomcat testen mit http://127.0.0.1 bzw. http://localhost (in Konsole wird bei Neustart angezeigt: „`INFO: Starting Coyote HTTP/1.1 on port 80`“)
17. Dateien entpacken:
    - `Soap-bin-2.3.1.zip` in das `C:\Java\`–Verzeichnis entpacken und das neue Verzeichnis in „soap“ umbenennen (der Einfachheit halber)
    - `Axis-1_1.zip` in das `C:\Java\`–Verzeichnis entpacken und das neue Verzeichnis in „axis“ umbenennen (der Einfachheit halber)
    - `xerces-J-bin.2.6.2.zip` nach `C:\Java\`–entpacken und das neue Verzeichnis in „xerces“ umbenennen (der Einfachheit halber)

- `javamail-1_3_1.zip` in das `C:\Java\`–Verzeichnis entpacken und das neue Verzeichnis in „javamail“ umbenennen (der Einfachheit halber)
- `jaf-1_0_2.zip` in das `C:\Java\`–Verzeichnis entpacken und das neue Verzeichnis in „jaf“ umbenennen (der Einfachheit halber)

18. Die `soap.war`–Datei (in Lotus–Notes vorhanden) im Tomcat–Manager lokal hochladen unter http://localhost/manager/html

19. `C:\Java\axis-1_1\webapps\axis`–Verzeichnis ins Tomcat–webapps–Verzeichnis kopieren

20. Nachfolgende Libraries aus dem `C:\Java\xerces`–Verzeichnis sollten zusätzlich nach `C:\java\tomcat\webapps\axis\WEB-INF\lib` kopiert werden:

    - `xercesImpl.jar`,
    - `xercesSamples.jar`,
    - `xml-apis.jar`,
    - `xmlParserAPIs.jar`

21. Umgebungsvariablen setzen:
    Vorsicht: Werte anpassen! (Verzeichnisse ändern, falls andere Pfade verwendet werden)!

    Unter „Einstellungen“ → „Systemsteuerung“ → „System“ → Reiter „Erweitert“ → „Umgebunsvariablen“ → „Systemvariablen“ eintragen:

```
<Name>          =   <Wert>
JAVA_HOME       =   C:\Java\j2sdk1.4.2
XERCES_HOME     =   C:\Java\xerces
JMAIL_HOME      =   C:\Java\javamail
JAF_HOME        =   C:\Java\jaf
SOAP_HOME       =   C:\Java\soap
SOAP_LIB        =   %SOAP_HOME%\lib
SOAPCLASSPATH   =   %SOAP_LIB%\soap.jar; %SOAP_LIB%\xerces.jar;
                    %JMAIL_HOME%\mail.jar; %JAF_HOME%\activation.jar
AXIS_HOME       =   C:\Java\axis
AXIS_LIB        =   %AXIS_HOME%\lib
AXISCLASSPATH   =   %AXIS_LIB%\axis.jar; %AXIS_LIB%\commons-discovery.jar;
                    %AXIS_LIB%\commons-logging.jar; %AXIS_LIB%\jaxrpc.jar;
                    %AXIS_LIB%\saaj.jar; %AXIS_LIB%\log4j-1.2.8.jar;
                    %AXIS_LIB%\wsdl4j.jar; %XERCES_HOME%\xml-apis.jar;
                    %XERCES_HOME%\xercesImpl.jar;
                    %XERCES_HOME%\xmlParserAPIs.jar
CLASSPATH       =   %AXISCLASSPATH%; %SOAPCLASSPATH%
```

22. Rechner neu starten

Teil III.

Beispielentwicklung

15. Einleitung

Nach den Schulungen wurden an die einzelnen Mitglieder Aufgaben verteilt, anhand derer sie das bislang Gelernte anwenden sollten. Die Aufgaben sind aufgeteilt in einfache Web Services, Web Services mit MySQL–Datenbank oder mit XML–Datenbank (Apache Xindice). Zu diesen Aufgaben gibt es verschiedenen Thematiken, wie z.B. das Erstellen eines Währungsrechner oder einer datenbankgestützten Literaturverwaltung – die Umsetzung und Lösungen aller Aufgaben sind auf den nachfolgenden Seiten dargestellt.

Ziel dieser Aufgaben war es, den Projektteilnehmern die Möglichkeit einer praxisorientierten Anwendung der Web Services–Technologien zu geben. Dies ist neben den Schulungen, dem theoretischen Teil des Projekts, die Grundlage für die kommende Aufgabe.

16. Taschenrechner

16.1. Ziel eines Web Services

Ein Web Service ist eine Art Webanwendung, die über Internet veröffentlicht wird, und die anschließend von Clients (Benutzer oder Anwendungen) lokalisiert und aufgerufen werden kann.

Eine der Haupteigenschaften von Web Services ist die hohe Abstraktion zwischen ihrer Implementierung und ihrer Verwendung. Dieses hohe Abstraktionsniveau ermöglicht das Verwenden von Web Services ohne die Implementationsdetails kennen zu müssen. In anderen Worten, die einzige Information, welche der Anbieter eines Web Services einem Benutzer mitteilen muss, ist das Format, Inhalt sowie Reihenfolge der Inputs und Outputs (in Form von SOAP-Nachrichten) und die Adresse an der das Web Service zugänglich ist. Weitere Details wie Programmiersprache, Hardware, Betriebssystem und sonstige Implementationsdetails werden vom Benutzer nicht benötigt.

Des Weiteren werden durch die Verwendung von Web Services Probleme mit Firewalls von vornherein ausgeschlossen, da auf die Web Services zumeist mittels HTTP über Port 80 zugegriffen wird, welcher fast immer offen ist.

16.2. Beschreibung des Web Services

Von jedem der BizWeb-Projektteilnehmer sollte ein Java Web Service erstellt werden.

In der folgenden Ausarbeitung wird das Web Service-Szenario eines Taschenrechners mit den Grundrechenfunktionen Addition, Subtraktion, Multiplikation, Division, Prozent und Wurzel vorgestellt.

Die Präsentation des Web Services soll als Swing GUI erfolgen und abschließend auf dem BizWeb-Server veröffentlicht werden.

16.3. Aufbau einer Entwicklungsumgebung

Um einen Web Service erstellen zu können, muss zunächst eine lauffähige Umgebung erzeugt werden. Dabei stehen in unserem Projekt folgende Technologien im Mittelpunkt:

- *Apache Tomcat* (Version 5)
- *Apache AXIS*

16.3.1. Apache Tomcat

Tomcat wird in der Axis Dokumentation für den Einsatz empfohlen. Er unterstützt die Verwendung von HTTP über SSL. Das bedeutet, dass er alle notwendigen Bedingungen erfüllt und stellt damit die passende Umgebung für den Einsatz der Integrationsplattform dar.
Als erstes muss der Tomcat Applikationsserver installiert werden. Dazu wurde der Installationspfad `C:\java\tomcat` ausgewählt.
Wenn die Installation ohne Probleme abgelaufen ist, sollte der Server unter der URL http://localhost:8080 auf dem lokalen System, nachdem dieser gestartet wurde, aufrufbar sein.

16.3.2. Apache AXIS

Grundsätzlich ist Axis eine SOAP Engine, also ein Framework zum Aufbau von Komponenten zur Verarbeitung von SOAP-Nachrichten, wie Clients, Server und Gateways. Darüber hinaus umfasst es auch Werkzeuge für die Unterstützung für WSDL, mit deren Hilfe z.B. Java Klassen aus WSDL–Beschreibungen erzeugt werden können.
Apache AXIS wurde in das Verzeichnis `C:\java\axis` installiert. Um AXIS im Tomcat Applikationsserver einzurichten, muss nur das Verzeichnis `C:\java\axis\webapps\axis` in das Tomcat-Anwendungsverzeichnis `tomcat\webapps` kopiert werden. Folgende `jar`-Dateien müssen im Verzeichnis `tomcat\webapps\axis\WEB-INF\lib` zu finden sein:

- `axis.jar`
- `axis-ant.jar`
- `commons-discovery.jar`
- `commons-logging.jar`
- `jaxrpc.jar`
- `log4j-1.2.8.jar`

- `saaj.jar`
- `wsdl4j.jar`

Darüber hinaus müssen folgende Umgebungsvariablen gesetzt werden:

```
AXIS_HOME       =  C:\Java\tomcat\webapps\axis
AXIS_LIB        =  %AXIS_HOME%\WEB-INF\lib
AXIS_CLASSPATH  =  %AXIS_LIB%\axis.jar;
                   %AXIS_LIB%\axis-ant.jar;
                   %AXIS_LIB%\commons-discovery.jar;
                   %AXIS_LIB%\commons-logging.jar;
                   %AXIS_LIB%\jaxrpc.jar;
                   %AXIS_LIB%\log4j-1.2.8.jar;
                   %AXIS_LIB%\saaj.jar;
                   %AXIS_LIB%\wsdl4j.jar
```

Danach muss der Server erneut gestartet werden, damit diesem die neuen Dateien bekannt sind. Die AXIS-Anwendung ist nun unter folgender URL zu erreichen:
http://localhost:8080/axis/index.html

Unter dem Link *Validate the local installation's configuration* kann geprüft werden, ob alle notwendigen Bestandteile installiert sind. Falls nicht wird dort eine Fehlermeldung ausgegeben.

16.4. Implementierung des Web Services

16.4.1. Programmierung

Als erstes muss ein Java-Package erstellt werden, in dem dann die Klasse `Taschenrech` erzeugt wird. Dort sind alle Methoden implementiert. Diese bekommen gewisse Parameter übergeben und liefern das Ergebnis zurück:

```
1  import java.math.*;
2  public class Taschenrech {
3    public double add(double i1, double i2)
4    {return i1 + i2;}
5    public double sub(double i1, double i2)
6    {return i1 - i2;}
7    public double mult(double i1, double i2)
8    {return i1 * i2;}
9    public double div(double i1, double i2)
10   {return i1 / i2;}
11   public double prozent(double i1, double i2)
12   {return (i1 *100) / i2;}
13   public double wurzel(double i1)
14   {return Math.sqrt (i1);}
15 }
```

Quelltext 16.1: Taschenrechner: Taschenrech.java

16.4.2. Deployment des Web Services unter AXIS

AXIS bietet ein Auto-Deployment an. Dazu muss die `Taschenrech.java` Datei in das Verzeichnis `tomcat\webapps\axis` kopiert werden und in `Taschenrech.jws` umbenannt werden. Mit Hilfe der JWS-Dateien (Java Web Service) kann eine simple Form des Deployments erreicht werden. Nach dem Neustart des Tomcat-Servers wird das Deployment selbständig durchgeführt. Die compilierte Klasse wird im Verzeichnis `WEB-INF\jws-Classes` abgelegt. Ob der Service erfolgreich deployed wurde, kann im Browser mit der URL: http://localhost:8080/axis/Taschenrech.jws überprüft werden.

16.4.3. Codegenerierung aus dem WSDL-Dokument

WSDL-Dokument

WSDL (Web Service Description Language) verwendet einen speziellen XML-Dialekt zur Beschreibung von Web Services. Dazu existieren in einer WSDL-Definition zunächst fünf verchiedene Konstrukte (auf die jetzt nicht weiter eingegangen werden soll), deren Inhalte teilweise aufeinander Bezug nehmen: `message`, `types`, `portType`, `binding` und `service`.

Die WSDL-Datei kann man sich automatisch über die URL
http://localhost:8080/axis/services/Taschenrechner?wsdl erstellen lassen:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/Taschenrech.jws"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://localhost:8080/axis/Taschenrech.jws"
   xmlns:intf="http://localhost:8080/axis/Taschenrech.jws"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:message name="prozentResponse">
    <wsdl:part name="prozentReturn" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="multRequest">
    <wsdl:part name="i1" type="xsd:double"/>
    <wsdl:part name="i2" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="subRequest">
    <wsdl:part name="i1" type="xsd:double"/>
    <wsdl:part name="i2" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="wurzelRequest">
    <wsdl:part name="i1" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="subResponse">
    <wsdl:part name="subReturn" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="prozentRequest">
    <wsdl:part name="i1" type="xsd:double"/>
    <wsdl:part name="i2" type="xsd:double"/>
  </wsdl:message>
```

```
32    <wsdl:message name="wurzelResponse">
33      <wsdl:part name="wurzelReturn" type="xsd:double"/>
34    </wsdl:message>
35    <wsdl:message name="addResponse">
36      <wsdl:part name="addReturn" type="xsd:double"/>
37    </wsdl:message>
38    <wsdl:message name="divResponse">
39      <wsdl:part name="divReturn" type="xsd:double"/>
40    </wsdl:message>
41    <wsdl:message name="divRequest">
42      <wsdl:part name="i1" type="xsd:double"/>
43      <wsdl:part name="i2" type="xsd:double"/>
44    </wsdl:message>
45    <wsdl:message name="multResponse">
46      <wsdl:part name="multReturn" type="xsd:double"/>
47    </wsdl:message>
48    <wsdl:message name="addRequest">
49      <wsdl:part name="i1" type="xsd:double"/>
50      <wsdl:part name="i2" type="xsd:double"/>
51    </wsdl:message>
52    <wsdl:portType name="Taschenrech">
53      <wsdl:operation name="add" parameterOrder="i1 i2">
54        <wsdl:input message="impl:addRequest" name="addRequest"/>
55        <wsdl:output message="impl:addResponse" name="addResponse"/>
56      </wsdl:operation>
57      <wsdl:operation name="sub" parameterOrder="i1 i2">
58        <wsdl:input message="impl:subRequest" name="subRequest"/>
59        <wsdl:output message="impl:subResponse" name="subResponse"/>
60      </wsdl:operation>
61      <wsdl:operation name="mult" parameterOrder="i1 i2">
62        <wsdl:input message="impl:multRequest" name="multRequest"/>
63        <wsdl:output message="impl:multResponse" name="multResponse"/>
64      </wsdl:operation>
65      <wsdl:operation name="div" parameterOrder="i1 i2">
66        <wsdl:input message="impl:divRequest" name="divRequest"/>
67        <wsdl:output message="impl:divResponse" name="divResponse"/>
68      </wsdl:operation>
69      <wsdl:operation name="prozent" parameterOrder="i1 i2">
70        <wsdl:input message="impl:prozentRequest" name="prozentRequest"/>
71        <wsdl:output message="impl:prozentResponse" name="prozentResponse"/>
72      </wsdl:operation>
73      <wsdl:operation name="wurzel" parameterOrder="i1">
74        <wsdl:input message="impl:wurzelRequest" name="wurzelRequest"/>
75        <wsdl:output message="impl:wurzelResponse" name="wurzelResponse"/>
76      </wsdl:operation>
77    </wsdl:portType>
78    <wsdl:binding name="TaschenrechSoapBinding" type="impl:Taschenrech">
79      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
80      <wsdl:operation name="add">
81        <wsdlsoap:operation soapAction=""/>
82        <wsdl:input name="addRequest">
83          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
84             namespace="http://DefaultNamespace" use="encoded"/>
85        </wsdl:input>
86        <wsdl:output name="addResponse">
87          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
88             namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
89        </wsdl:output>
90      </wsdl:operation>
91      <wsdl:operation name="sub">
92        <wsdlsoap:operation soapAction=""/>
93        <wsdl:input name="subRequest">
94          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
95             namespace="http://DefaultNamespace" use="encoded"/>
96        </wsdl:input>
97        <wsdl:output name="subResponse">
```

```
98          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
99             namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
100       </wsdl:output>
101     </wsdl:operation>
102     <wsdl:operation name="mult">
103       <wsdlsoap:operation soapAction=""/>
104       <wsdl:input name="multRequest">
105         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
106            namespace="http://DefaultNamespace" use="encoded"/>
107       </wsdl:input>
108       <wsdl:output name="multResponse">
109         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
110            namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
111       </wsdl:output>
112     </wsdl:operation>
113     <wsdl:operation name="div">
114       <wsdlsoap:operation soapAction=""/>
115       <wsdl:input name="divRequest">
116         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
117            namespace="http://DefaultNamespace" use="encoded"/>
118       </wsdl:input>
119       <wsdl:output name="divResponse">
120         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
121            namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
122       </wsdl:output>
123     </wsdl:operation>
124     <wsdl:operation name="prozent">
125       <wsdlsoap:operation soapAction=""/>
126       <wsdl:input name="prozentRequest">
127         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
128            namespace="http://DefaultNamespace" use="encoded"/>
129       </wsdl:input>
130       <wsdl:output name="prozentResponse">
131         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
132            namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
133       </wsdl:output>
134     </wsdl:operation>
135     <wsdl:operation name="wurzel">
136       <wsdlsoap:operation soapAction=""/>
137       <wsdl:input name="wurzelRequest">
138         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
139            namespace="http://DefaultNamespace" use="encoded"/>
140       </wsdl:input>
141       <wsdl:output name="wurzelResponse">
142         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
143            namespace="http://localhost:8080/axis/Taschenrech.jws" use="encoded"/>
144       </wsdl:output>
145     </wsdl:operation>
146   </wsdl:binding>
147   <wsdl:service name="TaschenrechService">
148     <wsdl:port binding="impl:TaschenrechSoapBinding" name="Taschenrech">
149       <wsdlsoap:address location="http://localhost:8080/axis/Taschenrech.jws"/>
150     </wsdl:port>
151   </wsdl:service>
152 </wsdl:definitions><
```

Quelltext 16.2: Taschenrechner: WSDL

Codegenerierung mit WSDL2Java

Mit Hilfe des Tools *WSDL2Java* wird dem Programmierer viel Arbeit erspart, da dieses aus dem WSDL-Dokument Java Proxies und Codegerüste für die Service-Implementierungen automatisch erstellt.

Es entstehen also neben der Datei `Taschenrech.java` noch folgende andere Java-Dateien:

- `Taschenrech.java` (Interface, das den Webservice beschreibt)
- `TaschenrechService.java` (Interface, das einen Stub liefert (die Factory))
- `TaschenrechSoapBindingStub.java` (Klasse, bzw. der Stub, der `Taschenrech` implementiert)
- `Taschenrech ServiceLocator.java` (Klasse, die die Factory implementiert)

Diese Dateien sind jetzt im Workspace der Entwicklungsumgebung zu übernehmen, sodass sie referenziert werden können.

16.4.4. Implementierung des Clients

Beim Erzeugen eines AXIS-Client kann auf die automatische Code-Generatoren zurückgegriffen werden, die uns einen Großteil der Kommunikationslogik erzeugt.

Es muss eine eigene Klasse erstellt werden. Die Programmierung des eigentlichen Clients erfordert jetzt nur noch wenige Code-Zeilen:

```
1 package localhost.axis.Taschenrech_jws;
2
3 public class InvokeTaschenrech {
4   public static void main(String[] args) throws Exception{
5     TaschenrechService myService = new TaschenrechServiceLocator();
6     Taschenrech myPort = myService.getTaschenrech();
7     ...
8   }
9 }
```

Quelltext 16.3: Taschenrechner: Client

16.4.5. Implementierung des Java-Programms (Swing-GUI)

Die Implementierung des AXIS-Client diente nur zu Testzwecken (ob der Web Service fehlerfrei funktioniert).

Im folgenden wird eine Client-Applikation mit einer grafischen Oberfläche Swing (Swing-GUI) implementiert. Dazu werden auch die Stubs, die mit WSDL2Java generiert wurden, verwendet.

Das Swing-GUI ist in der Klasse Oberfläche realisiert. Diese Benutzeroberfläche ermöglicht es, in zwei `jTextFields` Werte einzugeben. Durch die Auswahl der gewünschten Rechenoperation in der `ComboBox` und der Betätigung des `jButtons` wird der Web Service aufgerufen. Dem Web Service werden die eingegebenen Werte und die gewünschte Rechenoperation übergeben (Umwandlung der Werte von `String` zu `Double`). Der Web Service berechnet mit Hilfe der implementierten Methoden das Ergebnis und gibt dieses zurück. Das Ergebnis wird dann in einem `jTextField` ausgegeben (Umwandlung der Werte von `Double` zu `String`).

Der Web Service wird wie bei der Client-API mit Hilfe der Stub-Klassen aufgerufen:

```
1  ...
2  TaschenrechService myService = new TaschenrechServiceLocator();
3  Taschenrech myPort = myService.getTaschenrech();
4
5  wert1 = Double.parseDouble(getEingabe1().getText());
6  wert2 = Double.parseDouble(getEingabe2().getText());
7
8  if(getcombo().getSelectedItem().equals("Addition")) {
9    erg = myPort.add(wert1, wert2);
10   getErgebniss().setText(Double.toString(erg));
11 }
12 ...
```

Quelltext 16.4: Taschenrechner: Swing-GUI

16.5. Beispielaufruf des Web Services

Der Web Service ist fertig implementiert und liegt auf dem BizWeb-Server. Im Folgenden wird ein Beispielaufruf des Web Services durch das Starten der `Oberfläche.java` Datei gezeigt. Es müssen Werte eingegeben und ein Operator ausgewählt werden:

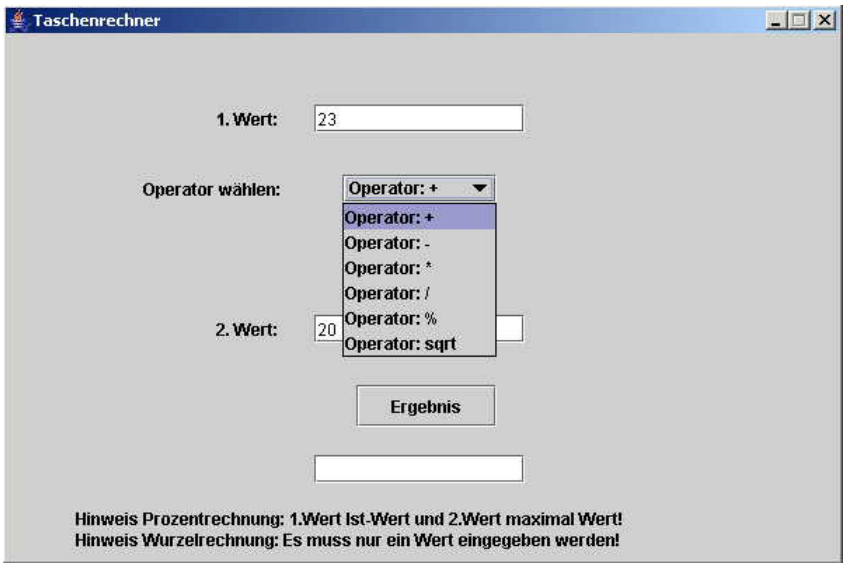


Abbildung 16.1.: Taschenrechner: GUI (1)

Nachdem der Benutzer den Ergebnis-Button geklickt hat, berechnet der Web Service das Ergebnis und liefert es zurück:

Abbildung 16.2.: Taschenrechner: GUI (2)

16.6. Resümee

Die Erstellung eines Web Services (von Anfang bis zum Ende) gab einen vertieften Einblick in alle theoretischen Kenntnisse, die wir uns im vorigen Semester erarbeitet haben. Die Aufgabe gab vertiefte Einblicke in die Architektur und Funktionsweise eines Web Services sowie mit dem Umgang des Werkzeuges Eclipse. Die Umsetzung der Aufgabenstellung hat letztendlich sehr viel Spaß gemacht und die benötigten Grundkenntnisse konnten durch die Präsentationen des letzten Semesters gut vermittelt werden.

Meines Erachtens ist das Thema Web Services ein sehr wichtiger Bestandteil des Studiums. Der Umgang und die Entwicklung von Web Services wird in Zukunft von sämtlichen Unternehmen verlangt. Aus diesem Grund ist das BizWeb-Projekt ein Sprungbrett für jeden Projekt-Teilnehmer. Ich würde mir wünschen, dass es in Zukunft eine Vorlesung an der Hochschule installiert wird, die dieses Thema behandelt.

17. Währungsrechner

17.1. Ziel des Web-Services

Ziel des Web-Services „Währungsrechner“ soll es sein eine Funktionalität zur Verfügung zu stellen, welche die Währungen Euro, US-Dollar, Britischer Pfund und Deutsche Mark, jeweils in die gewünschte andere Währung umrechnet. Also z.B. Euro in US-Dollar und umgekehrt. Das Programm, welches diese Funktionalität beinhaltet soll mit Hilfe der WSDL-Beschreibung von jedem als Web-Service genutzt werden können.

17.2. Pfade für die lauffähige Umgebung

Um die Aufgabe zu lösen wurden folgende vorbereitende Pfade in den Umgebungsvariablen gesetzt.

```
AXIS_HOME      =  D:\Entwurf\Tomcat\webapps\axis\WEB-INF
AXIS_LIB       =  %AXIS_HOME%\lib
AXISCLASSPATH  =  %AXIS_LIB%\axis.jar; %AXIS_LIB%\axis-ant.jar;
                  %AXIS_LIB%\commons-discovery.jar;
                  %AXIS_LIB%\commons-logging.jar;
                  %AXIS_LIB%\jaxrpc.jar;
                  %AXIS_LIB%\log4j-1.2.8.jar;
                  %AXIS_LIB%\saaj.jar;
                  %AXIS_LIB%\wsdl4j.jar;
                  %AXIS_LIB%\xercesImpl.jar;
                  %AXIS_LIB%\xercesSamples.jar;
                  %AXIS_LIB%\xml-apis.jar;
                  %AXIS_LIB%\xmlParserAPIs.jar
JAVA_HOME      =  D:\Entwurf\j2sdk1.4.2\_04
CLASSPATH      =  %AXISCLASSPATH%
```

Zusätzlich dazu habe ich noch folgende Umgebungsvariablen gesetzt.

```
SOAP_HOME       =  D:\Entwurf\Tomcat\webapps\soap\WEB-INF
SOAP_LIB        =  %SOAP_HOME%\lib
SOAPCLASSPATH   =  %SOAP_LIB%\soap.jar;
                   %SOAP_LIB%\xerces.jar;
                   %SOAP_LIB%\mail.jar;
                   %SOAP_LIB%\activation.jar;
                   %SOAP_LIB%\xmlapis.jar;
                   %SOAP_LIB%\xmlParserAPIs.jar;
                   %SOAP_LIB%\xercesImpl.jar;
                   %SOAP_LIB%\xercesSamples.jar
CLASSPATH       =  %AXISCLASSPATH%;
                   %SOAPCLASSPATH%
```

Anmerkungen:

Zusätzlich wurde die `activation.jar` aus dem SDK in `tomcat/webapps/axis/web-inf/lib` kopiert. Die Pakete für SOAP wurden nach `D:\Entwurf\Tomcat\webapps\soap\WEB-INF\lib` kopiert damit der Apache SOAP Admin funktioniert. Der SOAP-Admin ist für die Lösung aber nicht relevant, da Apache Axis verwendet werden sollte.

17.3. Beschreibung des Web-Services

Der Web-Service stellt eine Java-Klasse dar, in welcher einer Methode drei Parameter übergeben werden. Da sind zum einen die Basiswährung und die Umrechnungswährung und zum anderen der Betrag, welcher umgerechnet werden soll. Das Programm errechnet auf Grundlage dieser Werte in der Methode „kurs“ das Ergebnis und schickt es an seinen „Aufrufer“ zurück. Es nutz dabei Wechselkurse, die allerdings als Variablen festgelegt sind. Man könnte dies umgehen, indem bei jedem Aufruf z.B. eine Datenbank mit aktuellen Wechselkursen abgefragt wird. Somit würden diese Variablen immer aktuell gehalten. Der Quellcode des Web-Services ist unter dem Punkt „Programmdokumentation“ zu finden.

Die Datei des Java-Programms zur Währungsumrechnung wurde Currency.java genannt und nach der Fertigstellung in das Verzeichnis `%TOMCAT_HOME%/webapps/axis` kopiert und in Currency.jws umbenannt. Dadurch ist diese Datei in Axis nun als Web-Service „bekannt“ und die kompilierte Klasse wird in `%TOMCAT_HOME%/webapps/axis/WEB-INF/jwsClasses` abgelegt. Die WSDL-Datei des Web-Service kann nun unter http://localhost/axis/Currency.jws?wsdl aufgerufen und als XML-Datei gespeichert werden. Diese wird dann in Currency.wsdl umbenannt und in ein Java-Projekt in Eclipse importiert. Jetzt wird das wsdl2java-Plugin genutzt um aus der WSDL-Datei die Java-Klassen und –Interfaces für die Kommunikationslogik zu generieren.

Folgende Klassen und Interfaces wurden durch das wsdl2java-Plugin erstellt:

- Currency.java
- CurrencyService.java
- CurrencyServiceLocator.java
- CurrencySoapBindingStub.java

17.4. Java-Programm für den Aufruf

Das Java-Programm für den Aufruf des Web-Services nutzt die Klassen, die zuvor durch das wsdl2Java-Plugin generiert wurden. Zuvor sollten allerdings diese generierten Klassen in ein neu angelegtes Package verschoben werden, welches dieselbe Struktur hat wie der durch das wsdl2java-plugin erzeugte Verzeichnisbaum (localhost.axis.Currency_jws in diesem Fall). Danach sollten die jar-Pakete aus dem AXISCLASSPATH in das Projekt integriert werden, damit die generierten Klassen darauf zurückgreifen können.

In diesem Package wurde nun eine neue Klasse CurrencyClient angelegt um den Web Service aufzurufen und mit Übergabewerten zu füttern. Zum Testen erhält diese Klasse vorübergehend eine main-Methode.

```
1
2  package localhost.axis.Currency_jws;
3
4  public class CurrencyClient
5  {
6      double z;
7      String s;
8      String s1;
9
10     public CurrencyClient(String c, String c1,double x)
11     {
12         this.s = c;
13         this.s1 = c1;
14         this.z = x;
15     }
16
17
18
19
20     public double getErg() throws Exception
21     {
22         CurrencyService MyService = new CurrencyServiceLocator();
23         Currency MyPort = MyService.getCurrency();
24         return MyPort.kurs(s,s1,z);
25     }
26
27     public static void main (String[] args) throws Exception
28     {
29       CurrencyClient t = new CurrencyClient("EUR","USD",50);
30       System.out.println(t.getErg());
31     }
32 }
```

Quelltext 17.1: Währungen: Client

Im Konstruktor werden die Klassenvariablen initialisiert, welche dem Web-Service über die generierten Interfaces und Hilfsklassen übergeben werden. Der Rückgabewert der Currency.jws wird dann in der Konsole ausgegeben.

Dieser Test hat gezeigt, dass die Kommunikation mit dem in Axis hinterlegten Service funktioniert. Als nächstes wurde ein GUI (CurrencyGUI.java) programmiert, welches die Klasse CurrencyClient aufruft wenn ein Button gedrückt wird. Das GUI wurde mit Hilfe des Visual Editors für Eclipse erstellt und erhielt zwei Combo Boxen (jComboBox und jComboBox1) in denen die Basis- und Umrechnungswährungen ausgewählt werden können, sowie zwei Textfelder (jTextField und jTextField1).

Im ersten Textfeld soll der Betrag der Basiswährung eingetragen werden, die umgerechnet werden soll. Das Ergebnis der Berechnung wird nach dem Drücken des Buttons „Umrechnen“ im zweiten Textfeld dargestellt. Der Action Listener, welcher auf das „Drücken“ des Buttons wartet, sieht folgendermaßen aus:

```
1
2  jButton.addActionListener(new java.awt.event.ActionListener()
3  {
4  public void actionPerformed(java.awt.event.ActionEvent e)
5    {
6      try
7      {
8        t = Double.parseDouble(jTextField.getText());
9      }
10 catch (NumberFormatException ne)
11     {
12       t=0;
13     }
14
15 if(t!=0)
16     {
17       try
18       {
19         String i = (String)jComboBox.getSelectedItem();
20         String j = (String)jComboBox1.getSelectedItem();
21
22         CurrencyClient kc = new CurrencyClient(i,j,t);
23                           jTextField1.setText(String.valueOf(kc.getErg()));
24       }
25       catch(Exception e1)
26       {
27         e1.printStackTrace();
28       }
29     }
30
31 if(t==0)
32 {
33 JOptionPane.showInternalMessageDialog(jContentPane,
34 "Nur Zahlen und Kommata als Punkte bitte.",
35 "Fehler",JOptionPane.ERROR_MESSAGE);
36     }
37   }
38 });
```

Quelltext 17.2: Währungen: Client – Action Listener

17.5. Beispielsitzung

Das GUI wird über eine neue Klasse gestartet, die das „Look and Feel“ des jeweiligen verwendeten Betriebssystems ausliest.

```
1
2  package localhost.axis.Currency_jws;
3
4  import javax.swing.UIManager;
5
6  public class CurrencyStart
7  {
8      public static void main(String[] args)
9      {
10         try
11         {
12 UIManager.setLookAndFeel
13 (UIManager.getSystemLookAndFeelClassName());
14         }
15         catch(Exception e)
16         {
17      e.printStackTrace();
18         }
19
20 CurrencyGUI start = new CurrencyGUI();
21     }
22 }
```

Quelltext 17.3: Währungen: Client – GUI

Unter XP (mit XP-Design) sieht das GUI dann folgendermaßen aus:

Abbildung 17.1.: Währungen: GUI für den Web-Service 1

Zuerst wird im ersten Pulldown-Menü die Basiswährung ausgewählt.

Abbildung 17.2.: Währungen: GUI für den Web-Service 2

Danach ist im unteren Pulldown-Menü die Umrechnungswährung auszuwählen und im ersten Textfeld der Betrag einzugeben, der umgerechnet werden soll.

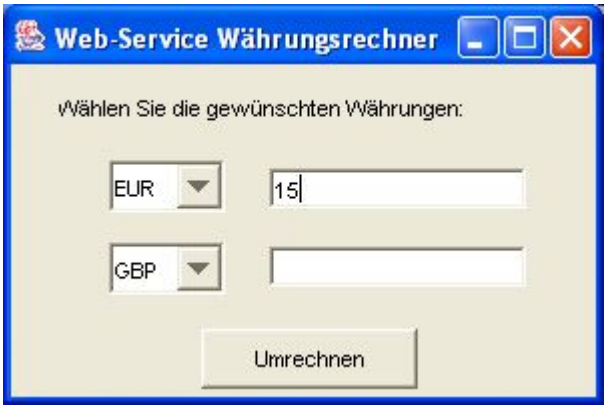


Abbildung 17.3.: Währungen: GUI für den Web-Service 3

Durch das Anklicken des Buttons „Umrechnen“ wird der Web Service aufgerufen und das Ergebnis, welches der Service zurückliefert im unteren Textfeld angezeigt.

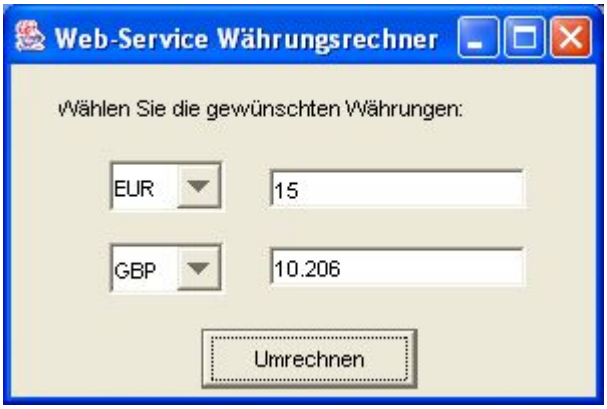


Abbildung 17.4.: Währungen: GUI für den Web-Service 4

Falls keinen Zahlen im ersten Textfeld eingegeben werden, wird eine Fehlermeldung angezeigt. Für Dezimalstellen müssen Punkte statt Kommata verwendet werden.

Abbildung 17.5.: Währungen: Fehlermeldung des GUI

17.6. Beantwortung der Fragen

Der Web-Service wurde zuvor auf den Tomcat-Server des BizWeb-Projektes verschoben. Das „localhost“ der oben angegebenen URLs muss demnach durch `195.37.183.100` bzw. `bizweb.hs-bremerhaven.de` ersetzt werden.

17.6.1. Welche URL ist notwendig um die WSDL-Beschreibung des Web-Services zu erhalten?

Die WSDL-Beschreibung des Web-Services ist unter http://195.37.183.100/axis/Currency.jws?wsdl (ohne Trennzeichen bei Currency) zu erreichen.

17.6.2. Der Informationsfluss zwischen Client und Server

Um den Informationsfluss zu untersuchen wurde das Werkzeug „Ethereal“ eingesetzt. Auf dem Heim-PC wurde nun mit Hilfe der WSDL-Datei (s.o.) sichergestellt, dass der Web-Service auf dem BizWeb-Server erreicht werden kann. Das GUI wurde ausgeführt und ein Betrag umgerechnet. Der dabei entstehende Netzwerkverkehr wurde von Ethereal mitgeloggt. Folgende Kommunikation fand zwischen Client und Server statt:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.10	195.37.183.100	TCP	1186 > http [SYN] Seq=0 Ack=0 Wi
2	0.070165	195.37.183.100	192.168.1.10	TCP	http > 1186 [SYN, ACK] Seq=0 Ack
3	0.070262	192.168.1.10	195.37.183.100	TCP	1186 > http [ACK] Seq=1 Ack=1 Wi
4	0.210903	192.168.1.10	195.37.183.100	HTTP	POST /axis/Currency.jws HTTP/1.0
5	0.324396	195.37.183.100	192.168.1.10	TCP	http > 1186 [ACK] Seq=1 Ack=794
6	0.338445	195.37.183.100	192.168.1.10	HTTP	HTTP/1.1 200 OK
7	0.338545	195.37.183.100	192.168.1.10	TCP	http > 1186 [FIN, ACK] Seq=686 A
8	0.338609	192.168.1.10	195.37.183.100	TCP	1186 > http [ACK] Seq=794 Ack=68
9	0.480037	192.168.1.10	195.37.183.100	TCP	1186 > http [FIN, ACK] Seq=794 A
10	0.550185	195.37.183.100	192.168.1.10	TCP	http > 1186 [ACK] Seq=687 Ack=79

Abbildung 17.6.: Währungen: Kommunikation zwischen Client und Server

In den ersten drei Zeilen finden der Verbindungsaufbau und die Synchronisation der TCP-Verbindung statt. Danach ruft der Client per HTTP-POST den Web-Service Currency.jws auf und versorgt diesen per SOAP mit den nötigen Parametern. Es folgt ein Ausschnitt aus dem Body des SOAP-Envelope, in dem zu sehen ist, welche Parameter mit welchen Werten übergeben werden.

```
<soapenv :Body>.
 <ns1:ku rs soape
nv:encod ingStyle
="http:/ /schemas
.xmlsoap .org/soa
p/encodi ng/" xml
ns:ns1=" http://D
efaultNa mespace"
>.   <w1  xsi:typ
e="xsd:s tring">E
UR</w1>.    <w2 x
si:type= "xsd:str
ing">USD </w2>.
 <betrag  xsi:typ
e="xsd:d ouble">5
0.0</bet rag>.  <
/ns1:kur s>. </so
apenv:Bo dy>.</so
apenv:En velope>
```

Abbildung 17.7.: Währungen: SOAP-Envelope (Auszug) 1

Der Server bestätigt in Zeile 5 über TCP mit „ACK“ den Erhalt des Pakets. Nach Ausführung des Web-Services wird im Body eines HTTP-Response-Pakets ein SOAP-Envelope, welcher unter anderem den Rückgabeparameter enthält, an den Client gesendet (Zeile 6). Hier ein Ausschnitt des Envelopes:

```
soapenv: Body>.
<ns1:kur sResons
e soapen v:encodi
ngStyle= "http://
schemas. xmlsoap.
org/soap /encodin
g/" xmln s:ns1="h
ttp://De faultNam
espace"> .   <ns1
:kursRet urn xsi:
type="xs d:double
">61.45< /ns1:kur
sReturn> .  </ns1
:kursRes ponse>.
</soapen v:Body>.
</soapen v:Envelo
pe>
```

Abbildung 17.8.: Währungen: SOAP-Envelope (Auszug) 2

Der Client erhält so das Ergebnis über das HTTP-Protokoll und kann es weiterverarbeiten. In den Zeilen 7-10 wird die TCP-Verbindung wieder abgebaut.

17.6.3. Veröffentlichung in einem UDDI-Verzeichnis

Der Web Service wurde unter http://uddi.ibm.com in der IBM UDDI Test Business Registry (https://uddi.ibm.com/testregistry/registry.html) als Technical Model veröffentlicht.

UDDI Business Test Registry
Universal Description, Discovery, and Integration

Edit a Technical Model

If you don't wish to edit the Technical Model press the **Cancel** button to return to the main menu. When you are satisfied with the information you have entered press the **Change** button.

Technical Model Information	
Key	
UUID:9FA920F0-12CC-11D9-A2B3-000629DC0A53	
Name	**Action**
Währungsrechner für die Währungen EUR, DEM, USD und GBP	Edit

Technical Model Description(s)			
Description	**Language**	**Actions**	
Web-Service erhält 2 Strings (z.B. USD, EUR) und einen Betrag der von der 1. in die 2. Währung umgerechnet werden soll. (Hier z.B. USD in EUR)	de	Edit	Delete

Add a new Description

Overview URL		
URL	**Description**	**Actions**
http://195.37.183.100/axis/Currency.jws?wsdl	WSDL-Datei für den Web-Service	Edit Delete

Technical Model Locator(s)

Add a new Locator

Abbildung 17.9.: Währungen: Eintrag des Web-Services im IBM UDDI-Verzeichniss

17.6.4. Kannst Du Deinen Web-Service mit Parametern und darauf folgendem Ergebnis über eine URL im Web-Browser aufrufen?

Um diese Aufgabenstellung zu lösen wurde ein Servlet entwickelt, welches über ein HTML-Form die benötigten Parameter per HTTP-GET einliest. So wird sichergestellt, dass die Parameter auch über die URL bestimmt werden können. Dazu wurde ein Tomcat-Projekt erstellt, in welchem ebenfalls die benötigten jar-Dateien aus dem AXIS-CLASSPATH (s.o.) importiert wurden. Diese müssen dann auch ins Verzeichnis WEB-INF/lib kopiert werden, damit sie dem Servlet bei einem Aufruf zur Verfügung stehen. Das Package mit der Kommunikationslogik, dem GUI usw. wurde in dieses Projekt in den Ordner WEB-INF/src kopiert. Dieses Package wurde jetzt um die Servlet-Klasse CallWS_Currency ergänzt. Das Servlet übergibt die, über die HTML-Form oder die URL definierten Parameter der Klasse CurrencyClient und erhält das Ergebnis zurück, welches dann im Browser dargestellt wird.

Es folgt ein beispielhafter Aufruf des Web-Services über ein Servlet.

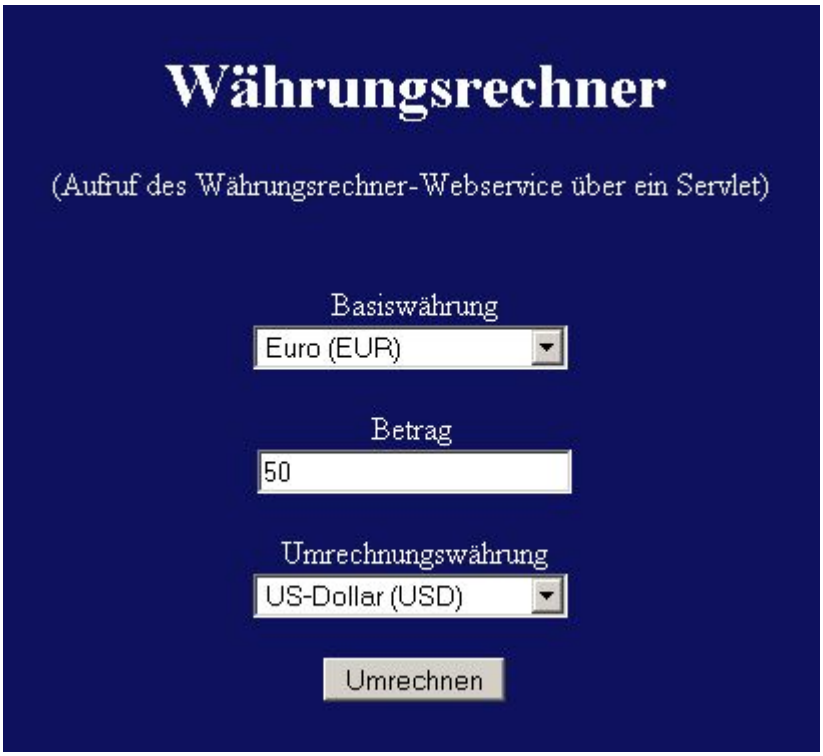


Abbildung 17.10.: Währungen: Währungsrechner-Servlet 1

Die Basis- und Umrechnungswährung, sowie der Umrechnungsbetrag werden in die HTML-Form eingeben bzw. ausgewählt. Durch das Drücken des „Umrechnen“-Buttons erscheint das folgende Bild:

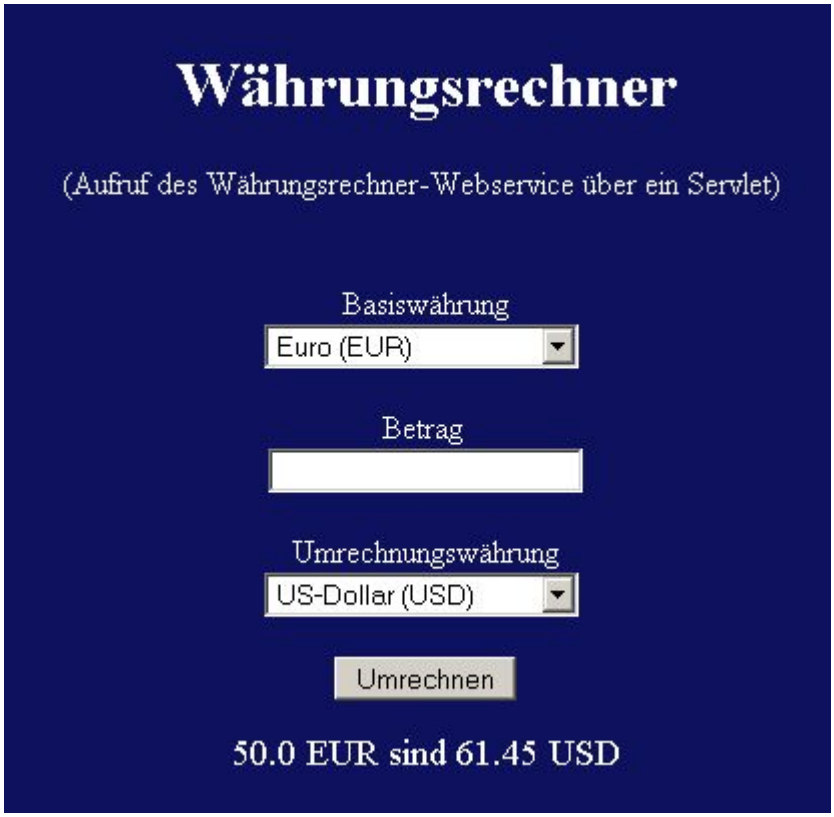


Abbildung 17.11.: Währungen: Währungsrechner-Servlet 2

Dieses Ergebnis würde man auch über die Eingabe der folgenden URL erhalten:
http://195.37.183.100/currencies/servlet/CallWS_Currency?w1=EUR&betrag=50&w2=USD

Das Servlet wird ohne Parameter über die URL:
http://195.37.183.100/currencies/servlet/CallWS_Currency aufgerufen.

17.7. Programmdokumentation / Quellcode des Web-Services

Die Methode Kurs erhält zwei Strings mit je einer Währungsbezeichnung und einen Währungsbetrag vom Typ Double. Je nachdem welche Währungs-Strings übergeben werden, wird eine andere Berechnung durchgeführt und das Ergebnis zurückgeliefert.

```
1
2  public class Currency {
3
4    String seuro = "EUR";
5
6    String sdollar = "USD";
7    double kdollarEUR = 1.2290; // 1.2290 $ für 1€
8
9    String spound = "GBP";
10   double kpoundEUR = 0.6804;     // 0.6804 Pfund für 1€
11   double kpoundUSD = 0.5573; // 0.5573 Pfund für 1$
12
13   String sdm = "DEM";
14   double kdm = 1.95583;    // 1.95583 DM für 1€
15
16   public double kurs (String w1, String w2, double betrag)
17   {
18       double d = -1.0;
19
20       if (w1.equals(seuro) && w2.equals(sdm))
21       {
22           return(betrag*kdm);
23       }
24       if(w1.equals(seuro)&&w2.equals(sdollar))
25       {
26           return(betrag*kdollarEUR);
27       }
28       if(w1.equals(seuro)&&w2.equals(spound))
29       {
30           return(betrag*kpoundEUR);
31       }
32       if(w1.equals(sdollar)&&w2.equals(seuro))
33       {
34           return(betrag/kdollarEUR);
35       }
36       if(w1.equals(sdollar)&&w2.equals(sdm))
37       {
38           return((betrag/kdollarEUR)*kdm);
39       }
40       if(w1.equals(sdollar)&&w2.equals(spound))
41       {
42           return  (betrag*kpoundUSD);
43       }
44       if(w1.equals(spound)&&w2.equals(seuro))
45       {
46           return(betrag/kpoundEUR);
47       }
48       if(w1.equals(spound)&&w2.equals(sdollar))
49       {
50           return(betrag/kpoundUSD);
51       }
52       if(w1.equals(spound)&&w2.equals(sdm))
53       {
54           return((betrag/kpoundEUR)*kdm);
55       }
```

```
56        if(w1.equals(sdm)&&w2.equals(seuro))
57        {
58            return(betrag/kdm);
59        }
60        if(w1.equals(sdm)&&w2.equals(sdollar))
61        {
62            return((betrag/kdm)*kdollarEUR);
63        }
64        if(w1.equals(sdm)&&w2.equals(spound))
65        {
66            return ((betrag/kdm)*kpoundEUR);
67        }
68        if(w1.equals(w2))
69        {
70            return (betrag);
71        }
72        else
73        return d;
74    }
75 }
```

Quelltext 17.4: Währungen: Web Service

17.8. Resümee

Die Aufgabe war sehr gut geeignet um die Funktionsweise von Web-Services und die dazu notwendigen Technologien praktisch kennen zu lernen und besser zu verstehen. Die meist theoretischen Schulungen waren sehr wichtig und eine gute Grundlage. Allerdings vertieft sich das Wissen am Besten, wenn es praktisch angewendet wird.

Die Umsetzung dieser Aufgabe hat viel Spaß gemacht. Man fing klein an und musste sich erst einmal wieder kurz eine Übersicht über die in den Schulungen behandelten Themen verschaffen. Dann lief eigentlich alles fast wie von selbst. Auch die Zusammenarbeit innerhalb des Teams hat gut funktioniert.

18. Flugbuchungsbestätigung

18.1. Ziel Web – Service

Von jedem Projektteilnehmer sollte ein eigener Web Service, zu einem vorgegebenen Thema, entwickelt werden. Das eigentliche Ziel dabei war es, dass sich jeder einzelne mit den Funktionalitäten, den genutzten Techniken und Tools auseinandersetzt, um diese zu verstehen und anwenden zu können. Für den entwickelten Web Service mußte dann noch eine (Client-) Anwendung geschrieben werden und er sollte in einem UDDI-Verzeichnis veröffentlicht werden.

18.2. Einrichtung der Entwicklungsumgebung

18.2.1. Tomcat Installation

Zunächst wird der Tomcat-Applikationsserver in ein beliebiges Verzeichnis installiert, hier: „D:\Programme\Javasoft\Tomcat“. Dieser muss dann noch in der Umgebungsvariable „CATALINA_HOME“ bekannt gemacht werden (siehe Punkt 18.2.3). Nach einem Neustart des Servers ist dieser jetzt unter der Adresse http://localhost:8080 erreichbar.

18.2.2. Axis Installation

Zur Installation von Axis werden die Apache-Axis-Binaries einfach in ein beliebiges Verzeichnis entpackt, in diesem Fall: „D:\Programme\Javasoft\Axis“. Um Axis dann in den Tomcat-Webserver einzubinden muss lediglich der „\webapps\axis“-Ordner aus dem Axis-Verzeichnis in den Webapps-Ordner des Tomcat-Verzeichnisses kopiert werden. In das Verzeichnis „\Tomcat\webapps\axis\WEB-INF\lib“ müssen dann, falls nicht vorhanden, folgenden jar-Dateien eingfügt werden:

- axis.jar
- axis-ant.jar
- commons-discovery.jar

- commons-logging.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j.jar

Um die Umgebung lauffähig zu machen müssen jetzt noch die Klassenpfade gesetzt werden (siehe Punkt 18.2.3).

18.2.3. Setzen der Klassenpfade

Folgenden Klassenpfade mussten in diesem Fall insgesamt gesetzt werden:

```
AXIS_HOME       =   D:\Programme\JavaSoft\Tomcat\webapps\AXIS
AXIS_LIB        =   %AXIS_HOME\lib%
AXISCLASSPATH   =   %AXIS_LIB%\axis.jar;
                    %AXIS_LIB%\commons-discovery.jar;
                    %AXIS_LIB%\commons-logging.jar;
                    %AXIS_LIB%\jaxrpc.jar;
                    %AXIS_LIB%\saaj.jar;
                    %AXIS_LIB%\log4j-1.2.8.jar;
                    %AXIS_LIB%\xml-apis.jar;
                    %AXIS_LIB%\xercesImpl.jar;
                    %AXIS_LIB%\xmlParserAPIs.jar;
                    %AXIS_LIB%\wsdl4j.jar
CATALINA_HOME   =   D:\Programme\Javasoft\Tomcat
CLASSPATH       =   %AXISCLASSPATH%;%SOAPCLASSPATH%
PATH            =   D:\Programme\JavaSoft\j2sdk1.4.2_05\bin;
                    D:\Programme\JavaSoft\j2sdk1.4.2_05\jre\bin
SOAP_HOME       =   D:\Programme\JavaSoft\soap
SOAP_LIB        =   %SOAP_HOME%\lib
SOAPCLASSPATH   =   %SOAP_LIB%\soap.jar;
                    %SOAP_LIB%\mail.jar;
                    %SOAP_LIB%\activation.jar;
                    %SOAP_LIB%\xerces.jar
```

Jetzt muss der Tomcat-Applikationsserver erneut durchgestartet werden. Die Axis-Webapplikation kann jetzt unter http://localhost:8080/axis/ aufgerufen werden.

18.3. Der Web Service

18.3.1. Beschreibung

Der Webservice „Buchungsbest“ soll eine Flugbuchung simulieren. Dazu bekommt er bekommt eine Reihe von Strings übergeben, die aus den Buchungsdaten des jeweiligen Kunden besteht. Da dieser Webservice laut Aufgabe mit keiner Datenbank arbeitet, wird der Rückgabewert anhand einer Random-funktion ermittelt. Mit jeweils 50-prozentiger Wahrscheinlichkeit werden entweder eine Meldung, dass die Buchung nicht möglich war, oder die Buchungsdaten mit einem Bestätigungscode, zurückgegeben.

Die Buchungsdaten, die als eine Kette von Strings an den Web Service übergeben werden setzten sich wie folgt zusammen:

- String nachname (Nachname der buchenden Person)
- String vorname (Vorname der bucheden Person)
- String fluggesellschaft (Fluggesellschaft, mit der geflogen werden soll)
- String sitze (Anzahl der zu reservierenden Sitze)
- String abflughafen (Abreise-Flughafen)
- String zielflughafen (Flughafen des Zielortes)
- String hinflugdatum (Datum der Abreise)
- String hinuhrzeit (Uhrzeit des Abreisefluges)
- String rueckflugdatum (Datum der Rückreise)
- String rueckuhrzeit (Uhrzeit der Rückfluges)

Der Vollständige Quelltext dazu befindet sich im Anhang.

18.3.2. Deployment

Wenn die o.g. Java-Datei („Buchungsbest.java“), also der Web-Service an sich, angelegt worden ist, erfolgt das Deployment. Es bietet sich an das Axis-Auto-deployment zu nutzen. Dazu muss die Java-Datei kopiert und in den Axis-Ordner von Tomcat gelegt werden. Daraufhin muss die Endung der Datei von „.java“ in „.jws“ geändert werden. Wenn das gemacht worden ist, kann der Tomcat Webserver gestartet werden. Der Webservice ist jetzt über einen Browser unter der Adresse http://195.37.183.100/axis/Buchungsbest.jws erreichbar.

18.3.3. Java Programm (Client)

Zum Erstellen des Clients benötigen wir die WSDL-Datei des Webservices. Diese kann in Axis unter der Adresse http://195.37.183.100/axis/Buchungsbest.jws?wsdl aufgerufen werden. Der vollständige Quelltext der WSDL-Datei befindet sich im Anhang.

Bevor die WSDL-Datei jedoch eingebunden werden kann, müssen erst einmal verschiedene Axis-Programmbibliotheken in das Projekt importiert werden.

Abbildung 18.1.: Flugbuchungsbestätigung: Axis Programmbibliotheken

Wenn das geschehen ist, kann die WSDL-Datei in den Ziel-SRC-Ordner des Projektes kopiert werden. Aus dieser kann dann mit dem Plugin „WSDL2Java“ ein Package mit mehreren Klassen generiert werden, welche den Zugriff auf den Webservice regeln.

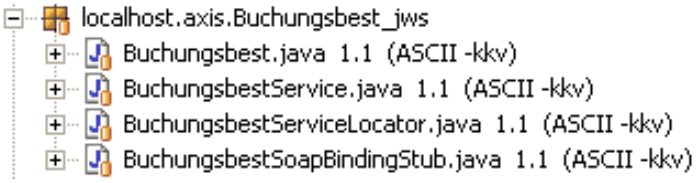


Abbildung 18.2.: Flugbuchungsbestätigung: Aus WSDL generiertes Package

Nachdem das Eclipse-Projekt so weit vorbereitet ist, kann ein Client geschrieben werden, der auf den Webservice zugreift. Im Rahmen dieser Aufgabe wurde zunächst ein Client ohne Benutzeroberfläche geschrieben. Da die Funktionsweise die gleiche ist, wie beim folgend erstellten Java-Swing-GUI, wird auf diesen nicht genauer eingegangen. Der Vollständige Quelltext der Anwendung befindet sich im Anhang.

Als nächstes wurde dann ein Java-Swing-GUI, für eine erleichterte Dateneingabe, angelegt. Der vollständige Quelltext von diesem befindet sich im Anhang. Das GUI sieht wie folgt aus:

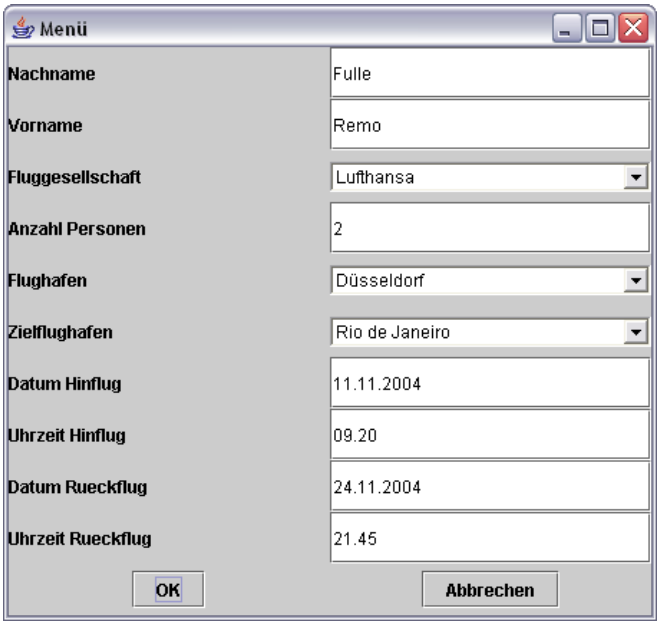


Abbildung 18.3.: Flugbuchungsbestätigung: Client GUI

Nachdem die notwendigen Daten in das GUI eingegeben wurden und diese mit „OK“ bestätigt worden sind, erhält der Benutzer eine Ausgabe auf der Konsole. Zum genauen Ablauf siehe Punkt 18.4.

Das gesamte Projekt in Eclipse hat jetzt die folgende Struktur:

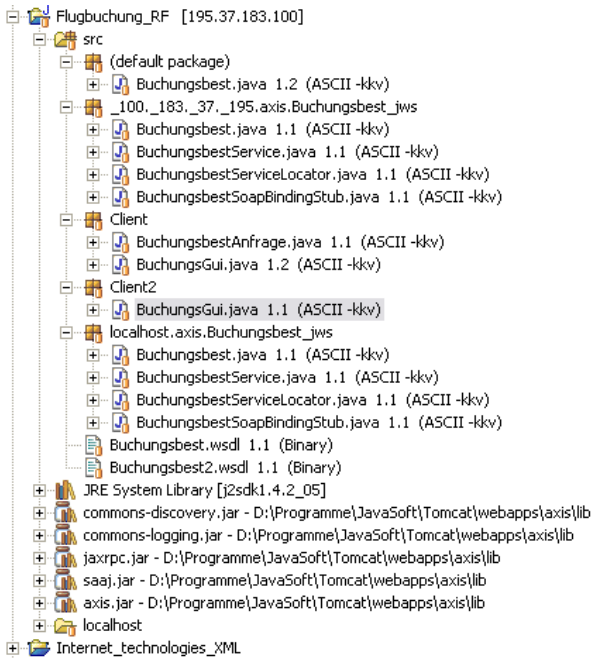


Abbildung 18.4.: Flugbuchungsbestätigung: Package Explorer in Eclipse

18.4. Beispielsitzung

Um das Client-GUI aufzurufen muss die Klasse „BuchungsGui.java“ ausgeführt werden. Im daraufhin erscheinenden Fenster müssen die notwendigen Daten in Textfelder eingegeben, bzw. in Choice-Menüs ausgewählt, werden.

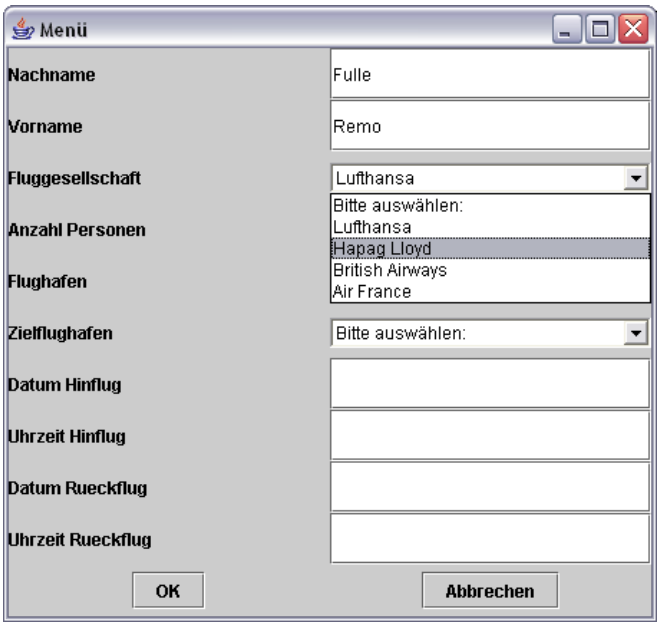


Abbildung 18.5.: Flugbuchungsbestätigung: Auswahl im GUI

Wenn die Eingaben erfolgt sind, muss die Buchungsanfrage mit „OK“ bestätigt werden.

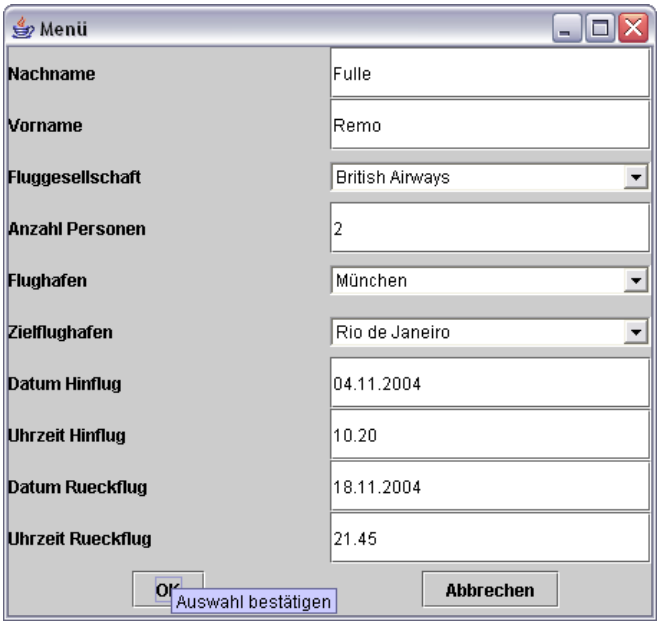


Abbildung 18.6.: Flugbuchungsbestätigung: Bestätigung der Eingaben

Daraufhin erhält der Benutzer eine Ausgabe. Entweder wird die Buchung bestätigt, oder aber, wie in diesem Fall, eine Meldung, dass die Buchung nicht möglich war.

```
Fluggesellschaft:       Lufthansa
Sitze:                  2
Flughafen:              Düsseldorf
Zielflughafen:          Rio de Janeiro
Hinflugdatum:           11.11.2004
Uhrzeit:                09.20
Rueckflugdatum:         24.11.2004
Uhrzeit:                21.45
Bestätigungscode:       Der Flug ist leider ausgebucht.
```

Abbildung 18.7.: Flugbuchungsbestätigung: Ausgabe der Buchung

18.5. Informationsfluss zwischen Client und Server

Um den Informationsfluss zwischen Client und Server festzuhalten, wurde der Web Service auf dem BizWeb-Server per Java-Client über Internet aufgerufen. Der Datenaustausch wurde dann mit dem Tool „Ethereal“ protokolliert.

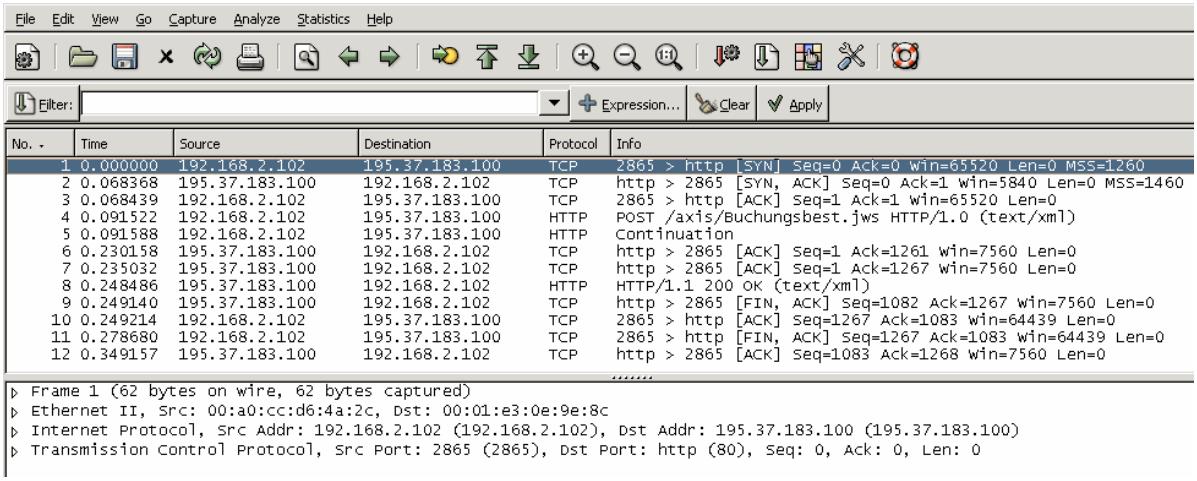


Abbildung 18.8.: Flugbuchungsbestätigung: Protokoll mit Ethereal

Als erstes wird eine TCP-Verbindung hergestellt und synchronisiert (Zeilen 1-3). Danach wird der Web Service per Http-Anfrage aufgerufen und es werden mit Hilfe von SOAP die Parameter übergeben (Zeilen 4-5). Der Server bestätigt anschließend über TCP die Anfrage (Zeilen 6-7) und sendet nach Ausführung des Web Services per Http-Response eine SOAP-Antwort an den Client zurück. Nach der Übertragung der Daten wird die Verbindung wieder abgebaut (Zeilen 9-12).

18.6. Veröffentlichung des Web Services im UDDI-Verzeichnis

Da der erstellte Web Service keinen praktischen Nutzen hat, wurde er im Test-UDDI-Verzeichnis von Microsoft (http://test.uddi.microsoft.com) veröffentlicht. Dort ist er unter dem Suchbegriff „Flugbuchungsbestaetigung“ zu finden.

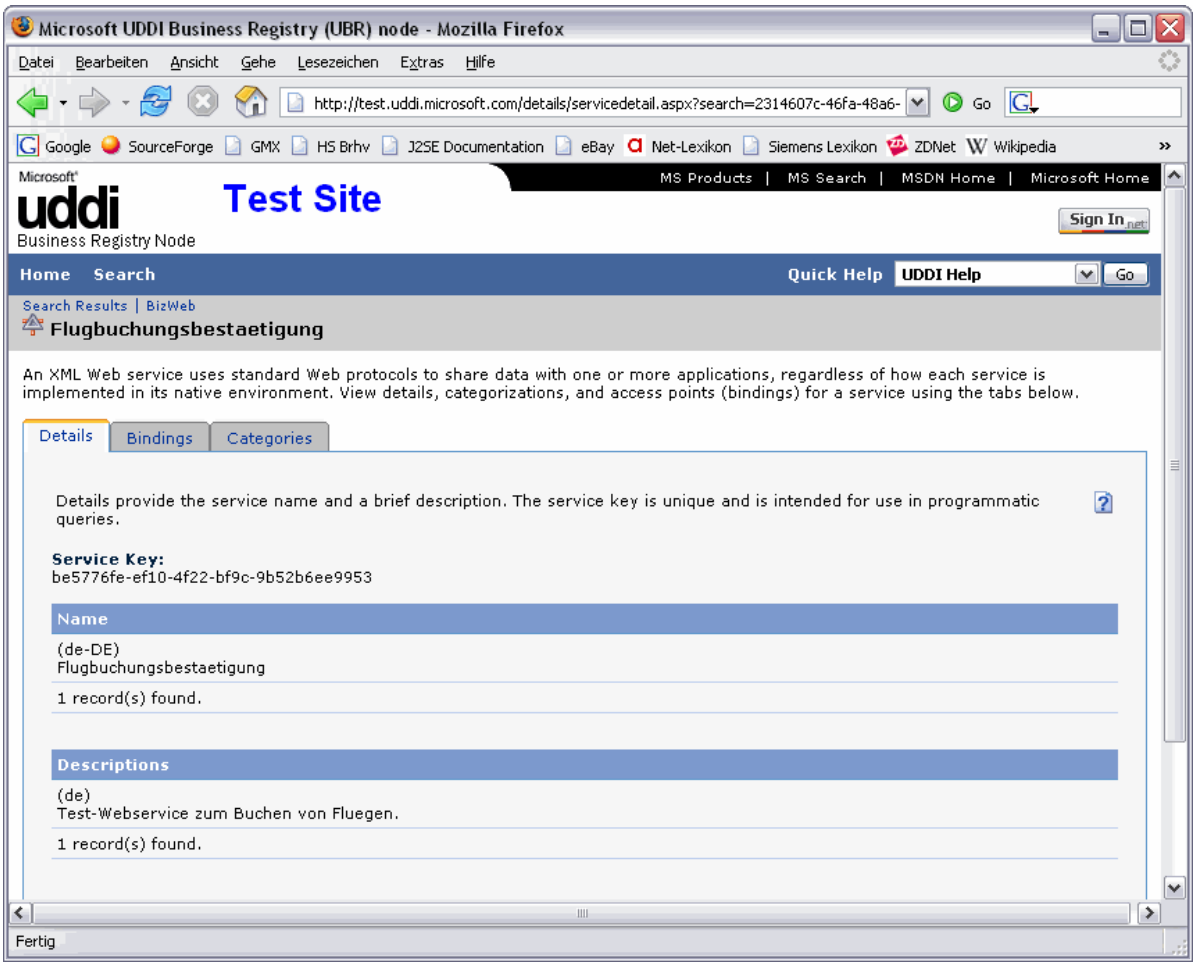


Abbildung 18.9.: Flugbuchungsbestätigung: UDDI-Eintrag

18.7. Aufruf des Web Services über den Browser

Der Web Service kann über der Adresse per Http-Post aufgerufen werden. Dafür muss im Browser das folgende Kommando eingegeben werden:
http://195.37.183.100/axis/Buchungsbest.jws?method=ausgabe&nachname=Fulle&vorname=Remo&fluggesellschaft=Lufthansa&sitze=2&abflughafen=Berlin&zielflughafen=Kairo&hinflugdatum=11.11.2004&hinuhrzeit=09:15&rueckflugdatum=25.11.2004&rueckuhrzeit=21:45
Der erste Teil, bis zum Fragezeichen ist die Zieladresse, dann erfolgt der Aufruf der Methode „ausgabe“, der die nachfolgenden Parameter übergeben werden. Nach dem Aufruf wird folgende Soap-Response zurückgegeben:

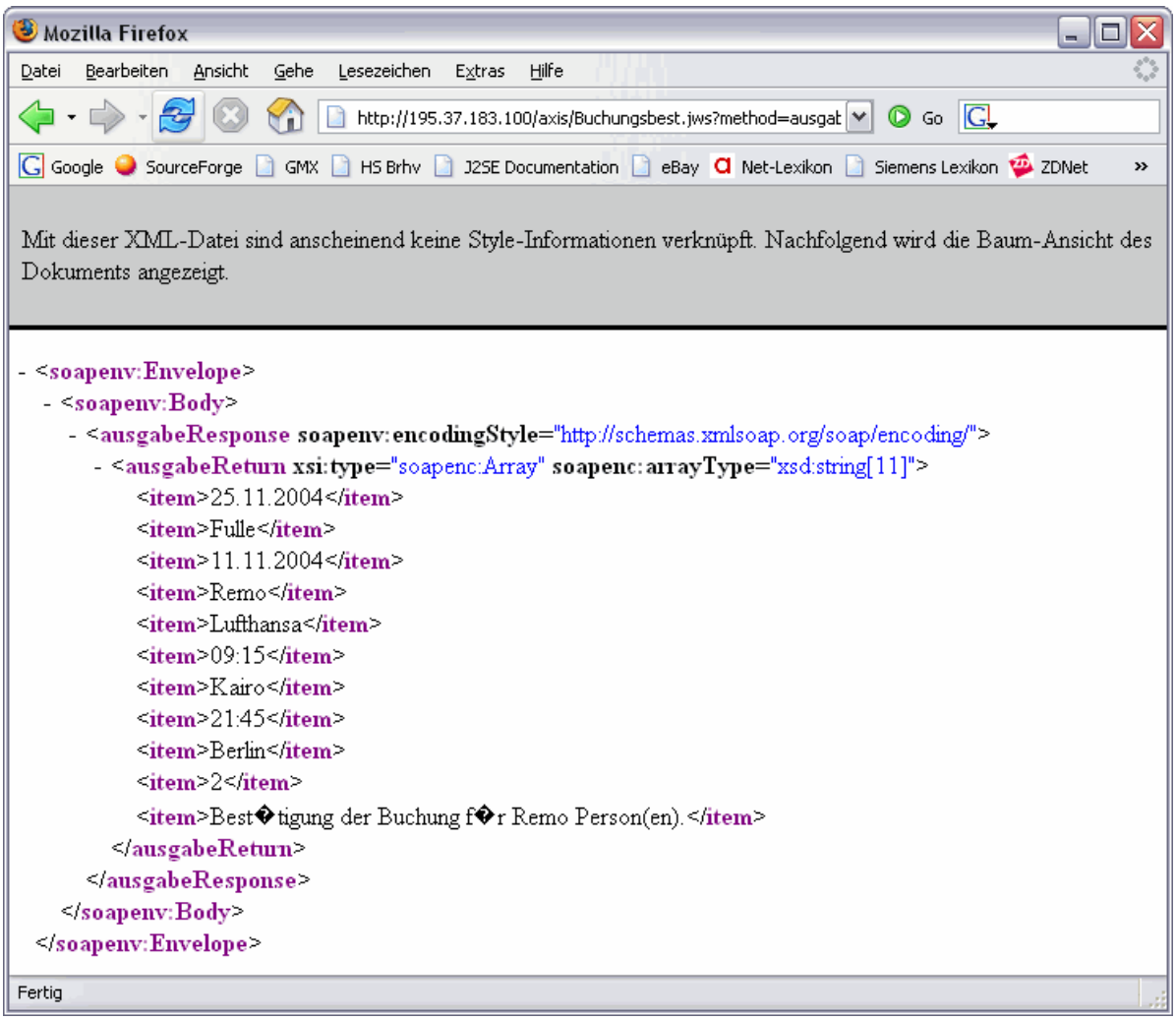


Abbildung 18.10.: Flugbuchungsbestätigung: Browseraufruf Web Service

Der Aufruf an sich funktioniert so weit, allerdings werden die übergebenen Parameter in der Soap-Response aus unerfindlichen Gründen unsortiert zurückgegeben.

18.8. Anhang

```
/*
 * Created on 30.09.2004
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
/**
 * @author Remo
 *
 * To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
import java.util.Random;

public class Buchungsbest{

  public String[] ausgabe(String nachname,
  String vorname,
  String fluggesellschaft,
  String sitze,
  String abflughafen,
  String zielflughafen,
  String hinflugdatum,
  String hinuhrzeit,
  String rueckflugdatum,
  String rueckuhrzeit)
  {
    String bestcode;
    Random r1 = new Random();
    double d1 = r1.nextDouble()*2;
    if (d1 < 1){
      bestcode = "Bestätigung der Buchung für " + sitze + " Person(en).";
    } else
      bestcode = "Der Flug ist leider ausgebucht.";

    String[] flugdaten = {nachname,
    vorname,
    fluggesellschaft,
    sitze,
    abflughafen,
    zielflughafen,
    hinflugdatum,
    hinuhrzeit,
    rueckflugdatum,
    rueckuhrzeit,
    bestcode};
    return flugdaten;
  }
}
```

Quelltext 18.1: Flugbuchungsbestätigung: Web Service Buchungsbest

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://195.37.183.100/axis/Buchungsbest.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://195.37.183.100/axis/Buchungsbest.jws"
xmlns:intf="http://195.37.183.100/axis/Buchungsbest.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types><schema targetNamespace="http://195.37.183.100/axis/Buchungsbest.jws"
```

```
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<complexType name="ArrayOf_xsd_string">
<complexContent><restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
</restriction></complexContent>
</complexType></schema></wsdl:types>
  <wsdl:message name="ausgabeRequest">
    <wsdl:part name="nachname" type="xsd:string"/>
    <wsdl:part name="vorname" type="xsd:string"/>
    <wsdl:part name="fluggesellschaft" type="xsd:string"/>
    <wsdl:part name="sitze" type="xsd:string"/>
    <wsdl:part name="abflughafen" type="xsd:string"/>
    <wsdl:part name="zielflughafen" type="xsd:string"/>
    <wsdl:part name="hinflugdatum" type="xsd:string"/>
    <wsdl:part name="hinuhrzeit" type="xsd:string"/>
    <wsdl:part name="rueckflugdatum" type="xsd:string"/>
    <wsdl:part name="rueckuhrzeit" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="ausgabeResponse">
    <wsdl:part name="ausgabeReturn" type="impl:ArrayOf_xsd_string"/>
  </wsdl:message>
  <wsdl:portType name="Buchungsbest">
    <wsdl:operation name="ausgabe" parameterOrder="nachname
    vorname
    fluggesellschaft
    sitze
    abflughafen
    zielflughafen
    hinflugdatum
    hinuhrzeit
    rueckflugdatum
    rueckuhrzeit">
      <wsdl:input message="impl:ausgabeRequest" name="ausgabeRequest"/>
      <wsdl:output message="impl:ausgabeResponse" name="ausgabeResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BuchungsbestSoapBinding" type="impl:Buchungsbest">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="ausgabe">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="ausgabeRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="ausgabeResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://195.37.183.100/axis/Buchungsbest.jws" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BuchungsbestService">
    <wsdl:port binding="impl:BuchungsbestSoapBinding" name="Buchungsbest">
      <wsdlsoap:address location="http://195.37.183.100/axis/Buchungsbest.jws"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Quelltext 18.2: Flugbuchungsbestätigung: WSDL-Datei

```
package Client2;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import _100._183._37._195.axis.Buchungsbest_jws.*;
```

```
7
8  /*
9   * Created on 30.09.2004
10  *
11  * To change the template for this generated file go to
12  * Window - Preferences - Java - Code Generation - Code and Comments
13  */
14 /**
15  * @author Remo
16  *
17  * To change the template for this generated type comment go to
18  * Window - Preferences - Java - Code Generation - Code and Comments
19  */
20 public class BuchungsGui extends JFrame implements ActionListener{
21   private javax.swing.JPanel jContentPane = null;
22
23   String[] rueckgabe;
24   String nachname;
25   String vorname;
26   String fluggesellschaft;
27   String sitze;
28   String abflughafen;
29   String zielflughafen;
30   String hinflugdatum;
31   String hinuhrzeit;
32   String rueckflugdatum;
33   String rueckuhrzeit;
34
35   JLabel l1 = new JLabel("Nachname");
36   JTextField t1 = new JTextField(20);
37
38   JLabel l2 = new JLabel("Vorname");
39   JTextField t2 = new JTextField(20);
40
41   JLabel l3 = new JLabel("Fluggesellschaft");
42   final Choice c3 = new Choice();
43
44   JLabel l4 = new JLabel("Anzahl Personen");
45   JTextField t4 = new JTextField(2);
46
47   JLabel l5 = new JLabel("Flughafen");
48   final Choice c5 = new Choice();
49
50   JLabel l6 = new JLabel("Zielflughafen");
51   final Choice c6 = new Choice();
52
53   JLabel l7 = new JLabel("Datum Hinflug");
54   JTextField t7 = new JTextField(10);
55
56   JLabel l8 = new JLabel("Uhrzeit Hinflug");
57   JTextField t8 = new JTextField(10);
58
59   JLabel l9 = new JLabel("Datum Rueckflug");
60   JTextField t9 = new JTextField(10);
61
62   JLabel l10 = new JLabel("Uhrzeit Rueckflug");
63   JTextField t10 = new JTextField(10);
64
65   JPanel p1 = new JPanel();
66   JPanel p2 = new JPanel();
67
68   JButton b1 = new JButton("OK");
69
70   JButton b2 = new JButton("Abbrechen");
71
72   /**
```

```
73     * This is the default constructor
74     */
75    public BuchungsGui() {
76      super("Menü");
77      initialize();
78    }
79    /**
80     * This method initializes this
81     *
82     * @return void
83     */
84    private void initialize() {
85  //    this.setSize(200,200);
86      this.setContentPane(getJContentPane());
87
88    }
89    /**
90     * This method initializes jContentPane
91     *
92     * @return javax.swing.JPanel
93     */
94    private javax.swing.JPanel getJContentPane() {
95      if(jContentPane == null) {
96        jContentPane = new javax.swing.JPanel();
97        jContentPane.setLayout(new java.awt.GridLayout(11,2));
98
99        c3.add("Bitte auswählen:"); c3.add("Lufthansa");
100       c3.add("Hapag Lloyd"); c3.add("British Airways"); c3.add("Air France");
101       c3.addItemListener(new ItemListener() {
102         public void itemStateChanged(ItemEvent e) {
103           fluggesellschaft = c3.getSelectedItem();
104         }
105       });
106
107       c5.add("Bitte auswählen:"); c5.add("Hamburg"); c5.add("Berlin");
108       c5.add("München"); c5.add("Düsseldorf"); c5.add("Hannover");
109       c5.add("Frankfurt (Main)"); c5.add("Leipzig");
110       c5.addItemListener(new ItemListener() {
111         public void itemStateChanged(ItemEvent e) {
112           abflughafen = c5.getSelectedItem();
113         }
114       });
115
116       c6.add("Bitte auswählen:"); c6.add("London"); c6.add("Paris");
117       c6.add("Bangkok"); c6.add("Kairo"); c6.add("Rio de Janeiro");
118       c6.add("Moskau"); c6.add("Tokio"); c6.add("Sydney");
119       c6.add("Madrid"); c6.add("Rom"); c6.add("Budapest");
120       c6.add("Hongkong"); c6.add("Dubai"); c6.add("Habana");
121       c6.add("Nairobi"); c6.add("Abu Dhabi"); c6.add("Kapstadt");
122       c6.add("New York"); c6.add("Riad"); c6.add("Ankara");
123       c6.addItemListener(new ItemListener() {
124         public void itemStateChanged(ItemEvent e) {
125           zielflughafen = c6.getSelectedItem();
126         }
127       });
128
129       b1.addActionListener(this);
130       b1.setToolTipText("Auswahl bestätigen");
131       p1.add(b1);
132
133       b2.addActionListener(this);
134       b2.setToolTipText("Menü verlassen");
135       p2.add(b2);
136
137       jContentPane.add(l1);
138       jContentPane.add(t1);
```

```
139       jContentPane.add(l2);
140       jContentPane.add(t2);
141       jContentPane.add(l3);
142       jContentPane.add(c3);
143       jContentPane.add(l4);
144       jContentPane.add(t4);
145       jContentPane.add(l5);
146       jContentPane.add(c5);
147       jContentPane.add(l6);
148       jContentPane.add(c6);
149       jContentPane.add(l7);
150       jContentPane.add(t7);
151       jContentPane.add(l8);
152       jContentPane.add(t8);
153       jContentPane.add(l9);
154       jContentPane.add(t9);
155       jContentPane.add(l10);
156       jContentPane.add(t10);
157       jContentPane.add(p1);
158       jContentPane.add(p2);
159       pack();
160       setVisible(true);
161
162     }
163     return jContentPane;
164   }
165
166   public void actionPerformed(ActionEvent event)
167   {
168     String cmd = event.getActionCommand();
169 //     try {
170       if (cmd.equals("OK")) {
171         nachname = t1.getText();
172         vorname = t2.getText();
173         sitze = t4.getText();
174         hinflugdatum = t7.getText();
175         hinuhrzeit = t8.getText();
176         rueckflugdatum = t9.getText();
177         rueckuhrzeit = t10.getText();
178
179         String buchungsdaten[] = {nachname, vorname,
180             fluggesellschaft, sitze, abflughafen,
181             zielflughafen, hinflugdatum, hinuhrzeit,
182             rueckflugdatum, rueckuhrzeit};
183
184         try{
185           BuchungsbestService myService = new BuchungsbestServiceLocator();
186           Buchungsbest myPort = myService.getBuchungsbest();
187
188           rueckgabe = myPort.ausgabe(nachname, vorname,
189               fluggesellschaft, sitze, abflughafen,
190               zielflughafen, hinflugdatum, hinuhrzeit,
191               rueckflugdatum, rueckuhrzeit);
192
193           System.out.println("Name: \t\t\t" + rueckgabe[1] + " " + rueckgabe[0]);
194           System.out.println("Fluggesellschaft: \t" + rueckgabe[2]);
195           System.out.println("Sitze: \t\t\t" + rueckgabe[3]);
196           System.out.println("Flughafen: \t\t" + rueckgabe[4]);
197           System.out.println("Zielflughafen: \t\t" + rueckgabe[5]);
198           System.out.println("Hinflugdatum: \t\t" + rueckgabe[6]);
199           System.out.println("Uhrzeit: \t\t" + rueckgabe[7]);
200           System.out.println("Rueckflugdatum: \t" + rueckgabe[8]);
201           System.out.println("Uhrzeit: \t\t" + rueckgabe[9]);
202           System.out.println("Bestätigungscode: \t" + rueckgabe[10] + "\n");
203
204           WindowListener l = new WindowAdapter() {
```

```
205             public void windowClosing(WindowEvent e) {
206               System.exit(0);
207             }
208           };
209
210           }catch(Exception e){
211             System.out.println("Fehler: " + e);
212           }
213
214       } else if (cmd.equals("Abbrechen")) {
215         System.exit(0);
216       }
217   }
218
219   public static void main(String[] args) {
220     BuchungsGui gui = new BuchungsGui();
221     gui.setLocation(100, 100);
222     gui.pack();
223     gui.setVisible(true);
224   }
225 }  //  @jve:visual-info  decl-index=0 visual-constraint="85,54"
```

Quelltext 18.3: Flugbuchungsbestätigung: Client BuchungsbestAnfrage

19. Wertpapierkurs

19.1. Ziel des Web Services

Als Ziel des Web Services zur Ermittlung eines Wertpapierkurses ist die Möglichkeit eines Benutzers, von seinem PC zu Hause auf den Web Service zuzugreifen. Es besteht dann für den Benutzer die Möglichkeit, mit einem in Java geschriebenen Client-Programm oder einem Servlet den aktuellen Aktienkurs einer Aktie angezeigt zu bekommen.

19.2. Beschreibung des Web Services

Mit dem Web Service soll es einem Benutzer möglich sein, einen Wertpapierkurs ermitteln zu können. Die verschiedenen Aktien und Kurse werden auf dem BizWeb-Server in einer MySQL-Datenbank gespeichert und dienen als Datengrundlage. Der Web Service selber wird ebenfalls neben der MySQL-Datenbank auf dem BizWeb-Server implementiert. Damit die Schnittstellen für den Web Service der Öffentlichkeit zugänglich gemacht werden können, wird der Web Servce im UDDI-Verzeichnis von Microsoft veröffentlicht.

Die Abfrage der Daten eines Wertpapiers soll dem Benutzers mittels eines Java-Client-Programms von seinem PC zu Hause möglich sein. Damit besteht die Möglichkeit sich aus einer Liste von Aktien eine auszuwählen und dann durch Klick auf den Button den Kurs für das ausgewählte Wertpapier anzeigen zu lassen.

19.3. Erstellen und Füllen der Tabellen der MySQL-Datenbank

Zur Vereinfachung des „nervigen“ Anlegens und Füllens der Datenbank mit Tabellen und Werten, kann folgender Code z.B. per Konsole oder „MySQL Control Center“ ausgeführt werden:

```
1 -- Datenbank anlegen für die Aktienkurse
2 CREATE DATABASE IF NOT EXISTS ‘BizWeb‘
3
4
5 -- Tabelle fuer die Aktien anlegen
6 CREATE TABLE ‘aktien‘
```

```
7  (
8    `AktienID`  int UNSIGNED  NOT NULL AUTO_INCREMENT ,
9    `AktienName`  varchar(50)   NOT NULL DEFAULT '' ,
10   PRIMARY KEY (`AktienID`)
11 ) TYPE=MYISAM PACK_KEYS=DEFAULT ROW_FORMAT=DEFAULT
12
13
14 -- Tabelle fuer die Kurse anlegen
15 CREATE TABLE `kurse`
16 (
17   `KursID`  int UNSIGNED  NOT NULL AUTO_INCREMENT ,
18   `KursDatum`   date      NOT NULL DEFAULT 'CURDATE' ,
19   `AktienID`  int       NOT NULL DEFAULT '' ,
20   `Kurs`    decimal     NOT NULL DEFAULT '' ,
21   PRIMARY KEY (`KursID`)
22 ) TYPE=MYISAM PACK_KEYS=DEFAULT ROW_FORMAT=DEFAULT
23
24
25 -- Aktie-Tabelle mit Daten fuellen
26 INSERT INTO aktien (AktienName)
27 VALUES ('SAP');
28 INSERT INTO aktien (AktienName)
29 VALUES ('Lufthansa');
30 INSERT INTO aktien (AktienName)
31 VALUES ('Infineon');
32 INSERT INTO aktien (AktienName)
33 VALUES ('Telekom');
34 INSERT INTO aktien (AktienName)
35 VALUES ('Commerzbank');
36 INSERT INTO aktien (AktienName)
37 VALUES ('Siemens');
38
39
40 -- Kurs-Tabelle mit Daten fuellen
41 INSERT INTO kurse (Datum, AktienID, Kurs)
42 VALUES ('2004.09.17',1,'130.01');
43 INSERT INTO kurse (Datum, AktienID, Kurs)
44 VALUES ('2004.09.17',2,'9.92');
45 INSERT INTO kurse (Datum, AktienID, Kurs)
46 VALUES ('2004.09.17',3,'8.46');
47 INSERT INTO kurse (Datum, AktienID, Kurs)
48 VALUES ('2004.09.17',4,'14.63');
49 INSERT INTO kurse (Datum, AktienID, Kurs)
50 VALUES ('2004.09.17',5,'15.33');
51 INSERT INTO kurse (Datum, AktienID, Kurs)
52 VALUES ('2004.09.17',6,'60.770');
53 INSERT INTO kurse (Datum, AktienID, Kurs)
54 VALUES ('2004.09.17',7,'35.20');
```

Quelltext 19.1: Aktienkurs: Datenbank anlegen

Nun ist die Aktiendatenbank fertig angelegt und mit Beispieldaten gefüllt. Die Aktiendaten stammen von: http://kurse.boerse.de/quotes.php3?text=liste&liste=dax30neu

19.3.1. Per Java mit der Datenbank verbinden

Die wichtigsten Eckdaten für eine ODBC-Verbindung mit der angelegten Datenbank „`bizweb`“:

```
String driverClass   = "com.mysql.jdbc.Driver";
String database = "jdbc:mysql://localhost:3306/bizweb";
String user     = "bizweb";
String password = "test";
ResultSet  result;
String     url;
String     sql ;
Connection con  = null;
Statement  stmt;
url = database;

try {
Class.forName(driverClass);
con = DriverManager.getConnection(url,user,password);
}
catch(SQLException e)
{ System.out.println("Fehler beim Verbindungsaufbau!"); }
catch(ClassNotFoundException e)
{ System.out.println("JDBC over ODBC Treiber nicht gefunden!"); }
```

Quelltext 19.2: Aktienkurs: ODBC–Verbindung

19.4. Das Java-Programm

19.4.1. Die Klasse StockWS.java (der Web Service als lokaler Client)

```
package StockWS;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class StockWS
{
    public String showResult(String stock)
    {
        // Variablen für die DB-Verbindung
        String driverClass   = "com.mysql.jdbc.Driver";
        String database = "jdbc:mysql://localhost:3306/bizweb";
        String user     = "root";  // wird nicht benötigt
        String password = "";  // wird nicht benötigt
        ResultSet  result;
        String     url;
        String     sql ;
        Connection con  = null;
        Statement  stmt;

        // member-Variablen
        boolean filled = false;
```

```
26          double price = 0;
27          String WSResult = null;
28
29          // Adresse der zu verbindenden Datenbank
30          url = database;
31
32          // Treiber laden und Verbindung herstellen
33          try
34          {
35              // JDBC over ODBC Treiber laden
36              Class.forName(driverClass);
37
38              // Verbindung mit der Datenbank aufnehmen
39              con = DriverManager.getConnection(url, user, password);
40          }
41          catch(SQLException e)
42          {
43              System.out.println("Fehler beim Verbindungsaufbau!");
44              System.exit(0);
45          }
46          catch(ClassNotFoundException e)
47          {
48              System.out.println("JDBC over ODBC Treiber nicht gefunden!");
49              System.exit(0);
50          }
51
52          // Einlesen und Importieren der gesamten Daten aus der Excel-Tabelle
53          try
54          {
55              // SQL-Select-Befehl
56              sql = "SELECT k.Kurs FROM Aktie a JOIN Kurs k " +
57                  "ON k.AktieID = a.AktieID " +
58                  "WHERE a.Aktie='"+stock+"'";
59
60              // Statement erstellen um SQL-Befehle auszuführen
61              stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
62                                         ResultSet.CONCUR_READ_ONLY);
63
64              // SQL-Befehl ausführen
65              result = stmt.executeQuery(sql);
66
67              // Ergebnisse der SQL-Abfrage bearbeiten
68              while(result.next())
69              {
70                  filled = true;  // auf true setzen, damit nur ein Ergebnis
71                  price = result.getDouble(1);
72              }
73
74              WSResult = new String(String.valueOf(price));
75
76              // Verbindungen schließen
77              result.close();
78              stmt.close();
79              con.close();
80          }
81          catch(SQLException e)
82          {
83              System.out.println("Fehler bei der Ausgeben!");
84          }
85
86          return WSResult;
87      }
88  }
```

Quelltext 19.3: Aktienkurs: StockWS.java

19.4.2. Die Klasse CallWS_Stock.java (Web Service als Servlet)

```
1  import java.io.IOException;
2  import java.io.PrintWriter;
3  import javax.servlet.ServletException;
4  import javax.servlet.http.HttpServlet;
5  import javax.servlet.http.HttpServletRequest;
6  import javax.servlet.http.HttpServletResponse;
7
8  public class CallWS_Stock extends HttpServlet
9  {
10     //~ Methods
11     // ----------------------------------------------------------------
12
13     public void doGet(HttpServletRequest request, HttpServletResponse response)
14             throws ServletException, IOException {
15         response.setContentType("text/html");
16         PrintWriter out = response.getWriter();
17         //String aktienliste = null;
18         String gewaehlteAktie = request.getParameter("aktienliste");
19
20         out.print("<html><head><title>");
21         out.print("Web Service: Aktienkurse aus MySQL-Datenbank anzeigen");
22         out.println("</title></head>");
23         out.println("<body bgcolor=#0E115E><center>");
24         out.print("<H1><font color='#FFFFFF'>");
25         out.println("Kurs eines Wertpapiers ermitteln</font></H1>");
26         out.println("<br></br>");
27         out.println("<form method='get' action=''>");
28         out.println("<br><font color='#FFFFFF'>Wählen Sie die Aktie aus !</br>");
29         out.println("<select style='width:230px' name='aktienliste'>");
30         out.println("<option selected value='SAP'>SAP</option>");
31         out.println("<option value='Lufthansa'>Lufthansa</option>");
32         out.println("<option value='Infineon'>Infineon</option>");
33         out.println("<option value='Telekom'>Telekom</option>");
34         out.println("<option value='Commerzbank'>Commerzbank</option>");
35         out.println("<option value='Siemens'>Siemens</option>");
36         out.println("</select>");
37         out.println("<br></br></font>");
38         out.println("<input type='submit' value='Kurs ausgeben'>");
39         out.println("</form>");
40
41         if (gewaehlteAktie != null) {
42             StockClient sc = new StockClient(gewaehlteAktie);
43             String d = null;
44             try {
45                 d = sc.getCourse();
46             } catch (Exception e) {
47                 e.printStackTrace();
48             }
49             out.println("<big><font color='#FFFFFF'>");
50             out.println("Der Kurs liegt bei " + d + "</font></big>");
51         }
52         out.println("</body></html>");
53     }
54
55     public void doPost(HttpServletRequest request, HttpServletResponse response)
56             throws ServletException, IOException {
57         doGet(request, response);
58     }
59 }
```

Quelltext 19.4: Aktienkurs: CallWS_Stock.java

19.4.3. Programmdokumentation

Der gesamte Web Service für die Anzeige der Aktienkurse aus einer MySQL-Datenbank besteht grundlegend aus sieben Klassen, die die Programmlogik, den Aufruf des Web Services und das GUI für den Web Service beinhalten.

Das Java-Servlet wird nicht weiter erläutert, weil der Aufruf des Web Services prinzipiell genauso erfolgt, wie beim Client.

Klasse `Stock.java`

Die Klasse „`Stock`“ bildet das Herzstück des Web Services. Sie liegt in einem speziellen Verzeichnis des Tomcat-Servers. In ihr wird zu allererst eine ODBC-Verbindung zur MySQL-Datenbank aufgebaut, welche sich ebenfalls auf dem BizWeb-Server befindet, wie derTomcat-Server. Daraufhin wird die Datenbank mittels einer SQL-Abfrage auf den Kurs des vom Benutzer anzugebenden Wertpapiers abgefragt. Der Wert der Aktie wird auf der Befehlszeile ausgegeben.

Das Client (-GUI)

Mit der grafischen Oberfläche erhält der Web Service-Anwender eine benutzerfreundliche Schnittstelle für den Web Service. Hier kann der Benutzer in einer Combo-Box die Aktie auswählen, zu der er den Kurs ermitteln möchte. Klickt er auf den Button, wird der Web Service ausgeführt und das Ergebnis angezeigt.

Das Servlet

Das Java-Servlet sieht für den Benutzer aus wie eine Internetseite die auf dem BizWeb-Server mit Tomcat-Server ausgeführt wird. Die Funktionen entsprechen denen des Clients zuvor.

19.5. Den Web Service per UDDI veröffentlichen

Zur Veröffentlichung wird der Web Service im UDDI-Verzeichnis von Microsoft veröffentlicht. Ist dieses geschehen und sucht jemand nach dem Begriff „biz“, erscheint folgendes Ergebnis:

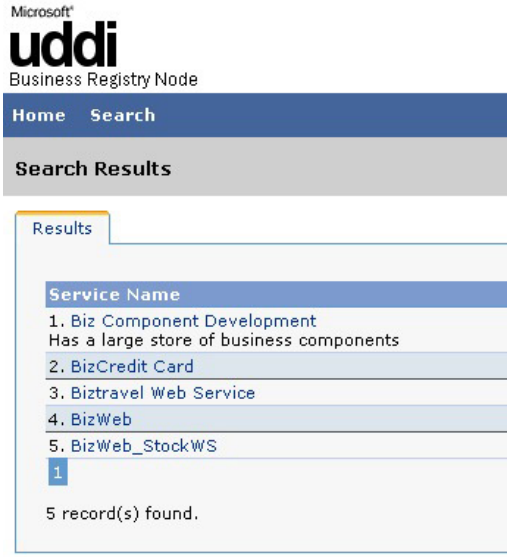


Abbildung 19.1.: Aktienkurs: Microsoft UDDI Suchergebnis

Nach Klick auf „BizWeb_StockWS“ erscheint diese Information...

Abbildung 19.2.: Aktienkurs: Details des UDDI-Eintrages

. . . und zum Schluss unter „Bindings“ die URL:

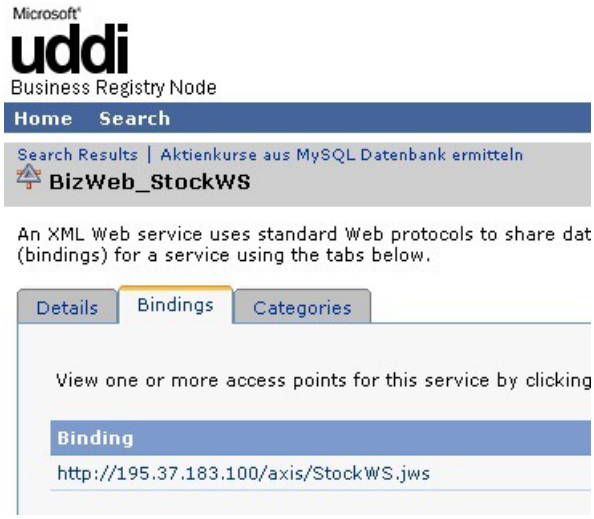


Abbildung 19.3.: Aktienkurs: UDDI Bindings

19.6. Der Informationsfluss zwischen Client und Server

Der Client „weis“ durch die erzeugte WSDL-Datei, mit welchem Server bzw. welcher IP er sich verbinden muss. Der ServiceLocator verbindet sich mit dem Server. Nun kann der Benutzer eine Aktie auswählen. Nach dem Klick auf den Button wird die Information – der ausgewählte String (= der Wertpapiername) an den Web Service auf dem Server übermittelt. Der Web Service verbindet sich mit der lokalen MySQL-Datenbank auf dem Server, sucht den aktuellen Kurs heraus und liefert ihn zurück an den Client, der das Ergebnis dem Benutzer anzeigt.

Zur Darstellung und Dokumentation des Informationsflusses zwischen dem lokalen Client und dem BizWeb-Server wird das Programm *Ethereal* verwendet. Es zeichnet alle Pakete auf, die zwischen beiden Instanzen ausgetauscht werden.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	217.225.153.1	195.37.183.100	TCP	2439 > http [SYN] Seq=0 Ack=0 Win=16384 Len=0 MSS=1440
2	0.054639	195.37.183.100	217.225.153.1	TCP	http > 2439 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
3	0.063592	217.225.153.1	195.37.183.100	TCP	2439 > http [ACK] Seq=1 Ack=1 Win=17280 Len=0
4	0.065563	217.225.153.1	195.37.183.100	HTTP	POST /axis/StockWS.jws HTTP/1.0
5	0.163444	195.37.183.100	217.225.153.1	TCP	http > 2439 [ACK] Seq=1 Ack=732 Win=6579 Len=0
6	0.179380	195.37.183.100	217.225.153.1	HTTP	HTTP/1.1 200 OK
7	0.180214	195.37.183.100	217.225.153.1	TCP	http > 2439 [FIN, ACK] Seq=708 Ack=732 Win=6579 Len=0
8	0.188621	217.225.153.1	195.37.183.100	TCP	2439 > http [ACK] Seq=732 Ack=709 Win=16573 Len=0
9	0.193210	217.225.153.1	195.37.183.100	TCP	2439 > http [FIN, ACK] Seq=732 Ack=709 Win=16573 Len=0
10	0.248658	195.37.183.100	217.225.153.1	TCP	http > 2439 [ACK] Seq=709 Ack=733 Win=6579 Len=0

```
⊞ Frame 6 (769 bytes on wire, 769 bytes captured)
⊞ Ethernet II, Src: 00:90:1a:10:10:64, Dst: 00:04:0e:03:1e:2e
⊞ PPP-over-Ethernet Session
⊞ Point-to-Point Protocol
⊞ Internet Protocol, Src Addr: 195.37.183.100 (195.37.183.100), Dst Addr: 217.225.153.1 (217.225.153.1)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 2439 (2439), Seq: 1, Ack: 732, Len: 707
⊞ Hypertext Transfer Protocol
```

Abbildung 19.4.: Aktienkurs: Übersicht einer WS-Anfrage

In den ersten drei Zeilen finden der Verbindungsaufbau und die Synchronisation der TCP-Verbindung statt. Danach ruft der Client per HTTP-POST den Web-Service `StockWS.jws` auf und versorgt diesen per SOAP mit den nötigen Parametern. Es folgt ein Ausschnitt aus dem Body des SOAP-Envelope, in dem zu sehen ist, welche Parameter mit welchen Werten übergeben werden.

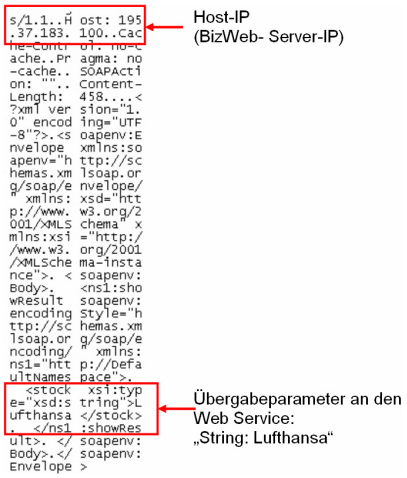


Abbildung 19.5.: Aktienkurs: Ethereal Parameter

Der Server bestätigt in Zeile 5 über TCP mit „`ACK`“ den Erhalt des Pakets. Nach Ausführung des Web-Services wird im Body eines HTTP-Response-Pakets ein SOAP-Envelope, welcher unter anderem den Rückgabeparameter enthält, an den Client gesendet (Zeile 6). Hier ein Ausschnitt des Envelopes:

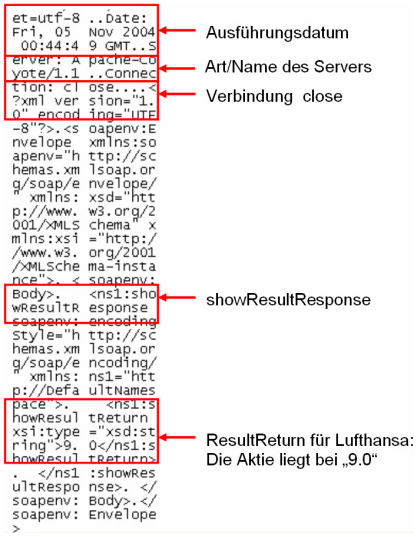


Abbildung 19.6.: Aktienkurs: Ethereal Results

Der Client erhält so das Ergebnis über das HTTP-Protokoll und kann es weiterverarbeiten. In den Zeilen 7-10 wird die TCP-Verbindung wieder abgebaut.

19.7. Aufruf des Web Services mit Parametern über die URL

Mit Hilfe des Servlets ist es möglich, den Web Service mit Parameter über die URL in einem Browser aufzurufen.

In diesem Fall wird zuerst die URL benötigt:
http://195.37.183.100/StockWS/servlet/CallWS_Stock

An diese wird folgendes angehängt: ?aktienliste=Telekom

Damit kann der Wertpapierkurs der Telekom aufgerufen und das Ergebnis ausgegeben werden.

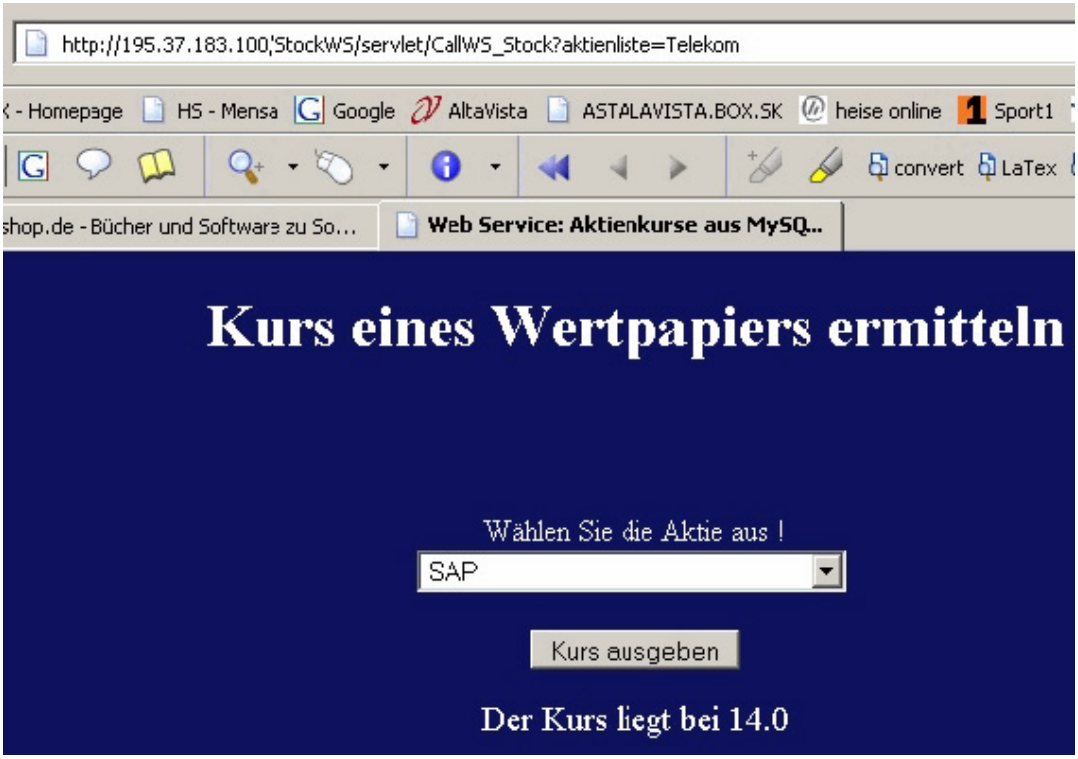


Abbildung 19.7.: Aktienkurs: Web Service Servlet per Parameter aufrufen

19.8. Eine Beispielsitzung mit dem Servlet

Der Web Service ist fertig auf dem BizWeb-Server implementiert, doch wie sieht eine Nutzung des Web Services aus?

Abbildung 19.8.: Aktienkurs: Web Service aufrufen

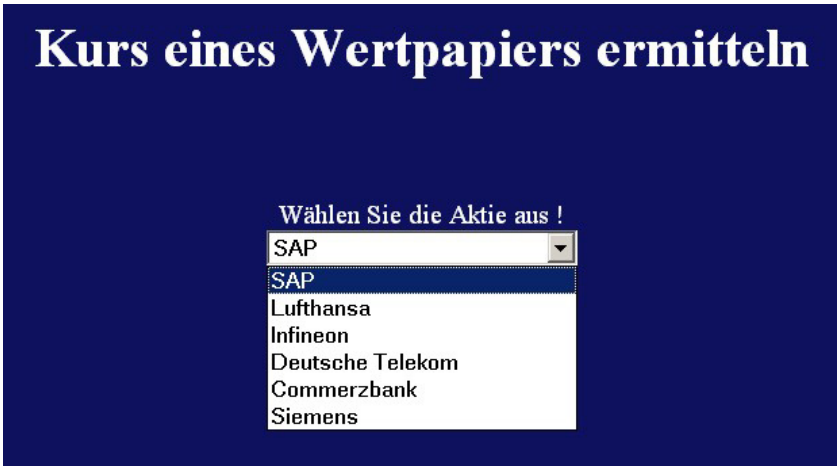


Abbildung 19.9.: Aktienkurs: Auswahl der Wertpapiere

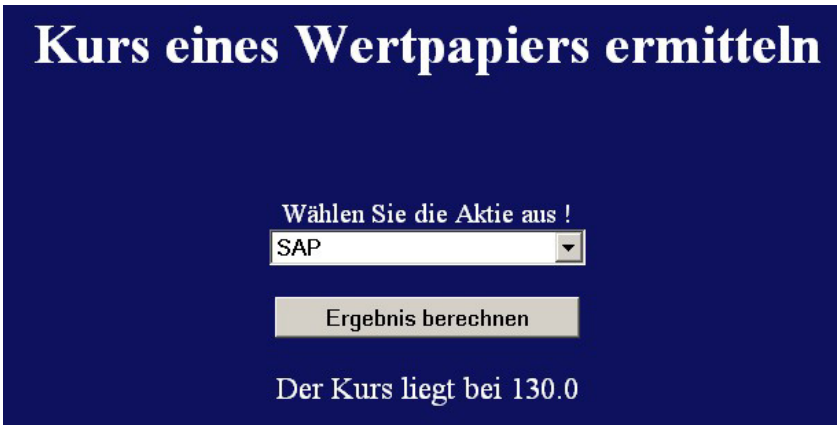


Abbildung 19.10.: Aktienkurs: Das Ergebnis

19.9. Fazit

Speziell diese Art von Hausarbeiten zum ersten richtigen Verstehen und Vertiefen der Materie sind sehr lehrreich. Wenn auch der Arbeitsaufwand wesentlich höher ist gegenüber einer „normalen“ theoretischen Ausarbeitung.

Es macht Spaß sich in das Thema reinzuarbeiten und die ersten Erfolge, aber auch Misserfolge zu erleben. Letztere verhelfen zu immer tieferen Wissen über die Materie, und auch die Hintergründe, was wann warum passiert, werden dadurch wesentlich klarer. In der reinen Theorie – wie z.B. in der vorhergegangenen Schulung – kam es nie zu bestimmten Fehler.

Eine Zusammenarbeit von zwei Personen ist durchaus ratsam und sehr hilfreich. Die Fehler, die der eine gemacht hat, können so bei der zweiten Person sofort gelöst oder verhindert werden.

20. Kreuzfahrtliste

20.1. Aufgabenstellung

Im Rahmen des BizWeb-Projektes an der Hochschule Bremerhaven soll von den Projektteilnehmern ein eigener Java Web Service unter Axis implementiert werden. Diese Ausarbeitung beschreibt eine mögliche Implementierung für ein Web Service-Szenario, in dem eine Liste von Schiffsrouten zu einem Schiff zurückliefert wird. Die erforderlichen Informationen sind dazu aus einer MySQL-Datenbank auszulesen. Das Ergebnis ist in einem Swing GUI zu präsentieren. Der Web Service soll abschließend auf dem BizWeb-Server zu veröffentlicht werden.

20.2. Aufbau einer Entwicklungsumgebung

In diesem Abschnitt soll die Einrichtung einer lauffähigen Umgebung für den Web Service unter *SuSe Linux 8.2* beschrieben werden. Die Implementierung des Web Services basiert auf Grundlage der folgenden Technologien:

- *Apache Tomcat* (in der Version 5), ein Open-Source-Applikationsserver (Referenzimplementierung für Servlets und Java-Server-Pages)
- *Apache Axis*, die Open-Source-Implementierung des Web Service-Standards SOAP
- *MySQL-Datenbank*, eine webfähige Open-Source-Datenbank

20.2.1. Tomcat-Installation

Zunächst ist der Tomcat-Applikationsserver zu installieren. Als Installationspfad wurde hierzu `/usr/java/tomcat` gewählt, der auch in der Umgebungsvariable `CATALINA_HOME` bekannt zu machen ist.

Mit SuSe Linux kann hierzu die Datei `/etc/profile.local` um die Anweisung `export CATALINA_HOME=/usr/java/tomcat` erweitert werden. Falls die Datei noch nicht vorhanden ist, kann sie in der Regel problemlos erstellt werden (ggf. sind root-Rechte erforderlich). Bei erfolgreicher Installation sollte auf einem lokalen System der Server über die URL http://localhost:8080 erreichbar sein.

20.2.2. Axis-Installation

Die Axis-Webanwendung kann auf sehr einfache Weise unter Tomcat installiert werden. Es genügt bereits, das Verzeichnis `webapps/axis` in das Tomcat-Anwendungsverzeichnis `[CATALINA_HOME]/webapps` zu kopieren und Tomcat neu zu starten. In der Standardkonfiguration ist die Axis-Webanwendung jetzt unter der folgenden URL zu erreichen: http://localhost:8080/axis/index.html Die Konfigurationsumgebung ist im Verzeichnis `[CATALINA_HOME]/webapps/axis/WEB-INF/lib` zu finden und muss mindestens folgende jar-Dateien umfassen:

- `axis.jar`
- `axis-ant.jar`
- `commons-discovery.jar`
- `commons-logging.jar`
- `jaxrpc.jar`
- `log4j-1.2.8.jar`
- `saaj.jar`
- `wsdl4j.jar`

Das Axis-Framework ist ggf. um fehlende Pakete zu erweitern. Auf der Startseite von Axis kann der Link „Validate the local installation's configuration“ zur Prüfung der vollständigen Konfigurationsumgebung genutzt werden. Weiter sind als Umgebungsvariablen die folgenden Pfade in der Datei `/etc/profil.local` zu setzen:

```
export AXIS_HOME       =  /usr/java/tomcat/webapps/axis
export AXIS_LIB        =  $AXIS_HOME/WEB-INF/lib
export AXIS_CLASSPATH  =  $AXIS_LIB/axis.jar:
                          $AXIS_LIB/axis-ant.jar:
                          $AXIS_LIB/commons-discovery.jar:
                          $AXIS_LIB/commons-logging.jar:
                          $AXIS_LIB/jaxrpc.jar:
                          $AXIS_LIB/log4j-1.2.8.jar:
                          $AXIS_LIB/saaj.jar:
                          $AXIS_LIB/wsdl4j.jar
```

20.2.3. Einrichtung einer MySQL-Datenbank

Nach erfolgreicher Installation der MySQL-Datenbank-Binaries kann die Datenbank bizweb erstellt werden. Die Datenbank soll Informationen über Schiffsrouten umfassen. Unter Beachtung der Normalisierung sind folgende Tabellen mit ihren Attributen zu erstellen:

```
CREATE TABLE `bizweb`.`routes` (
  `route_id` SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `ship_id` SMALLINT NOT NULL REFERENCES `bizweb`.`ships`,
  `route_from` SMALLINT NOT NULL REFERENCES `bizweb`.`ports`,
  `route_to` SMALLINT NOT NULL REFERENCES `bizweb`.`ports`
);

CREATE TABLE `bizweb`.`ports` (
 `port_id` SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
 `port_name` VARCHAR(30) NOT NULL
);

CREATE TABLE `bizweb`.`ships` (
 `ship_id` SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
 `ship_name` VARCHAR(30) NOT NULL,
 `shipping_company` VARCHAR(30) NOT NULL
);
```

Quelltext 20.1: Kreuzfahrtliste: Erstellen der Tabellen

Die Einrichtung der Datenbank kann mit der Erstellung eines guest-Accounts mit `SELECT`-Priviliegen für *bizweb* abgeschlossen werden.

Um eine JDBC-Verbindung zur Datenbank vom Applikationsserver aus zu ermöglichen, ist im Verzeichnis `[CATALINA_HOME]/common/lib` die Connector-Bibliothek[1] abzulegen.

20.3. Implementierung des eigenen Web Services unter Axis

Dieser Abschnitt behandelt die Entwicklung und das Deployment einer Axis–Webanwendung, sowie die Programmierung eines Clients.

20.3.1. Implementierung des Service-Providers

Auf der Provider-Seite soll eine Schnittstelle (Service) implementiert werden, die einem Service-Consumer (Client) eine bestimmte Dienstleistung über das Netzwerk zur Verfügung stellt. Der Service kann sowohl als eine einfache Java-Routine als auch als komplexe Host-Transaktion implementiert werden. In diesem Beispiel soll ein Service mit Java implementiert werden, der Schiffsrouten aus der MySQL-Datenbank zurückliefert.

[1] `mysql-connector-java-3.0.15-ga-bin.jar`

Die Daten werden in SOAP-Nachrichten verpackt und über das separate Transportprotokoll versendet. Hierfür wird meistens HTTP verwendet (wie auch in diesem Beispiel), alternativ wäre auch ein Einsatz von SMTP, FTP oder TCP möglich.

Die Provider-Klasse RouteList

Die Provider-Klasse RouteList ist Bestandteil des Packages `de.bizweb.uebung` und dient der eigentlichen Implementierung der Service-Schnittstelle. Über die öffentliche Methode `getList(String paramShip)` werden die Daten in einem Array der Bean-Klasse `ListElement` (vgl. Abschnitt 20.3.1) zurückgeliefert. `paramShip` ist das Selektionskriterium, zu dem die Informationen aus der MySQL-Datenbank ausgelesen werden sollen.

Nachdem eine Datenbank-Verbindung aufgebaut wurde, können die Schiffsrouten über ein `preparedStatement` mit folgender Query selektiert werden:

```
1 SELECT r.route_id,s.shipping_company,s.ship_name, p.port_name,
2        p2.port_name
3   FROM routes r JOIN ships s ON r.ship_id=s.ship_id
4                 JOIN ports p ON r.route_from=p.port_id
5                 JOIN ports p2 ON r.route_to=p2.port_id
6   WHERE s.ship_name = ?
```

Quelltext 20.2: Kreuzfahrtliste: Query

Das Datenbankergebnis wird zunächst in einem `ResultSet` gespeichert, dann in eine dynamische Datenstruktur umgewandelt (in diesem Beispiel wurde ein `Vector` gewählt), bevor es schließlich in das `ListElement`-Array überliefert wird. Es existieren sicher bessere Möglichkeiten der Konvertierung, doch soll dieser Ansatz für die Lösung der Aufgabenstellung ausreichen.

Der komplette Quellcode der Klasse `RouteList` ist im Anhang aufgeführt.

Die Bean-Klasse ListElement

Die Bean-Klasse ListElement dient der Repräsentation des Datenbankergebnisses. Die Spalten sind unterschiedliche (primitive) Datentypen, die in einem komplexen Datentyp zusammengefasst werden sollen. Ein Objekt dieser Beanklasse repräsentiert also genau eine Zeile des Datenbankresultats. Gemäß der Bean-Spezifikationen wurden bei der Implementierung ein Standard-Konstruktor, sowie die `getter`- und `setter`-Methoden berücksichtigt.

Das gesamte Ergebnis kann schließlich problemlos als Array dieser Objekte gespeichert und zurückgeliefert werden. Wie diese Objekte in SOAP serialisiert werden, wird im Abschnitt 20.3.2 behandelt.

Der komplette Quellcode der Klasse `ListElement` ist im Anhang aufgeführt.

Die Exception-Klasse NoSuchShipException

Anwendungsspezifische Fehler werden in der Exception-Klasse `NoSuchException` abgefangen und clientseitig als AXIS-Faults verwendet. `NoSuchShipException` wird von der Klasse `RemoteException` abgeleitet. Eine entsprechende Exception-Instanz wird geworfen, wenn keine Daten zu einem Schiff ermittelt werden können. Die SOAP-Response enthält im Header-Element `fault` die Informationen `faultcode`, `faultstring` und `detail`. Die Fehlerbehandlung muss bei der Implementierung des Clients erfolgen.

20.3.2. Deployment unter Axis

Axis unterstützt zwei Deployment-Mechanismen. Durch den Einsatz sog. JWS-Dateien (Java Web Service) kann eine einfache Form von Deployment erreicht werden. Dazu sind die Quellcode-Dateien der als Web Service zu veröffentlichenden Java-Klasse einfach in das Wurzelverzeichnis der Axis-Anwendung zu kopieren und mit der Dateiendung `.jws` zu versehen. Beim Neustart von Tomcat führt die Axis-Anwendung dann ein Auto-Deployment durch.

Eine wesentlich flexiblere Möglichkeit für den Entwickler ist die zweite Alternative — das Deployment mit Hilfe einer WSDD-Datei (Web Service Deployment Descriptor) zu vollziehen. Die WSDD-Datei ist eine auf XML basierende Konfigurationsdatei, in der die spezifischen Deployment-Informationen enthalten sind. Die Integration des Services erfolgt auf Grundlage der bereits vorkompilierten Provider-Klassen. Die `.class` Dateien sind dazu in das Verzeichnis `[CATALINA_HOME]/webapps/axis/WEB-INF/classes` zu kopieren. Dabei ist ggf. auch die sich aus dem Package ergebende Verzeichnisstruktur nachzubilden (z.B. für das Package `de.bizweb.uebung` die Ordner `de`, dann darunter `bizweb` und abschließend `uebung`).

Für den Web Service ist folgende WSDD-Datei zu erstellen:

```
1  <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3
4    <service name="RouteList" provider="java:RPC">
5      <parameter name="className"
6        value="de.bizweb.uebung.RouteList"/>
7
8      <parameter name="allowedMethods" value="*"/>
9
10     <typeMapping qname="myNS:ListElement"
11       xmlns:myNS="http://uebung.bizweb.de"
12       languageSpecificType="java:de.bizweb.uebung.ListElement"
13       serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
14       deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
15       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
16   </service>
17
18 </deployment>
```

Quelltext 20.3: Kreuzfahrtliste: Deploy.wsdd

Die Descriptor-Datei umfasst Namespaces, den Service-Namen, die zu instanzierende Java-Klasse des Services (und deren aufrufbare Methoden) und in diesem speziellen Fall auch Angaben zum „Type-Mapping“. Das Java-XML-Mapping definiert, wie ein Java-Objekt in XML-Format dargestellt werden kann. Das Mapping ist erforderlich, um die Objekte mit SOAP „auf die Reise“ schicken zu können. Im Wesentlichen werden folgende Java-Typen werden von JAX-RPC unterstützt, so dass kein explizites Type-Mapping in der WSDD-Datei erforderlich ist:

- primitive Datentypen: `boolean`, `byte`, `short`, `int`, `long`, `float` und `double`
- Standardklassen wie `java.util.Date` oder `java.lang.String`
- Arrays, deren Elementtypen von JAX-RPC unterstützt werden

Das Ergebnis aus der MySQL-Datenbank wird in Form eines Arrays der Bean-Klasse `ListElement` zurückgeliefert (vgl. Abschnitt 20.3.1). Eine Instanz von `ListElement` bildet genau eine Reihe des Abfrageergebnisses aus der Datenbank ab. Sie wird als komplexer Datentyp verstanden, für den ein explizites Type-Mapping erforderlich ist. Damit eine Java-Klasse von der JAX-RPC-Runtime in XML-Format umgewandelt werden kann, muss diese Klasse einen JAX-RPC-Value-Typ darstellen. Ein JAX-RPC-Value-Typ muss im Wesentlichen folgende Bedingungen erfüllen:

- die Klasse muss einen öffentlichen `Defaultkonstruktor` besitzen
- die Klasse muss JavaBean-konform sein (d.h. die Properties müssen über `getter`-Methoden zugreifbar bzw. über `setter`-Methoden veränderbar sein und der jeweilige Property-Typ muss ein von JAX-RPC unterstützter Datentyp sein)
- die Klasse kann `public`-, `protected`- und `private`-Attribute besitzen, wobei jedes `public`-Attribut einen JAX-RPC-kompatiblen Datentyp haben muss
- die Klasse darf nicht `java.rmi.Remote` implementieren

Alle diese Bedingungen erfüllt die Klasse `ListElement`. Die Serialisierung von Value-Typen in JAX-RPC ist nur für öffentliche Attribute und JavaBean-Properties (Attribute, die `getter` und `setter`-Methoden besitzen) möglich — als `protected` und `private` deklarierte Attribute bleiben dagegen unberücksichtigt. Die Bean Klasse muss auch nicht explizit das Interface `java.io.Serializable` implementieren, was sich von der Standard-Serialisierung in Java unterscheidet. Die Serialisierung bzw. Deserialisierung wird von den standardmäßig von Axis zur Verfügung gestellten Factories[2] vorgenommen.

[2] `org.apache.axis.encoding.ser.BeanSerializerFactory`
bzw.
`org.apache.axis.encoding.ser.BeanDeserializerFactory` (vgl. WSDD-Datei)

Das Deployment wird letztlich mit dem Admin-Client ausgeführt. Dazu ist in der Konsole der Befehl `java org.apache.client.AdminClient Pfad_zur_Deploy.wsdd` einzugeben. Nach erfolgreichem Deployment sollte der Web Service jetzt lokal unter der URL http://localhost:8080/axis/services/RouteList erreichbar sein.

Ein Undeployment kann mit folgender WSDD-Datei erfolgen:

```
1 <undeployment xmlns="http://xml.apache.org/axis/wsdd/"
2   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3
4   <service name="RouteList"/>
5
6 </undeployment>
```

Quelltext 20.4: Kreuzfahrtliste: Undeploy.wsdd

Der entsprechende Konsolenbefehl lautet dann:
`java org.apache.client.AdminClient Pfad_zur_Undeploy.wsdd`

20.3.3. Generierung der Stubs aus dem WSDL-Dokument

Der Aufbau des WSDL-Dokuments

Das `definitions`-Element dient als Container für die Servicebeschreibung, es ist das Wurzelelement. Hier werden die Namespaces definiert.

Das `types`-Element ist optional und dient der Definition von komplexen Datentypen. Einfache Datentypen wie `integer`, `float`, `double`, `date` und `string` sind bereits im XML-Schema definiert. Oft reichen diese jedoch nicht aus, um z.B. eine bestimmte Applikationslogik abzubilden.

Mit `message`-Elementen werden die Nachrichten beschrieben, die zwischen Client und Web Service beim Aufruf einer Operation ausgetauscht werden. Die verwendeten Datentypen stammen entweder aus dem XML-Schema oder wurden im `types`-Element definiert. Das Aufrufen einer Methode bewirkt den Transport von zwei Nachrichten: dem Funktionsaufruf an den Server (Request) und der Antwort an den Client (Response).

Im Abschnitt `portType` werden die Operationen des Web Services beschrieben. Jedes Element namens `operation` entspricht dabei einer von außen zugänglichen Operation des Web Services.

Über das `bindings`-Element wird für jede Operation der Web Service-Schnittstelle (`portType`) das verwendete Nachrichtenformat, also der Aufbau des SOAP-Body, und die Kodierung der in den Operationen ausgetauschten Daten sowie das Transportprotokoll festgelegt.

Das `service`-Element ist das letzte Element der WSDL-Beschreibung. Es dient der Definition konkreter Netzwerkadressen, also an welche Netzwerkadressen sich der Client wenden muss, um mit dem Web Service zu kommunizieren.

Für den Web-Service kann man sich das WSDL-Dokument automatisch über die URL http://localhost:8080/axis/services/RouteList?wsdl erstellen lassen:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/RouteList"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/services/RouteList"
  xmlns:intf="http://localhost:8080/axis/services/RouteList"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns1="http://uebung.bizweb.de"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:types>
      <schema targetNamespace="http://uebung.bizweb.de"
        xmlns="http://www.w3.org/2001/XMLSchema">
          <import
            namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
              <complexType name="ListElement">
                <sequence>
                  <element name="portFrom" nillable="true"
                           type="xsd:string"/>
                  <element name="portTo" nillable="true"
                           type="xsd:string"/>
                  <element name="routeID" type="xsd:int"/>
                  <element name="shipName" nillable="true"
                           type="xsd:string"/>
                  <element name="shippingCompany" nillable="true"
                           type="xsd:string"/>
                </sequence>
              </complexType>
              <complexType name="NoSuchShipException">
                <sequence>
                  <element name="cause" nillable="true"
                           type="xsd:anyType"/>
                  <element name="message" nillable="true"
                           type="xsd:string"/>
                </sequence>
              </complexType>
      </schema>
      <schema targetNamespace=
                   "http://localhost:8080/axis/services/RouteList"
          xmlns="http://www.w3.org/2001/XMLSchema">
         <import namespace=
                    "http://schemas.xmlsoap.org/soap/encoding/"/>
            <complexType name="ArrayOf_tns1_ListElement">
               <complexContent><restriction base="soapenc:Array">
                  <attribute ref="soapenc:arrayType"
                      wsdl:arrayType="tns1:ListElement[]"/>
                      </restriction>
              </complexContent>
            </complexType>
      </schema>
    </wsdl:types>
    <wsdl:message name="getListRequest">
    <wsdl:part name="paramShip" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getListResponse">
    <wsdl:part name="getListReturn"
        type="impl:ArrayOf_tns1_ListElement"/>
  </wsdl:message>
  <wsdl:message name="NoSuchShipException">
    <wsdl:part name="fault" type="tns1:NoSuchShipException"/>
  </wsdl:message>
  <wsdl:portType name="RouteList">
```

```
64      <wsdl:operation name="getList" parameterOrder="paramShip">
65        <wsdl:input message="impl:getListRequest"
66                    name="getListRequest"/>
67        <wsdl:output message="impl:getListResponse"
68                     name="getListResponse"/>
69        <wsdl:fault message="impl:NoSuchShipException"
70                    name="NoSuchShipException"/>
71      </wsdl:operation>
72    </wsdl:portType>
73    <wsdl:binding name="RouteListSoapBinding" type="impl:RouteList">
74      <wsdlsoap:binding style="rpc"
75                transport="http://schemas.xmlsoap.org/soap/http"/>
76      <wsdl:operation name="getList">
77        <wsdlsoap:operation soapAction=""/>
78        <wsdl:input name="getListRequest">
79          <wsdlsoap:body
80             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
81             namespace="http://uebung.bizweb.de" use="encoded"/>
82        </wsdl:input>
83        <wsdl:output name="getListResponse">
84          <wsdlsoap:body
85             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
86             namespace="http://localhost:8080/axis/services/RouteList"
87             use="encoded"/>
88        </wsdl:output>
89        <wsdl:fault name="NoSuchShipException">
90          <wsdlsoap:fault
91             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
92             namespace="http://localhost:8080/axis/services/RouteList"
93             use="encoded"/>
94        </wsdl:fault>
95      </wsdl:operation>
96    </wsdl:binding>
97    <wsdl:service name="RouteListService">
98      <wsdl:port binding="impl:RouteListSoapBinding" name="RouteList">
99        <wsdlsoap:address
100            location="http://localhost:8080/axis/services/RouteList"/>
101     </wsdl:port>
102   </wsdl:service>
103 </wsdl:definitions>
```

Quelltext 20.5: Kreuzfahrtliste: WSDL

Codegenerierung mit WSDL2Java

Mit dem Axis-Tool *WSDL2Java* können sowohl Java Proxies für die Clients als auch Codegerüste (Skeletons) für die Service-Implementierungen aus dem WSDL-Dokument generiert werden. Das Axis-Tool ist Teil des Packages `org.apache.axis.wsdl.WSDL2Java`. Mit dem Konsolen-Befehl

```
java org.apache.axis.wsdl.WSDL2Java Pfad_zum_wsdl-Dokument
```

werden die erforderlichen Klassen für den Client im aktuellen Verzeichnis erzeugt. Für diesen Web Service muss daher der folgende Befehl (im richtigen Verzeichnis) ausgeführt werden:

```
java org.apache.axis.wsdl.WSDL2Java http://localhost:8080/axis/services/RouteList?wsdl
```

Die folgenden Klassen und Interfaces werden so in der Verzeichnisstruktur bzw. im Package `localhost.axis.services` erstellt:

- Den Port-Type: `RouteList.java`
- Das Service-Interface: `RouteListService.java`
- Der Locator, der das Service-Interface implementiert: `RouteListServiceLocator.java`
- Den Binding-Stub (Proxy): `RouteListSoapBindingStub.java`

20.3.4. Implementierung eines Service-Consumers

Nachdem die Stubs mit WSDL2Java generiert wurden, können sie nun in einer Client-Applikation verwendet werden. Die Client-Applikation soll – genau wie der Service – in Java implementiert werden. Für die Benutzerschnittstelle soll eine grafische Oberfläche mit Swing (Swing-GUI) programmiert werden.

Der Service-Consumer wurde im Package `de.bizweb.uebung.application` implementiert. Die (grafische) Benutzeroberfläche ist in der Klasse `Start` realisiert. Die Eingabe des Schiffsnamen (Selektionskriterium) kann in einem `jTextField` vorgenommen werden. Bei Betätigen des Refresh-Buttons wird der Web Service (mit dem aus `jTextField` übergebenen Parameter) aufgerufen und das Ergebnis in einer `jTable` ausgegeben. In einer Statuszeile im unteren Fensterbereich wird der Benutzer über Erfolgs- und Fehlermeldungen informiert.

Für die Ausgabe in der `jTable` werden die Daten zunächst in einem Table-Model aufbereitet. Die entsprechende Implementierung erfolgt in der Klasse `TableControlModel`, die wiederum von der Klasse `AbstractTableModel` abgeleitet ist. In der `TableControlModel`-Klasse existiert die Methode `getListData(String paramShip)`, mit der der Web Service aufgerufen wird. Die `jTable` der Klasse Start wird mit den Daten einer `TableControlModel`-Instanz gefüllt.

Der Aufruf des Web Services erfolgt mit Hilfe der Stub-Klassen:

```
1 ...
2 public ListElement[] getListData(String paramShip) throws Exception{
3       RouteListService myService = new RouteListServiceLocator();
4       RouteList myPort = myService.getRouteList();
5
6       resultList = myPort.getList(paramShip);
7       return resultList;
8 ...
```

Quelltext 20.6: Kreuzfahrtliste: Client

Das Abfrageergebnis, also `ListElement[]`, ist schließlich für die Ausgabe in einer `jTable` in ein zweidimensionales Array vom Typ `Object` zu konvertieren. Die Konvertierung erfolgt direkt im Konstruktor. Exceptions (einschließlich der anwendungsspezifischen `NoSuchShipException`) werden an die Klasse `Start` übergeben und dort behandelt.

20.4. Dokumentation einer Beispielsitzung

In diesem Abschnitt soll eine Beispielsitzung beschrieben werden. In diesem Beispiel möchte der Kunde (Service-Consumer) Schiffsrouten für das Schiff „North Sea Pride“ vom Service-Provider erfragen. Hierzu startet er die Client-Applikation und gibt in die Zeile „Schiffsname“ den Namen des Schiffes, also „North Sea Pride“, ein.

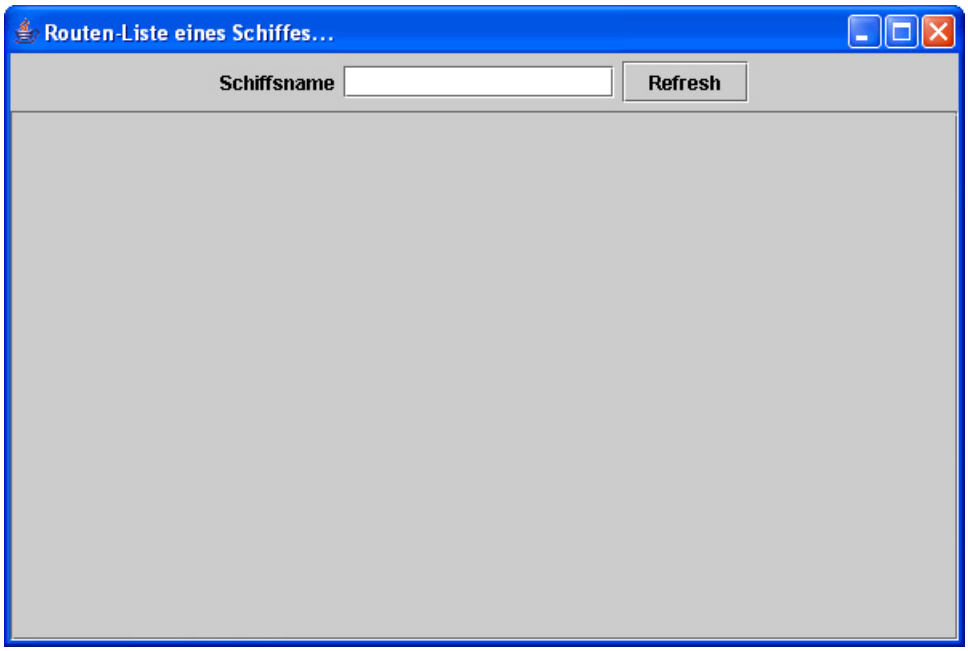


Abbildung 20.1.: Kreuzfahrtliste: GUI (1)

Durch Betätigen des „Refresh“-Buttons erhält er das Ergebnis in Form einer Tabelle und zudem die Erfolgsmeldung, „Verbindung zur Datenbank erfolgreich“, im unteren Teil des Fensters.

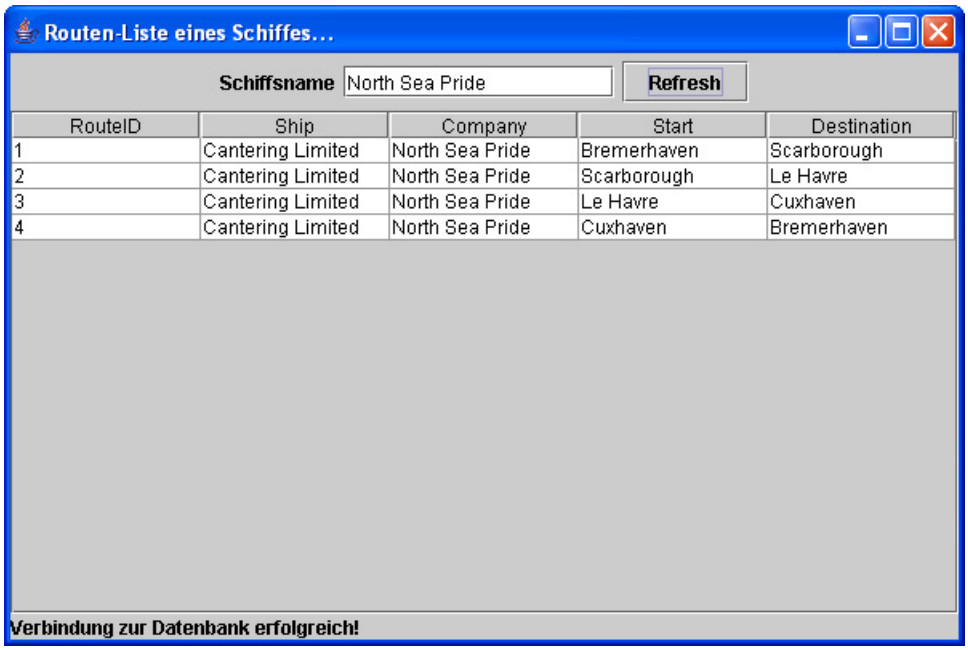


Abbildung 20.2.: Kreuzfahrtliste: GUI (2)

Falls keine Abfrageergebnisse ermittelt werden können, wird im unteren Fensterbereich die anwendungsspezifische Axis-Fault Message „de.bizweb.uebung.NoSuchShipException: Keine Ergebnisse zu dieser Eingabe“ ausgegeben:

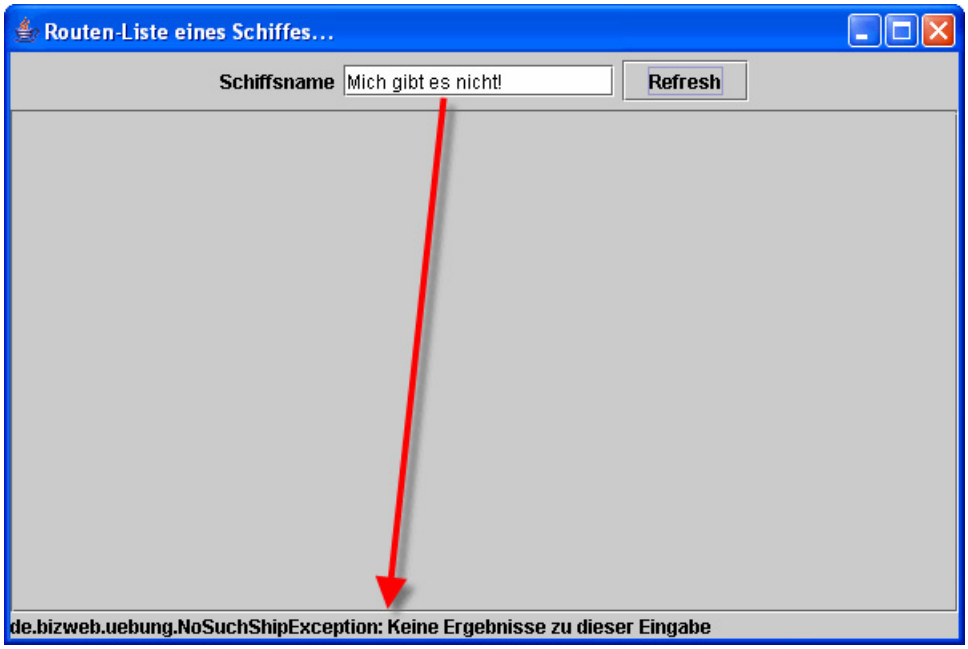


Abbildung 20.3.: Kreuzfahrtliste: GUI (3)

20.5. Resümee

Dieser Web Service beschreibt beispielhaft, wie Informationen aus einer Datenbank von einem Service-Provider über SOAP-Nachrichten zu einem Service-Consumer übermittelt werden können. Im Gegensatz zu anderen Projekt-Aufgaben kann das Abfrageergebnis nicht in einem primitiven Datentypen zurückgeliefert werden. Die Umsetzung dieser Teilaufgabe erforderte eine Lösung für einen komplexen Datentypen, der durch eine Bean-Klasse repräsentiert wird. Die Bean-Instanzen sind vor dem SOAP-Versand zu serialisieren und beim Empfang wieder zu deserialisieren. Hierfür können die BeanFactory-Klassen von Axis verwendet werden. Die erforderlichen Informationen hierfür werden über den Deployment-Deskriptor eingestellt. Ein Auto-Deployment ist für diesen Service nicht möglich.

Ein weiteres Feature ist die anwendungsspezifische Fehlerbehandlung. Statt der Rückgabe von `null` für die leere Ergebnismenge einer Abfrage wird eine selbst implementierte Exception geworfen.

Der Web Service ist in dieser (vereinfachten) Form für die Praxis sicher ungeeignet, da die Daten unverschlüsselt – und daher auch ungesichert – übermittelt werden. Maßnahmen gegen eine mögliche Manipulation der SOAP-Nachrichten sollen hier aber nicht weiter behandelt werden, da sie nicht Bestandteil der Aufgabenstellung sind.

21. Hotelbuchung

21.1. Ziel und Beschreibung des WebService

Es soll ein Java WebService unter Axis entwickelt werden der auf unseren BizWeb-Server implementiert werden soll. Der WebService „Hotelbuchung“ soll dabei die Buchungsdaten in einem String übergeben bekommen, dabei wird auf eine MySQL Datenbank zugegriffen und als Ergebnis wird zur Buchungsbestätigung eine BuchungsID zurückgeliefert.

Als GUI werden zwei Swing Fenster gleichzeitig geöffnet. Im ersten Fenster erfolgt die Eingabe und das Senden der Buchungsdaten, wobei in dem Zweiten die Buchungsbestätigung zurückgegeben wird. Die WSDL-Beschreibung des WebService wird durch das Auto-Deployment des AXIS-Frameworks generiert.

21.2. Erstellen einer MySQL Datenbank

Für die Aufgabe wird eine MySQL Datenbank erstellt. Hierzu habe ich das MySQL Control Center v0.9.4-beta verwendet Für die Verbindung zur Datenbank und Unterstützung aller Funktionen sind die folgenden Treiber zuständig:

- mysql-connector-java-3[1].0.15-ga.zip
- MyODBC-standard-3.51.9-win.exe

Im MySQL Control Center werden die vorher per Skript generierten Tabellen eingelesen:

```
1  CREATE DATABASE IF NOT EXISTS ‘bizweb‘;
2  use ‘bizweb‘;
3
4  create table IF NOT EXISTS ‘login‘ (
5  ‘LoginID‘ int(15)   UNSIGNED AUTO_INCREMENT PRIMARY KEY,
6  ‘KundeID‘   int(10)   NOT NULL DEFAULT ’’ references kunde,
7  ‘Password‘  varchar(10) NOT NULL DEFAULT ’’ references kunde);
8
9  create table IF NOT EXISTS ‘buchung‘ (
10 ‘BuchungID‘ int(3)    UNSIGNED AUTO_INCREMENT PRIMARY KEY,
11 ‘KundeID‘   int(10)   NOT NULL DEFAULT ’’ references kunde,
12 ‘HotelID‘   int(5)    NOT NULL DEFAULT ’’ references hotel,
13 ‘BDatum‘  date    NOT NULL DEFAULT ’CURDATE’,
14 ‘Zimmertyp‘   varchar(40)   NOT NULL DEFAULT ’’,
15 ‘Datum_von‘   date    NOT NULL DEFAULT ’CURDATE’,
```

```
16 `Datum_bis`   date    NOT NULL DEFAULT 'CURDATE');
17
18 create table IF NOT EXISTS `hotel` (
19 `HotelID`   int(5)    UNSIGNED AUTO_INCREMENT PRIMARY KEY,
20 `HName`   varchar(60)   NOT NULL DEFAULT '',
21 `HStrasse`  varchar(60)   NOT NULL DEFAULT '',
22 `HPLZ`  varchar(10) NOT NULL DEFAULT '',
23 `HOrt`    varchar(29)   NOT NULL DEFAULT '',
24 `HLand`   varchar(40)   NOT NULL DEFAULT '',
25 `HTelefon`  varchar(40)   NOT NULL DEFAULT '',
26 `HEmail`  varchar(40) NOT NULL DEFAULT '');
27
28 create table IF NOT EXISTS `kunde` (
29 `KundeID`   int(10)   UNSIGNED AUTO_INCREMENT PRIMARY KEY,
30 `Password`  varchar(10) NOT NULL DEFAULT '',
31 `KDName`  varchar(40)   NOT NULL DEFAULT '',
32 `KDVorName`   varchar(40)   NOT NULL DEFAULT '',
33 `KDStrasse`   varchar(40)   NOT NULL DEFAULT '',
34 `KDPLZ`   varchar(10) NOT NULL DEFAULT '',
35 `KDOrt`   varchar(29)   NOT NULL DEFAULT '',
36 `KDLand`  varchar(40) NOT NULL DEFAULT '',
37 `KDTelefon`   varchar(40)   NOT NULL DEFAULT '');
38
39 --insert in Tabelle Login
40 INSERT INTO login (KundeID, Password)
41 Values ('0000000005','654321');
42
43 --insert in Tabelle Buchung
44 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
45 VALUES ('00001', '0000000001', '2004.10.12', 'Doppelzimmer', '2005.06.15', '2005.06.30');
46 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
47 VALUES ('00002', '0000000002', '2004.10.15', 'Einzelzimmer', '2005.07.15', '2005.07.30');
48 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
49 VALUES ('00003', '0000000003', '2004.10.01', 'Doppelzimmer', '2005.08.03', '2005.08.18');
50 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
51 VALUES ('00004', '0000000004', '2004.09.19', 'Doppelzimmer', '2005.09.20', '2005.10.05');
52 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
53 VALUES ('00005', '0000000005', '2004.10.22', 'Doppelzimmer', '2005.06.18', '2005.07.02');
54 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
55 VALUES ('00006', '0000000006', '2004.09.06', 'Einzelzimmer', '2005.06.28', '2005.07.20');
56 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
57 VALUES ('00007', '0000000007', '2004.10.04', 'Einzelzimmer', '2005.08.04', '2005.08.22');
58 INSERT INTO buchung (HotelID, KundeID, BDatum, Zimmertyp, Datum_von, Datum_bis)
59 VALUES ('00008', '0000000008', '2004.10.12', 'Einzelzimmer', '2005.02.24', '2005.03.10');
60 --insert in Tabelle Hotel
61 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
62 VALUES ('Holiday Inn, CHICAGO-MART PLAZA (RIVERVIEW)','350 N ORLEANS','IL 60654',
63 'Chicago', 'USA', '(1)-312-836-5000', 'Roomreservations@martplaza.com');
64 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
65 VALUES ('Memory Manor Bed & Breakfast','514 Glen Street','NY 12801',
66 'Glens Falls', 'USA', '(1)-518-793-2699', 'mail@memorymanor.com');
67 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
68 VALUES ('Holiday Inn Select - NIAGARA FALLS','300 THIRD ST','NY 14303',
69 'NIAGARA FALLS', 'USA', '(1)-80095-FALLS', 'fomnia@lodgian.com');
70 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
71 VALUES ('Candlewood Suites - LAS VEGAS','4034 PARADISE RD.','NV 89109',
72 'LAS VEGAS', 'USA', '(1)-702-836-3660', 'adam.dougherty@ichotelsgoup.com');
73 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
74 VALUES ('Conrad Jupiters Gold Coast - Broadbeach Island', 'Broadbeach Island','4218',
75 'Queensland', 'Australia', '(61)-7-5592-1133', 'goldcoast@conradhotels.com');
76 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
77 VALUES ('Conrad Miami - Espirito Santo Plaza','1395 Brickell Avenue','FL 33131',
78 'Miami', 'USA', '(1)-305-503-6500', 'miamiinfo@conradhotels.com');
79 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
80 VALUES ('Hilton Bremen','Boettcherstrasse 2','NA DE 28195', 'Bremen',
81 'Germany', '(49)-421-36960', 'no e-mail');
```

```
82 INSERT INTO hotel (HName, HStrasse, HPLZ, HOrt, HLand, HTelefon, HEmail)
83 VALUES ('Marriott Hotel Hamburg','ABC Strasse 52','DE 20354', 'Hamburg',
84 'Germany', '(49)-40-35050', 'no e-mail');
85
86 --insert in Tabelle Kunde
87 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
88 KDTelefon)
89 VALUES ('ABCDEFG', 'Carman', 'Steve', '843 Wood Ave.', 'NJ 08820',
90 'Edison', 'USA', '(1)-732-499-3678');
91 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
92 KDTelefon)
93 VALUES ('123ABC', 'PClub', 'Bernie', 'Sesamstrasse 5', 'DE 48238',
94 'Sesamhausen', 'Germany', '(49)-435-454563');
95 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
96 KDTelefon)
97 VALUES ('123456', 'dasBrot', 'Bernd', 'Brotstrasse 93', 'DE 99933',
98 'Brotsdorf', 'Germany', '(49)-345-BROT');
99 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
100 KDTelefon)
101 VALUES ('098765', 'Future', 'Captain', 'Galaxystreet 4', 'NY 03374',
102 'New York', 'USA', '(1)-543-343-2435');
103 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
104 KDTelefon)
105 VALUES ('654321', 'Spok', 'Mister', 'Enterprisestr. 34', 'DE 45325',
106 'Raumschiffhausen', 'Germany', '(49)-403-114411');
107 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
108 KDTelefon)
109 VALUES ('CBA321', 'Simpson', 'Bart', 'Michiganstreet 9', 'SP 35675',
110 'Springfield', 'USA', '(1)-009-345-1211');
111 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
112 KDTelefon)
113 VALUES ('UVWXYZ', 'Brown', 'James', 'Brownistreet 2', 'CA 43068',
114 'San Jose', 'USA', '(1)-222-333-4444');
115 INSERT INTO kunde (Password, KDName, KDVorName, KDStrasse, KDPLZ, KDOrt, KDLand,
116 KDTelefon)
117 VALUES ('ZYXWVU', 'Schmidt', 'Harald', 'Schmidtstrasse 87', 'DE 53854',
118 'Berlin', 'Germany', '(49)-30-399222');
```

Quelltext 21.1: Weltuhrzeit: Datenbankerstellung

Die Datenbank kann genauso über die Konsole wie über das MySQL Control Center erstellt werden. Ich beschränke mich hierbei auf die Einrichtung im MySQL Control Center.

Es wir eine neue Datenbank „bizweb“ mit folgenden Zugangsdaten erstellt:

User Name: „**root**“
Password: „ “
Host Name: **localhost**

Die Datenbank wird Standardmäßig auf den Port 3306 gesetzt.

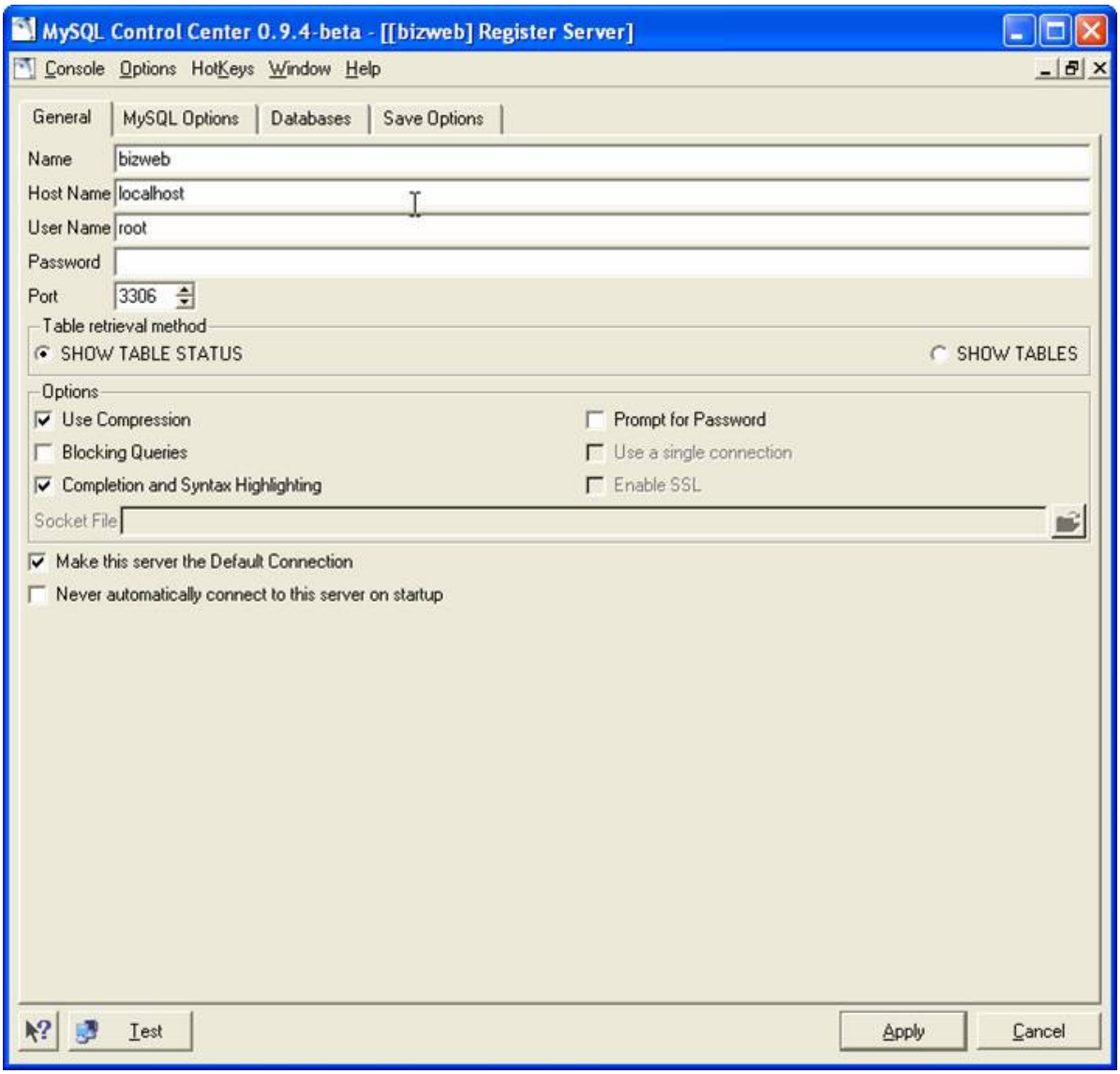


Abbildung 21.1.: Hotelbuchung: Erstellung der BizWeb Datenbank im MySQL Control Center

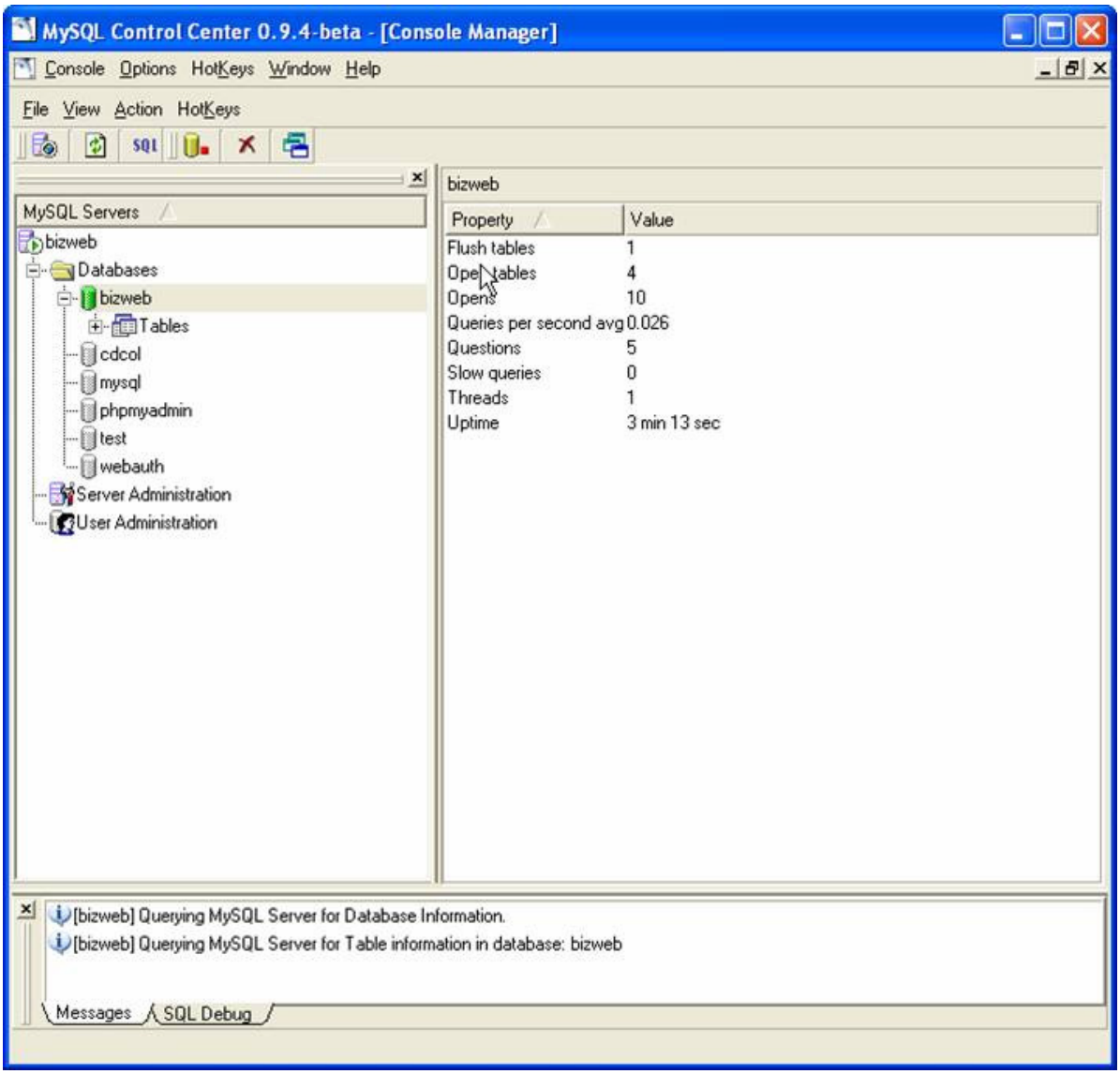


Abbildung 21.2.: Hotelbuchung: erstellte BizWeb Datenbank

Durch den SQL-Button können die vorbereiteten Tabellen als Skript eingefügt werden.

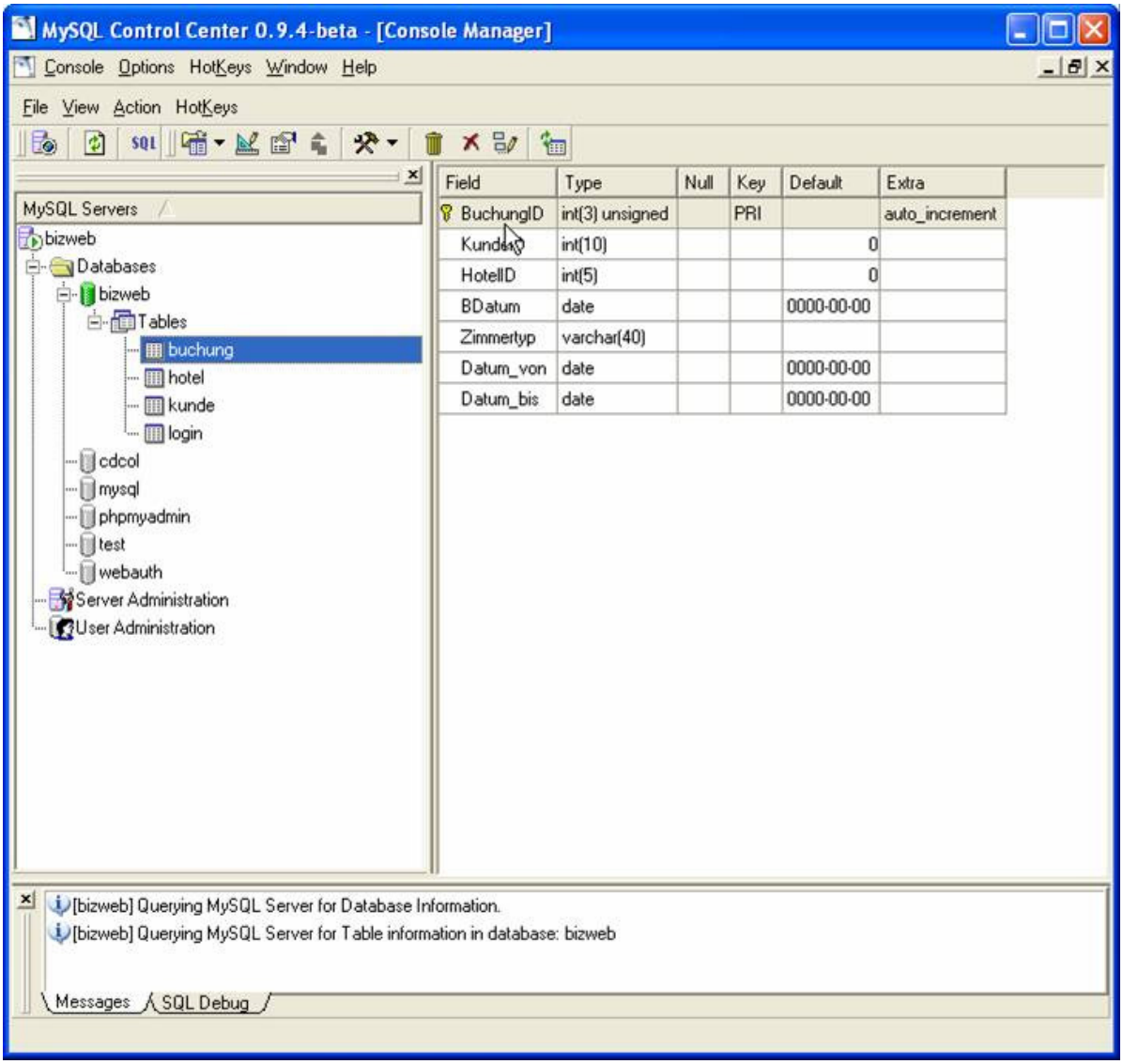
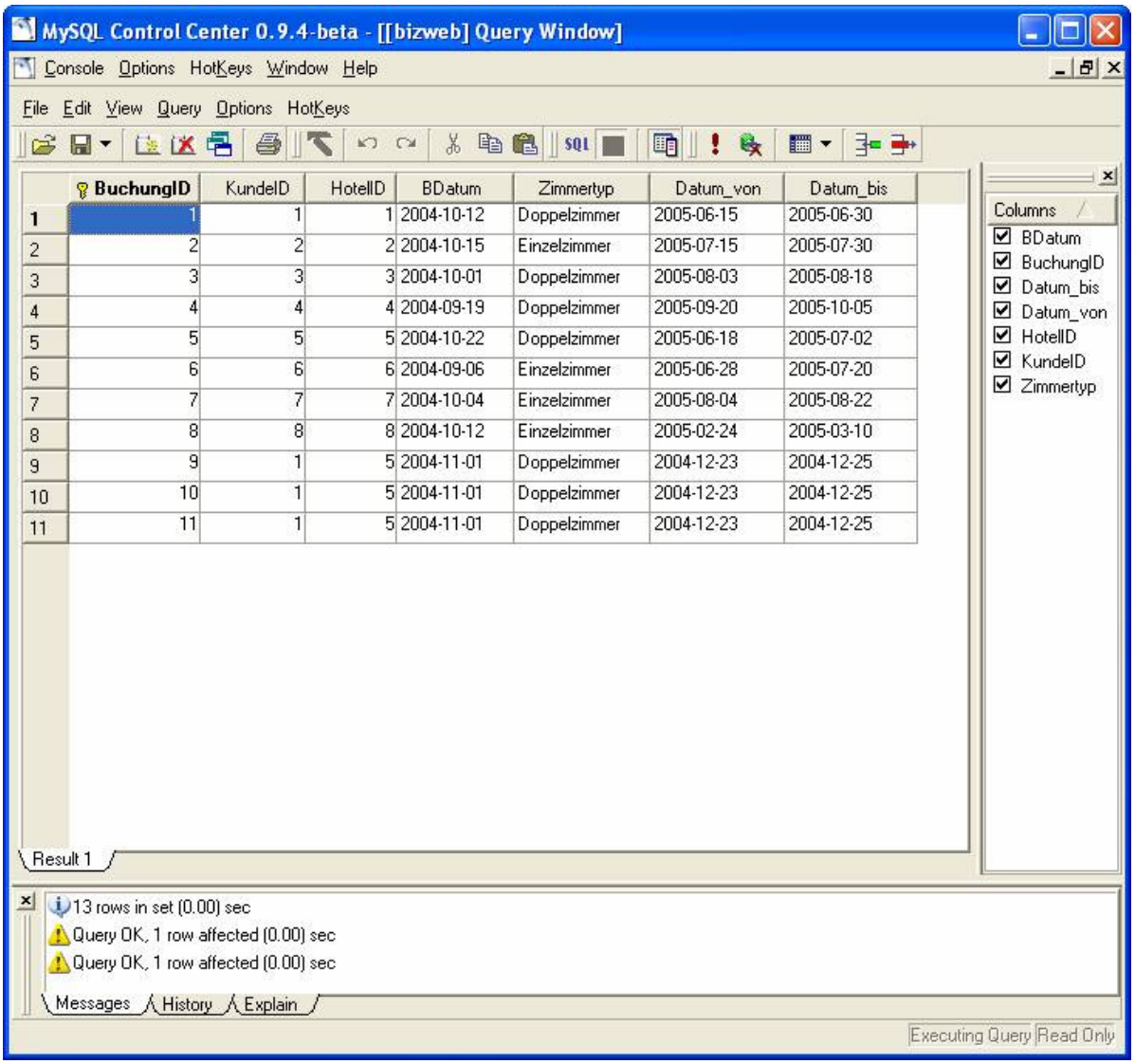


Abbildung 21.3.: Hotelbuchung: erstellte BizWeb Datenbank mit den Tabellen

Nach dem Import des Skripts sind die Tabellen „buchung“, „hotel“, „kunde“ und „login“ zu sehen.



	BuchungID	KundeID	HotelID	BDatum	Zimmertyp	Datum_von	Datum_bis
1	1	1	1	2004-10-12	Doppelzimmer	2005-06-15	2005-06-30
2	2	2	2	2004-10-15	Einzelzimmer	2005-07-15	2005-07-30
3	3	3	3	2004-10-01	Doppelzimmer	2005-08-03	2005-08-18
4	4	4	4	2004-09-19	Doppelzimmer	2005-09-20	2005-10-05
5	5	5	5	2004-10-22	Doppelzimmer	2005-06-18	2005-07-02
6	6	6	6	2004-09-06	Einzelzimmer	2005-06-28	2005-07-20
7	7	7	7	2004-10-04	Einzelzimmer	2005-08-04	2005-08-22
8	8	8	8	2004-10-12	Einzelzimmer	2005-02-24	2005-03-10
9	9	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25
10	10	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25
11	11	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25

Abbildung 21.4.: Hotelbuchung: BizWeb Datenbank - Inhalt der Buchungstabelle

Nachdem die Tabellen mit Ihren Inhalten erfolgreich eingeflegt wurden, muss nur noch die Umgebung für den TomCat, Eclipse und Axis eingerichtet werden. Axis wird im TomCat-Verzeichnis angelegt, wobei die unterschiedlichen Libraries in den TomCat eingefügt werden:

- axis.jar
- axis-1_1.zip
- Activation.jar
- mail.jar
- log4j-1.2.8.jar
- mysql-connector-java-3[1].0.15-ga-bin.jar
- xercesImpl.jar
- xercesSamples.jar
- xml-apis.jar
- xmlParserAPIs.jar
- UDDI4j.jar
- xmlsec.jar
- wsdl4j.jar

Daraufhin werden noch folgende Umgebungsvariablen gesetzt:

```
AXIS_HOME       =  C:\Java\tomcat\webapps\axis
AXIS_LIB        =  %AXIS_HOME%\lib
AXIS_CLASSPATH  =  %AXIS_LIB%\axis.jar;
                   %AXIS_LIB%\axis-ant.jar;
                   %AXIS_LIB%\commons-discovery.jar;
                   %AXIS_LIB%\commons-logging.jar;
                   %AXIS_LIB%\jaxrpc.jar;
                   %AXIS_LIB%\log4j-1.2.8.jar;
                   %AXIS_LIB%\saaj.jar;
                   %AXIS_LIB%\wsdl4j.jar;
                   %AXIS_LIB%\xercesImpl.jar;
                   %AXIS_LIB%\xercesSamples.jar;
                   %AXIS_LIB%\xml-apis.jar;
                   %AXIS_LIB%\xmlParserAPIs.jar
JAVA_HOME       =  Pfad zum JDK, z.B. C:\Java\AppServer\jdk
CLASSPATH       =  %AXIS_CLASSPATH%
```

21.3. Beschreibung des WebService Hotelbuchung

Der WebService „Hotelbuchung“ besteht aus einer Klasse der einen String übergeben bekommt, diesen in die Datenbank einfügt und dabei zur Bestätigung die BuchungsID zurückliefert.

Zunächst wird eine Verbindung zur Datenbank hergestellt

```
1  //  Stellt eine Verbindung zur Datenbank her
2  public static void DBVerbindung() throws SQLException {
3    try {
4      Class.forName("com.mysql.jdbc.Driver");
5    }
6    catch (ClassNotFoundException e1) {
7      e1.printStackTrace();
8    }
9    con = DriverManager.getConnection("jdbc:mysql://localhost/bizwebbuchung","root","");
10   System.out.println("Verbindung hergestellt");
11 }
```

Quelltext 21.2: Hotelbuchung: Datenbank-Verbindung

Daraufhin folgt die Buchungsmethode „BookConfirm“. Diese beinhaltet das Einfügen der Buchung in die Datenbank und das Auslesen der Buchungsdaten zur Buchungsbestätigung, wobei nur die BuchungsID hierbei zurückgeliefert wird.

```
1  String query = "insert into buchung (kundeID, hotelID, bdatum, zimmertyp, " +
2          "datum_von, datum_bis) values (?,?,?,?,?,?)";
3        PreparedStatement grBuchungConfirm = con.prepareStatement(query);
4        grBuchungConfirm.setString(1, kundeID);
5        grBuchungConfirm.setString(2, hotelID);
6        grBuchungConfirm.setString(3, buchungsDatum);
7        grBuchungConfirm.setString(4, zimmerTyp);
8        grBuchungConfirm.setString(5, datumVon);
9        grBuchungConfirm.setString(6, datumBis);
10       grBuchungConfirm.executeUpdate();
```

Quelltext 21.3: Hotelbuchung: Buchungsdaten in die Datenbank einfügen

```
1
2  String query2 = "select BuchungID from buchung where KundeID = ? and HotelID = ?" +
3             "and BDatum = ? and Zimmertyp = ? and Datum_von = ? and Datum_bis = ?";
4        PreparedStatement grBuchungGetLast = con.prepareStatement(query2);
5        grBuchungGetLast.setString(1, kundeID);
6        grBuchungGetLast.setString(2, hotelID);
7        grBuchungGetLast.setString(3, buchungsDatum);
8        grBuchungGetLast.setString(4, zimmerTyp);
9        grBuchungGetLast.setString(5, datumVon);
10       grBuchungGetLast.setString(6, datumBis);
11
12       ResultSet rs = grBuchungGetLast.executeQuery();
13       while(rs.next())
14         {
15          BIDconf = rs.getInt(1);
```

Quelltext 21.4: Hotelbuchung: Buchungsdaten aus der Datenbank lesen

21.4. Vorgehensweise bei Erstellung des Webservice

Um den WebService zu erstellen muss nun die oben kurz vorgestellte Java-Datei „Hotelbuchung.java“ in eine Jws-Datei umbenannt werden. Diese wird dann in das tomcat/webapps/axis Verzeichnis abgelegt. Im Browser wird nun die Datei aufgerufen und somit die Wsdl-Datei generiert.

http://localhost:8080/axis/hotelbuchung.jws?wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://195.37.183.100/axis/Hotelbuchung.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://195.37.183.100/axis/Hotelbuchung.jws"
xmlns:intf="http://195.37.183.100/axis/Hotelbuchung.jws"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:message name="BookConfirmResponse">
    <wsdl:part name="BookConfirmReturn" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="BookConfirmRequest">
    <wsdl:part name="kundeID" type="xsd:string"/>
    <wsdl:part name="hotelID" type="xsd:string"/>
    <wsdl:part name="buchungsDatum" type="xsd:string"/>
    <wsdl:part name="zimmerTyp" type="xsd:string"/>
    <wsdl:part name="datumVon" type="xsd:string"/>
    <wsdl:part name="datumBis" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="DBVerbindungRequest">
  </wsdl:message>
  <wsdl:message name="DBVerbindungResponse">
  </wsdl:message>
  <wsdl:portType name="Hotelbuchung">
    <wsdl:operation name="BookConfirm" parameterOrder="kundeID hotelID
    buchungsDatum zimmerTyp datumVon datumBis">
      <wsdl:input message="impl:BookConfirmRequest" name="BookConfirmRequest"/>
      <wsdl:output message="impl:BookConfirmResponse" name="BookConfirmResponse"/>
    </wsdl:operation>
    <wsdl:operation name="DBVerbindung">
      <wsdl:input message="impl:DBVerbindungRequest" name="DBVerbindungRequest"/>
      <wsdl:output message="impl:DBVerbindungResponse" name="DBVerbindungResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HotelbuchungSoapBinding" type="impl:Hotelbuchung">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="BookConfirm">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="BookConfirmRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="BookConfirmResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://195.37.183.100/axis/Hotelbuchung.jws" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="DBVerbindung">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="DBVerbindungRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace" use="encoded"/>
```

```
55        </wsdl:input>
56        <wsdl:output name="DBVerbindungResponse">
57          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
58          namespace="http://195.37.183.100/axis/Hotelbuchung.jws" use="encoded"/>
59        </wsdl:output>
60      </wsdl:operation>
61    </wsdl:binding>
62    <wsdl:service name="HotelbuchungService">
63      <wsdl:port binding="impl:HotelbuchungSoapBinding" name="Hotelbuchung">
64        <wsdlsoap:address location="http://195.37.183.100/axis/Hotelbuchung.jws"/>
65      </wsdl:port>
66    </wsdl:service>
67  </wsdl:definitions>
```

Quelltext 21.5: Hotelbuchung: WSDL-Beschreibung des Webservice Hotelbuchung

21.5. Die Entwicklung des Clients

Die WSDL Code Generierung wird nun als Wsdl-Datei abgespeichert und in das Eclipse Projekt eingefügt. Eclipse wird nun aktualisiert und die Wsdl-Datei mit dem Plugin WSDL2Java wieder in Java Dateien umgewandelt. Es entstehen folgende Klassen:

- Hotelbuchung.java
- HotelbuchungService.java
- HotelbuchungServiceLocator.java
- HotelbuchungSoapBindingStub.java

Der Aufruf des WebService und die Übergabe des Strings erfolgt über folgende Zeilen:

```
1  public HotelBuchungClient() throws Exception{
2    HotelbuchungService MyService = new HotelbuchungServiceLocator();
3    Hotelbuchung MyPort = MyService.getHotelbuchung();
4    int test = MyPort.bookConfirm("'1"',
5                                  "'5"',
6                                  "'2004-11-01"',
7                                  "'Doppelzimmer"',
8                                  "'2004-12-23"',
9                                  "'2004-12-25"');
```

Quelltext 21.6: Hotelbuchung: Übergabe der Daten an den WebService

Der eigentliche Aufruf des WebService erfolgt in der grafischen Swing Oberfläche BookConfirm.java:

```
1  package _100._183._37._195.axis.Hotelbuchung_jws;
2  import javax.swing.JFrame;
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
```

```
7  public class BookConfirm extends JFrame {
8    private JPanel jContentPane = null;
9    private JLabel jLabel1 = null;
10   private JTextField jTFieldBID = null;
11   private JButton jButton = null;
12   private JButton jButton1 = null;
13   private int BIDconf;
14
15   public BookConfirm(){
16       super();
17       init();
18   }
19
20   //  Erzeugt das Fenster
21   public void init() {
22     this.setSize(300,100);
23     this.setTitle("Buchungsbestätigung");
24     this.setContentPane(getJContentPane());
25     this.setResizable(false);
26     this.setVisible(true);
27   }
28
29   //  Erzeugt die ContentPane
30   private JPanel getJContentPane() {
31     if(jContentPane == null) {
32       jContentPane = new JPanel();
33       jContentPane.add(getJLabel1(), null);
34       jContentPane.add(getjTFieldBID(), null);
35       jContentPane.add(getJButton(), null);
36       jContentPane.add(getJButton1(), null);
37       jContentPane.setLayout(new GridLayout(2,2));
38     }
39     return jContentPane;
40   }
41
42   //  Erzeugt das Label das zur Eingabe des Nachnamens aufruft
43   private JLabel getJLabel1() {
44     if (jLabel1 == null) {
45       jLabel1 = new JLabel();
46       jLabel1.setText("  Buchungsbestätigung:");
47     }
48     return jLabel1;
49   }
50
51   //  Erzeugt das Textfeld zur Eingabe des Buchungsdatums
52   private JTextField getjTFieldBID() {
53     if (jTFieldBID == null) {
54       jTFieldBID = new JTextField(10);
55     }
56     return jTFieldBID;
57   }
58
59   public void callService(String KundeID,
60                           String HotelID,
61                           String BDatum,
62                           String Zimmertyp,
63                           String Datum_von,
64                           String Datum_bis){
65
66       try{
67             HotelbuchungService MyService = new HotelbuchungServiceLocator();
68           Hotelbuchung MyPort = MyService.getHotelbuchung();
69           BIDconf = MyPort.bookConfirm(KundeID,
70                                        HotelID,
71                                        BDatum,
72                                        Zimmertyp,
```

```
                                        Datum_von,
                                        Datum_bis);
          //System.out.println(test);
    } catch(Exception e1) {
        e1.printStackTrace();
    }
  }
  //  Erzeugt den Button zum Einlesen der Daten
  private JButton getJButton() {
    if (jButton == null) {
      jButton = new JButton();
      jButton.setText("Anfordern");
      jButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent arg0) {
                getjTFieldBID().setText("" + BIDconf);
            }
      });}
    return jButton;
  }

  //  Erzeugt den Beenden-Button
  private JButton getJButton1() {
    if (jButton1 == null) {
      jButton1 = new JButton();
      jButton1.setText("Beenden");
      jButton1.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
          dispose();
        }
      });
    }
    return jButton1;
  }
}
```

Quelltext 21.7: Hotelbuchung: WebService Aufruf in der BookConfirm.java GUI

```
package _100._183._37._195.axis.Hotelbuchung_jws;

import javax.swing.JFrame;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class BookInsert extends JFrame {
  private BookConfirm bc = null;
    private JPanel jContentPane = null;

  private JTextField jTFieldKundeID = null;
  private JTextField jTFieldHotelID = null;
  private JTextField jTFieldBDatum = null;
  private JTextField jTFieldZimmertyp = null;
  private JTextField jTFieldDatum_von = null;
  private JTextField jTFieldDatum_bis = null;

  private JLabel jLabel1 = null;
  private JLabel jLabel2 = null;
  private JLabel jLabel3 = null;
  private JLabel jLabel4 = null;
  private JLabel jLabel5 = null;
  private JLabel jLabel6 = null;

  private JButton jButton = null;
  private JButton jButton1 = null;
  private int BuchungID;

```

```
30  public BookInsert(){
31      super();
32      init();
33      bc = new BookConfirm();
34  }
35
36  //  Erzeugt das Fenster
37  public void init() {
38    this.setSize(500,300);
39    this.setTitle("Eingabe der Buchungsdaten");
40    this.setContentPane(getJContentPane());
41    this.setResizable(false);
42    this.setVisible(true);
43  }
44
45  //  Erzeugt die ContentPane
46  private JPanel getJContentPane() {
47    if(jContentPane == null) {
48      jContentPane = new JPanel();
49      jContentPane.add(getJLabel1(), null);
50      jContentPane.add(getjTFieldBDatum(), null);
51      jContentPane.add(getJLabel2(), null);
52      jContentPane.add(getjTFieldKundeID(), null);
53      jContentPane.add(getJLabel3(), null);
54      jContentPane.add(getjTFieldHotelID(), null);
55      jContentPane.add(getJLabel4(), null);
56      jContentPane.add(getjTFieldZimmertyp(), null);
57      jContentPane.add(getJLabel5(), null);
58      jContentPane.add(getjTFieldDatum_von(), null);
59      jContentPane.add(getJLabel6(), null);
60      jContentPane.add(getjTFieldDatum_bis(), null);
61      jContentPane.add(getJButton(), null);
62      jContentPane.add(getJButton1(), null);
63      jContentPane.setLayout(new GridLayout(7,2));
64    }
65    return jContentPane;
66  }
67
68  //  Erzeugt das Label das zur Eingabe des Nachnamens aufruft
69  private JLabel getJLabel1() {
70    if (jLabel1 == null) {
71      jLabel1 = new JLabel();
72      jLabel1.setText("  Buchungsdatum (JJJJ-MM-TT):");
73    }
74    return jLabel1;
75  }
76
77    //  Erzeugt das Label das zur Eingabe des Vornamens aufruft
78  private JLabel getJLabel2() {
79    if (jLabel2 == null) {
80      jLabel2 = new JLabel();
81      jLabel2.setText("  Kunden-Nr.:");
82    }
83    return jLabel2;
84  }
85
86    //  Erzeugt das Label das zur Eingabe des Geburtsdatums aufruft
87  private JLabel getJLabel3() {
88    if (jLabel3 == null) {
89      jLabel3 = new JLabel();
90      jLabel3.setText("  Hotel-Nr.:");
91    }
92    return jLabel3;
93  }
94
95  //  Erzeugt das Label das zur Eingabe der Strasse aufruft
```

```
96   private JLabel getJLabel4() {
97     if (jLabel4 == null) {
98       jLabel4 = new JLabel();
99       jLabel4.setText("  Zimmertyp (Einzelzimmer/Doppelzimmer):");
100    }
101    return jLabel4;
102  }
103
104  //  Erzeugt das Label das zur Eingabe der PLZ aufruft
105  private JLabel getJLabel5() {
106    if (jLabel5 == null) {
107      jLabel5 = new JLabel();
108      jLabel5.setText("  Datum von (JJJJ-MM-TT):");
109    }
110    return jLabel5;
111  }
112
113  //  Erzeugt das Label das zur Eingabe des Orts aufruft
114  private JLabel getJLabel6() {
115    if (jLabel6 == null) {
116      jLabel6 = new JLabel();
117      jLabel6.setText("  Datum bis (JJJJ-MM-TT):");
118    }
119    return jLabel6;
120  }
121
122  //  Erzeugt das Textfeld zur Eingabe des Buchungsdatums
123  private JTextField getjTFieldBDatum() {
124    if (jTFieldBDatum == null) {
125      jTFieldBDatum = new JTextField(10);
126    }
127    return jTFieldBDatum;
128  }
129
130  //  Erzeugt das Textfeld zur Eingabe der Kundennummer
131  private JTextField getjTFieldKundeID() {
132    if (jTFieldKundeID == null) {
133      jTFieldKundeID = new JTextField(10);
134    }
135    return jTFieldKundeID;
136  }
137
138  //  Erzeugt das Textfeld zur Eingabe der Hotelnummer
139  private JTextField getjTFieldHotelID() {
140    if (jTFieldHotelID == null) {
141      jTFieldHotelID = new JTextField(10);
142    }
143    return jTFieldHotelID;
144  }
145
146  //  Erzeugt das Textfeld zur Eingabe des Zimmertyps
147  private JTextField getjTFieldZimmertyp() {
148    if (jTFieldZimmertyp == null) {
149      jTFieldZimmertyp = new JTextField(40);
150    }
151    return jTFieldZimmertyp;
152  }
153
154  //  Erzeugt das Textfeld zur Eingabe des Zeitraums "Buchung von"
155  private JTextField getjTFieldDatum_von() {
156    if (jTFieldDatum_von == null) {
157      jTFieldDatum_von = new JTextField(10);
158    }
159    return jTFieldDatum_von;
160  }
161
```

```
162   //  Erzeugt das Textfeld zur Eingabe des Zeitraums "Buchung bis"
163   private JTextField getjTFieldDatum_bis() {
164     if (jTFieldDatum_bis == null) {
165       jTFieldDatum_bis = new JTextField(10);
166     }
167     return jTFieldDatum_bis;
168   }
169
170   //  Erzeugt den Button zum Einlesen der Daten
171   private JButton getJButton() {
172     if (jButton == null) {
173       jButton = new JButton();
174       jButton.setText("Hotel buchen");
175       jButton.addActionListener(new ActionListener(){
176             public void actionPerformed(ActionEvent arg0) {
177               if (bc != null){
178                   bc.callService(jTFieldKundeID.getText(),
179                                  jTFieldHotelID.getText(),
180                                  jTFieldBDatum.getText(),
181                                  jTFieldZimmertyp.getText(),
182                                  jTFieldDatum_von.getText(),
183                                  jTFieldDatum_bis.getText());
184               }
185             }
186       });
187     }
188     return jButton;
189   }
190
191   //  Erzeugt den Beenden-Button
192   private JButton getJButton1() {
193     if (jButton1 == null) {
194       jButton1 = new JButton();
195       jButton1.setText("Beenden");
196       jButton1.addActionListener(new ActionListener(){
197         public void actionPerformed(ActionEvent e){
198           dispose();
199         }
200       });
201     }
202     return jButton1;
203   }
204
205   public static void main(String[] args){
206       new BookInsert();
207   }
208 }
```

Quelltext 21.8: Hotelbuchung: Start GUI BookInsert.java

21.6. Benötigte Bibliotheken

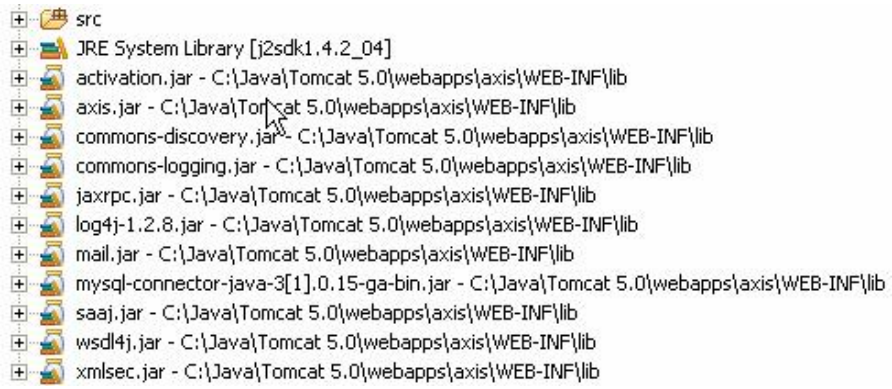


Abbildung 21.5.: Hotelbuchung: Benötigte Bibliotheken

21.7. Beispielsitzung

Beim Ausführen der BookInsert.java erscheint ein Fenster zur Eingabe der Buchungsdaten. Es beinhaltet die Felder: „Buchungsdatum“, „Kunden-Nr.“, „Hotel-Nr.“, „Zimmertyp“, „Datum von“ und „Datum bis“. Die „Kunden-Nr.“ und die „Hotel-Nr.“ werden hierbei dem Kunden als bekannt festgelegt. Theoretisch hat der Kunde eine Zeitschrift wo er die Hotel-Nr. nachschlagen kann. Mit dem Bestätigen des Buttons „Hotel buchen“ werden die Daten in die MySQL Datenbank eingefügt

Abbildung 21.6.: Hotelbuchung: BuchungsGUI - Eingabe der Buchungsdaten

	BuchungID	KundeID	HotelID	BDatum	Zimmertyp	Datum_von	Datum_bis
1	1	1	1	2004-10-12	Doppelzimmer	2005-06-15	2005-06-30
2	2	2	2	2004-10-15	Einzelzimmer	2005-07-15	2005-07-30
3	3	3	3	2004-10-01	Doppelzimmer	2005-08-03	2005-08-18
4	4	4	4	2004-09-19	Doppelzimmer	2005-09-20	2005-10-05
5	5	5	5	2004-10-22	Doppelzimmer	2005-06-18	2005-07-02
6	6	6	6	2004-09-06	Einzelzimmer	2005-06-28	2005-07-20
7	7	7	7	2004-10-04	Einzelzimmer	2005-08-04	2005-08-22
8	8	8	8	2004-10-12	Einzelzimmer	2005-02-24	2005-03-10
9	9	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25
10	10	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25
11	11	1	5	2004-11-01	Doppelzimmer	2004-12-23	2004-12-25
12	23	3	5	2004-10-15	Doppelzimmer	2004-11-15	2004-11-21
13	24	3	6	2004-11-04	Doppelzimmer	2004-11-05	2004-11-10

Abbildung 21.7.: Hotelbuchung: MySQL Control Center - eingefügte Buchungsdaten

Zur Kontrolle wird das MySQL Control Center nochmals aufgerufen, ob die Daten auch eingefügt wurden.

Abbildung 21.8.: Hotelbuchung: BuchungsGUI - Buchungsbestätigung

Im Buchungsbestätigungs Fenster wird bei Bestätigen des Buttons „anfordern“ die BuchungsID „24“ zurückgegeben. Dieses bestätigt genau den Eintrag der zuvor in die Datenbank durch die Buchungsprozedur eingefügt wurde.

21.8. Veröffentlichung des WebService im UDDI-Verzeichnis

Der WebService soll für die Öffentlichkeit auch zugänglich gemacht werden. Hierbei wurde er im Microsoft UDDI Verzeichnis (uddi.microsoft.com) publiziert. Der Service „Hotelbuchung“ wird gefunden, wenn man im Suchmodul „BizWeb“ eingibt. Er besitzt folgenden Service Key: 366669d3-a28b-4dfd-b576-be338b12a680

Abbildung 21.9.: Hotelbuchung: uddi.microsoft.com search

21.9. Direkter Aufruf des WebService über den Browser

Der WebService kann über den Browser getestet werden, indem die Adresse wie folgt umgewandelt wird:

http://195.37.183.100/axis/Hotelbuchung.jws?method=BookConfirm&BDatum=2004-11-11&kundeID=1&hotelID=2&zimmerTyp=Doppelzimmer&datumVon=2004-11-13&datumBis=2004-11-15

Das Ergebnis wird als SOAP Nachricht im Browser zurückgeliefert Hierbei ist dieses die BuchungsID „13“.

```
<soapenv:Envelope>
  <soapenv:Body>
    <BookConfirmResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <BookConfirmReturn xsi:type="xsd:int">13</BookConfirmReturn>
    </BookConfirmResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 21.9: Hotelbuchung: SOAP Ergebnis

21.10. Resümee

Die eigene Entwicklung eines WebService und sich mit dem Themen UDDI, SOAP, WSDL, XML, MYSQL und viele weitere Bereiche auseinanderzusetzen war eine sehr gute Erfahrung. Man hat intensiv mit den Schwierigkeiten die dabei auftraten gelernt und das Thema WebService an sich besser verstehen können. Hierbei wurde eine gute Basis für unser laufendes Projekt BizWeb geschaffen, da jeder sich mit dem Thema auseinander setzen musste.

22. Textanalysierer

22.1. Ziel

Es ist ein WebService zu entwickeln und auf dem BizWeb–Server zu implementieren. Der WebService soll als Funktionalität einen String übergeben bekommen, der anhand verschiedener Kriterien ausgewertet werden soll. Es sollen Sätze, Wörter, Trennzeichen und Zeichen eines übergebenen Strings ermittelt und zurückgegeben werden. Der WebService soll also einen einfachen Textanalysierer repräsentieren, auf den über SOAP zugegriffen werden kann.

Genutzt werden soll dazu das AXIS–Framework, welches in der vorangegangenen Schulung schon ausführlich erläutert wurde.

Ziel dieser individuellen Aufgabe für die Projektteilnehmer ist es, dass die Funktionalitäten und das Handhaben von WebServices und deren in Verbindung stehenden Tools vermittelt werden. Aus diesem Grund soll sich mit dieser Aufgabe individuell auseinandergesetzt werden, indem sich ausgiebig mit den WebServices und dessen Technologien beschäftigt wird.

22.2. Beschreibung

Der TextAnalysierer enthält insgesamt vier verschiedene Funktionen:

- `saetze_zaehlen()`
  Nach Übergabe eines Strings wird die Anzahl der enthaltenen Sätze berechnet und zurückgegeben. Dies geschieht anhand des regulären Ausdrucks „`[\.\!\?]`“. Dieser trennt den String, wenn eines der Zeichen (. oder ! oder ?) im String vorkommt und füllt ein Array mit den entstehenden Teilstücken des Strings. Schließlich wird die Länge des Array zurückgegeben und an den Client übermittelt.

Implementierung:

```
String[] saetze;
saetze = text.split(''[\\.\\!\\?]'');
return saetze.length;
```

- `worte_zaehlen()`
  Nach Übergabe eines Strings wird die Anzahl der enthaltenen Worte berechnet und zurückgegeben. Dies geschieht anhand des regulären Ausdrucks „`[\s]`“ (Terminal–Symbol für das Leerzeichen). Dieser trennt den String, wenn ein Leerzeichen im String vorkommt und füllt ein Array mit den entstehenden Teilstücken des Strings. Schließlich wird die Länge des Array zurückgegeben und an den Client übermittelt.

  Implementierung:

```
String[] worte;
worte = text.split("[\\s]");
return worte.length;
```

- `trennzeichen_zaehlen()`
  Nach Übergabe eines Strings wird dieser zeichenweise von Anfang bis Ende daraufhin überprüft, ob das aktuelle Zeichen ein Trennzeichen ist. Ist dies der Fall, wird ein Zähler um Eins erhöht. Wenn der String abgearbeitet ist, wird die im Zähler gespeicherte Zahl an den Client zurückgegeben.

  Implementierung:

```
int counter = 0;
for(int i = 0; i < text.length(); i++) {
  char c = text.charAt(i);
  if( c == '.' || c == '!' || c == '?') {
    counter++;
  }
}
return counter;
```

- `zeichen_zaehlen()`
  Nach Übergabe eines Strings wird die Länge desselben an den Client zurückgegeben.

  Implementierung:

```
return text.length();
```

Der vollständige Quelltext befindet sich im Anhang.

Um den WebService nun lauffähig zu machen, muss dieser in eine AXIS–Umgebung eingebunden werden. Dazu muss die `.java`–Klasse in `.jws` umbenannt werden und in den AXIS–Ordner kopiert werden. Wenn der Tomcat gestartet wurde, ist der WebService nun nutzbar. Erreichbar ist dieser unter http://localhost:8080/axis/TextAnalyser.jws.

Da ebenfalls ein Client für den Service entwickelt werden soll, wird die AXIS–interne Funktion für die Generierung einer WSDL–Beschreibung genutzt. Dazu wird im Browser http://localhost:8080/axis/TextAnalyser.jws?wsdl aufgerufen, welches eine WSDL–Beschreibung für den TextAnalysierer liefert.

Aus dieser WSDL lässt sich nun ein auf diesen WebService zugeschnittenen Client erstellen:

22.3. Java-Programm (Client)

Es wird sich auf bei der Client–Erstellung den AXIS–Funktionen bedient, die dieses annähernd komplett per Knopfdruck ermöglichen.

Dazu müssen die AXIS–Programmbibliotheken in den Build–Path der zukünftigen Client–Anwendung geladen werden:

Abbildung 22.1.: TextAnalyser: AXIS–JAR's im Build–Path

Zur Erstellung eines Clients wird nun eine aktuelle WSDL–Datei des WebServices benötigt. Diese kann auf dem BizWeb–Server unter AXIS mit folgender URL aufgerufen werden: http://195.37.183.100/axis/TextAnalyser.jws?wsdl

Der Inhalt dieser Datei gibt Aufschluss über die Implementierung des WebService und über seine zur Verfügung gestellten Methoden. Die WSDL–Datei für den TextAnalyser sieht nun wie folgt aus:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions targetNamespace="http://195.37.183.100/axis/TextAnalyser.jws"
3     xmlns="http://schemas.xmlsoap.org/wsdl/"
4     xmlns:apachesoap="http://xml.apache.org/xml-soap"
5     xmlns:impl="http://195.37.183.100/axis/TextAnalyser.jws"
6     xmlns:intf="http://195.37.183.100/axis/TextAnalyser.jws"
7     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
8     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9     xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
10    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11
12   <wsdl:message name="zeichen_zaehlenRequest">
13     <wsdl:part name="text" type="xsd:string"/>
14   </wsdl:message>
15
16   <wsdl:message name="trennzeichen_zaehlenRequest">
17     <wsdl:part name="text" type="xsd:string"/>
18   </wsdl:message>
19
```

```
20    <wsdl:message name="worte_zaehlenRequest">
21      <wsdl:part name="text" type="xsd:string"/>
22    </wsdl:message>
23
24    <wsdl:message name="trennzeichen_zaehlenResponse">
25      <wsdl:part name="trennzeichen_zaehlenReturn" type="xsd:int"/>
26    </wsdl:message>
27
28    <wsdl:message name="saetze_zaehlenResponse">
29      <wsdl:part name="saetze_zaehlenReturn" type="xsd:int"/>
30    </wsdl:message>
31
32    <wsdl:message name="worte_zaehlenResponse">
33      <wsdl:part name="worte_zaehlenReturn" type="xsd:int"/>
34    </wsdl:message>
35
36    <wsdl:message name="zeichen_zaehlenResponse">
37      <wsdl:part name="zeichen_zaehlenReturn" type="xsd:int"/>
38    </wsdl:message>
39
40    <wsdl:message name="saetze_zaehlenRequest">
41      <wsdl:part name="text" type="xsd:string"/>
42    </wsdl:message>
43
44    <wsdl:portType name="TextAnalyser">
45
46      <wsdl:operation name="saetze_zaehlen" parameterOrder="text">
47        <wsdl:input message="impl:saetze_zaehlenRequest"
48            name="saetze_zaehlenRequest"/>
49        <wsdl:output message="impl:saetze_zaehlenResponse"
50            name="saetze_zaehlenResponse"/>
51      </wsdl:operation>
52
53      <wsdl:operation name="trennzeichen_zaehlen" parameterOrder="text">
54        <wsdl:input message="impl:trennzeichen_zaehlenRequest"
55            name="trennzeichen_zaehlenRequest"/>
56        <wsdl:output message="impl:trennzeichen_zaehlenResponse"
57            name="trennzeichen_zaehlenResponse"/>
58      </wsdl:operation>
59
60      <wsdl:operation name="worte_zaehlen" parameterOrder="text">
61        <wsdl:input message="impl:worte_zaehlenRequest"
62            name="worte_zaehlenRequest"/>
63        <wsdl:output message="impl:worte_zaehlenResponse"
64            name="worte_zaehlenResponse"/>
65      </wsdl:operation>
66
67      <wsdl:operation name="zeichen_zaehlen" parameterOrder="text">
68        <wsdl:input message="impl:zeichen_zaehlenRequest"
69            name="zeichen_zaehlenRequest"/>
70        <wsdl:output message="impl:zeichen_zaehlenResponse"
71            name="zeichen_zaehlenResponse"/>
72      </wsdl:operation>
73
74    </wsdl:portType>
75
76    <wsdl:binding name="TextAnalyserSoapBinding" type="impl:TextAnalyser">
77      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
78
79      <wsdl:operation name="saetze_zaehlen">
80        <wsdlsoap:operation soapAction=""/>
81        <wsdl:input name="saetze_zaehlenRequest">
82          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
83        namespace="http://DefaultNamespace" use="encoded"/>
84        </wsdl:input>
85        <wsdl:output name="saetze_zaehlenResponse">
```

```
86          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
87        namespace="http://195.37.183.100/axis/TextAnalyser.jws" use="encoded"/>
88        </wsdl:output>
89      </wsdl:operation>
90
91      <wsdl:operation name="trennzeichen_zaehlen">
92        <wsdlsoap:operation soapAction=""/>
93        <wsdl:input name="trennzeichen_zaehlenRequest">
94          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
95        namespace="http://DefaultNamespace" use="encoded"/>
96        </wsdl:input>
97        <wsdl:output name="trennzeichen_zaehlenResponse">
98          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
99        namespace="http://195.37.183.100/axis/TextAnalyser.jws" use="encoded"/>
100       </wsdl:output>
101     </wsdl:operation>
102
103     <wsdl:operation name="worte_zaehlen">
104       <wsdlsoap:operation soapAction=""/>
105       <wsdl:input name="worte_zaehlenRequest">
106         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
107       namespace="http://DefaultNamespace" use="encoded"/>
108       </wsdl:input>
109       <wsdl:output name="worte_zaehlenResponse">
110         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
111       namespace="http://195.37.183.100/axis/TextAnalyser.jws" use="encoded"/>
112       </wsdl:output>
113     </wsdl:operation>
114
115     <wsdl:operation name="zeichen_zaehlen">
116       <wsdlsoap:operation soapAction=""/>
117       <wsdl:input name="zeichen_zaehlenRequest">
118         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
119       namespace="http://DefaultNamespace" use="encoded"/>
120       </wsdl:input>
121       <wsdl:output name="zeichen_zaehlenResponse">
122         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
123       namespace="http://195.37.183.100/axis/TextAnalyser.jws" use="encoded"/>
124       </wsdl:output>
125     </wsdl:operation>
126
127   </wsdl:binding>
128
129   <wsdl:service name="TextAnalyserService">
130     <wsdl:port binding="impl:TextAnalyserSoapBinding" name="TextAnalyser">
131       <wsdlsoap:address location="http://195.37.183.100/axis/TextAnalyser.jws"/>
132     </wsdl:port>
133   </wsdl:service>
134
135 </wsdl:definitions>
```

Quelltext 22.1: TextAnalyser: TextAnalyser.wsdl

Mit dem PlugIn „WSDL2Java“ kann in Eclipse direkt ein neues Package erstellt werden, welches den Zugriff auf den WebService regelt. Dazu werden aus der soeben dargestellten WSDL–Datei folgende Klassen generiert:

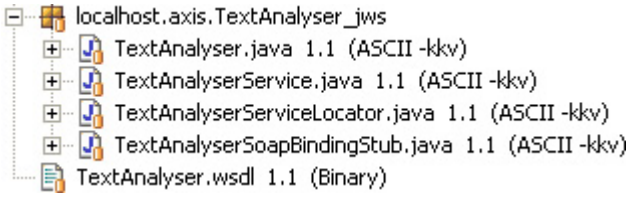


Abbildung 22.2.: TextAnalyser: Generiertes Package

Ein einfacher Client ohne Benutzeroberfläche der die generierten Klassen nutzt und somit auf den WebService zugreift sieht folgendermaßen aus:

```
1  /*
2   * Created on 17.08.2004
3   */
4  /**
5   * @author Bastian Onken
6   */
7  //import localhost.axis.TextAnalyser_jws.*;
8  import _100._183._37._195.axis.TextAnalyser_jws.*;
9
10 public class TextAnalyserClient {
11
12   public static void main(String[] args) throws Exception {
13     String text =
14        "Franz fuhr im total verwahrlosten Taxi quer durch Bayern. " +
15        "Wieso eigentlich Bayern? " +
16        "Da gibts doch nur verrückte, weisswurst-essende Schuhplattler, oder?!";
17     TextAnalyserService myService = new TextAnalyserServiceLocator();
18     TextAnalyser myPort = myService.getTextAnalyser();
19
20     System.out.println(myPort.saetze_zaehlen(text)+" Sätze");
21     System.out.println(myPort.worte_zaehlen(text)+" Worte");
22     System.out.println(myPort.trennzeichen_zaehlen(text)+" Trennzeichen");
23     System.out.println(myPort.zeichen_zaehlen(text)+" Zeichen");
24   }
25 }
```

Quelltext 22.2: TextAnalyser: TextAnalyserClient.java

Wird dieser Client ausgeführt, wird folgender Output in der Java–Konsole erzeugt:

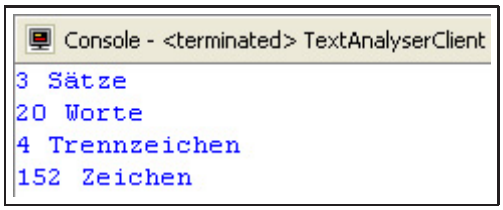


Abbildung 22.3.: TextAnalyser: Ausgabe in der Konsole

Da nun noch kein intuitives Handling mit dem WebService möglich ist, muss noch ein GUI entworfen werden. Dieses wird im folgenden Abschnitt dargestellt, der zugehörigen Quelltext befindet sich im Anhang.

22.4. Beispielsitzung

Für die GUI–Programmierung wurde das JAVA–eigene Swing–Framework genutzt. Das GUI besteht aus einem Fenster in dem sich ein Eingabe– und ein Ausgabefeld befindet, sowie vier weitere Buttons. Diese Anordnung reicht aus, um den WebService mit seinen Funktionalitäten zu bedienen. Um den Client zu starten, muss die Klasse `TextAnalyserClientGUI.java` ausgeführt werden.
Beim Aufruf des Clients öffnet sich zu Beginn folgendes Fenster:

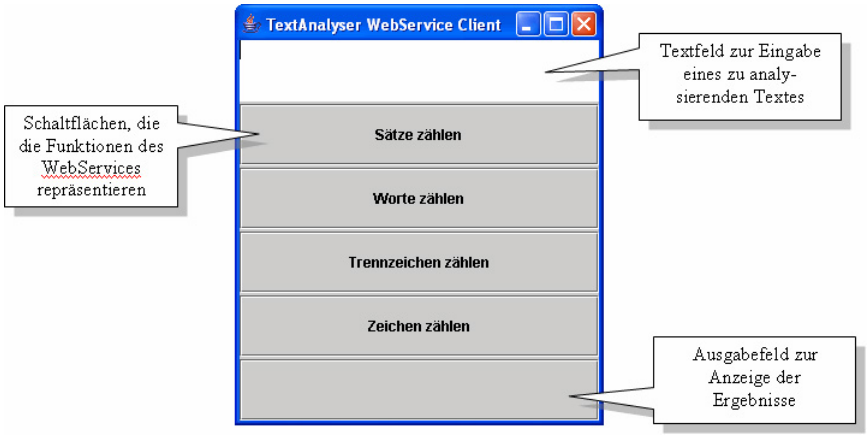


Abbildung 22.4.: TextAnalyser: Startfenster des Clients

Hier kann nun ein beliebiger Text in das obere Textfeld eingegeben werden. Am besten eignen sich Texte mit mehreren Sätzen und verschiedenen Zeichen, um die volle Funktionalität des WebService auszunutzen.
Nachdem nun ein Text eingegeben wurde, ist es möglich eine der Schaltflächen anzuklicken, die jeweils eine Funktion des WebService ausführen. Im Folgenden sieht man Screenshots von allen Funktionen (bei der gerade genutzten Funktion ist der Button–Text durch eine Umrandung markiert):

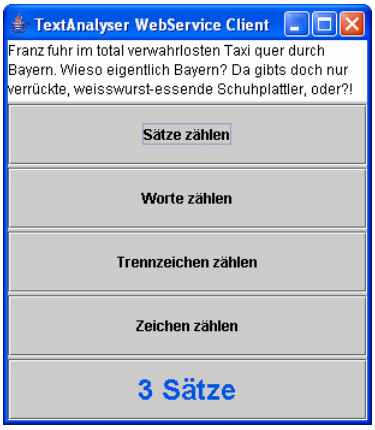


Abbildung 22.5.: TextAnalyser: Sätze zählen

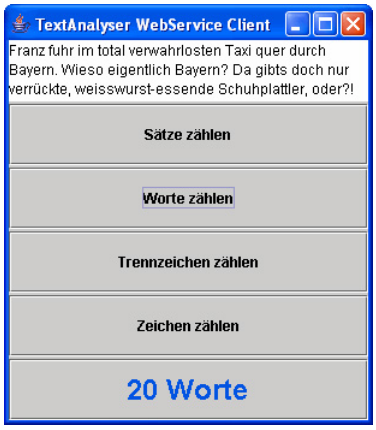


Abbildung 22.6.: TextAnalyser: Worte zählen

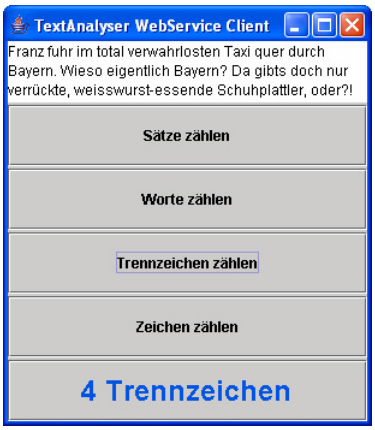


Abbildung 22.7.: TextAnalyser: Trennzeichen zählen

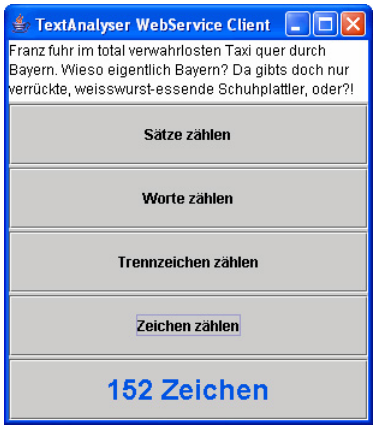


Abbildung 22.8.: TextAnalyser: Zeichen zählen

Der WebService ist aber auch ohne den hier entwickelten Client direkt über einen Browser aufrufbar. Zurückgeliefert wird dazu eine SOAP–Nachricht, die dann weiter verarbeitet werden kann. Der Textanalyser kann unter der folgenden URL manuell mit Übergabe von Parametern angesprochen werden:

http://195.37.183.100/axis/TextAnalyser.jws?method=saetze_zaehlen
&value=Dies ist ein Satz. Dies hier ist noch einer!

Der Methodenname des WebService wird mit „`method=...`“ festgelegt, die übergebenen Daten mit „`value=...`“.

Die Antwort auf den Request per Browser ist wie folgt aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <saetze_zaehlenResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <saetze_zaehlenReturn xsi:type="xsd:int">2</saetze_zaehlenReturn>
    </saetze_zaehlenResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 22.3: TextAnalyser: SOAP–Antwort

Der Vorteil bei dem Aufruf direkt über den Browser liegt klar auf der Hand: So kann direkt auf die Ergebnisse des WebService (auch für die Kontrolle der Ergebnisse) zugegriffen werden. Über die Entwicklung eines Client kann später nachgedacht werden.

22.5. Informationsfluss zwischen Client und Server

Zur Veranschaulichung ist hier nun der Informationsfluss zwischen Nutzer und WebService dargestellt, wie er von Client zu Server (Request) und von Server zu Client (Response) abläuft:

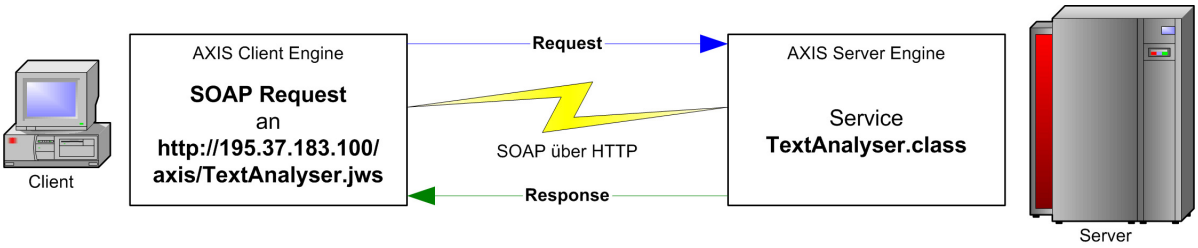


Abbildung 22.9.: TextAnalyser: Informationsfluss

Der Request wird nun folgendermaßen vom Client formuliert und an den WebService–Provider gesendet (hier am Beispiel der „`saetze_zaehlen`“–Funktion:

```
POST /axis/TextAnalyser.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 608

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:saetze_zaehlen
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://DefaultNamespace">
      <text xsi:type="xsd:string">Franz fuhr im total verwahrlosten Taxi quer durch
         Bayern. Wieso eigentlich Bayern? Da gibts doch nur verrückte, weisswurst-
         essende Schuhplattler, oder?!</text>
    </ns1:saetze_zaehlen>
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 22.4: TextAnalyser: SOAP–Request

Der Empfänger verarbeitet die Anfrage und sendet nach Berechnung der Ergebnisse folgende Antwort auf den Request zurück an den Client:

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0663D6A6111EAF23F5C1FF8B2B644EA3; Path=/axis
Content-Type: text/xml;charset=utf-8
Date: Wed, 03 Nov 2004 10:59:00 GMT
Server: Apache-Coyote/1.1
Connection: close

```

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:saetze_zaehlenResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="http://DefaultNamespace">
      <ns1:saetze_zaehlenReturn xsi:type="xsd:int">3</ns1:saetze_zaehlenReturn>
    </ns1:saetze_zaehlenResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 22.5: TextAnalyser: SOAP–Response

22.6. Veröffentlichung im UDDI–Verzeichnis

Der fertige WebService ist nun im Internet verfügbar. Damit dieser WebService nun auch von der Öffentlichkeit gefunden werden kann, bietet es sich an, diesen in einem UDDI–Verzeichnis zu hinterlegen. Hier werden WebServices verschiedenster Kategorien gesammelt, nach denen gesucht werden kann.

Da der UDDI–Dienst von IBM zeitweise nicht verfügbar war, habe ich mich für die Registrierung in Microsofts UDDI–Verzeichnis entschieden. Nun ist der TextAnalyser dort mit einer eindeutigen ID hinterlegt: `63897de1-7195-4bcb-abb8-c1defb277422`

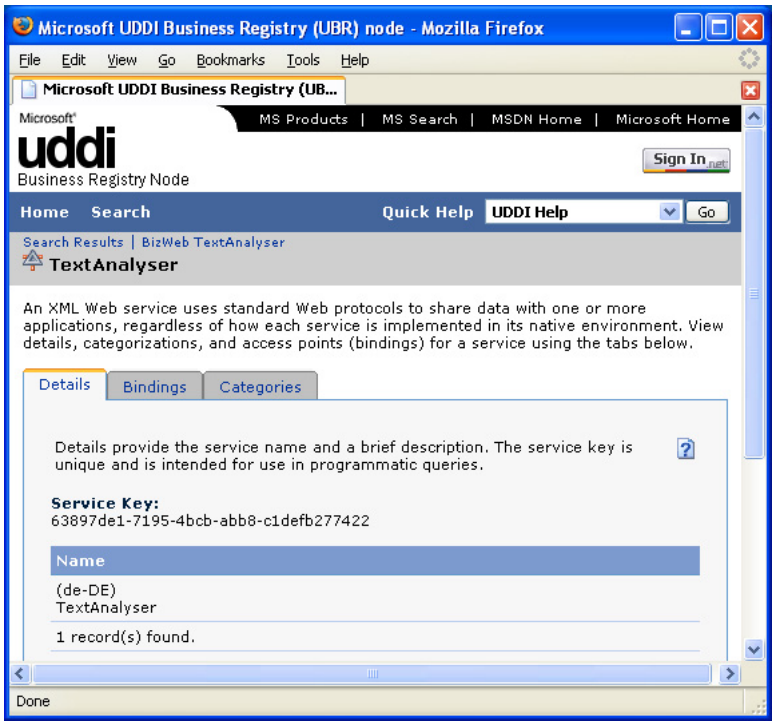


Abbildung 22.10.: TextAnalyser: UDDI MS (http://uddi.microsoft.com)

22.7. Resümee

Durch die individuelle Erstellung von WebServices für die Projektteilnehmer sind einige Unklarheiten die WebService–Technologie betreffend beseitigt worden. Da sich nun mehr mit der Thematik auseinandergesetzt wurde, wurde das Umgehen mit einem WebService verinnerlicht — Das Gesamtverständnis ist enorm gestiegen. Die Aufgabe war dementsprechend nicht trivial und erforderte intensive Recherche in den Schulungsunterlagen und verschiedensten Dokumentationen. Eine der größeren Hilfen war dabei die AXIS–Dokumentation der Apache–Group.

Auf jeden Fall hat aber das selbständige Entwickeln eines WebService sehr viel Spaß gemacht und ich denke, dass ich das angeeignete Wissen weiter in das Projekt stecken kann. Ebenfalls denke ich, dass das Wissen zur Erstellung eines WebService wichtig für die Zukunft ist, da sich das Wissen in zukünftigen Arbeitsverhältnissen mit Sicherheit einbringen lässt und eine wichtige Zusatzqualifikation darstellt.

23. Buch-Datenbank

23.1. Ziel des Web-Service

Ziel des Web-Service ist es, eine Umgebung zu schaffen mit deren Hilfe eine Literatursuche aus einer selbst eingerichteten XML-Datenbank erfolgen kann. Als zu verwendende Datenbank ist Apache Xindice vorgegeben. Es sollen alle Technologien eines Web-Service angewandt und verstanden werden.

Der Web-Service wird so aussehen, dass man nach einem Autor suchen und dessen Buch mit ISBN, Titel, Ausgabejahr sowie weiteren Autoren in einem separaten Fenster als DOM-Baum angezeigt bekommt. Die Eingabe, des zu suchenden Autoren, sowie die Ausgabe erfolgt in einem Java Swing GUI.

23.2. Beschreibung des Web-Service

Der genannte Web-Service besteht zum Größtenteil aus der Java-Datei „literatursuche.java“. Diese sieht wie folgt aus:

```
1 import org.xmldb.api.base.*;
2 import org.xmldb.api.modules.*;
3 import org.xmldb.api.*;
```

Quelltext 23.1: Literatursuche: Web Service – Import

Die Import-Zeilen sind für die Xindice-Datenbank notwendig und ermöglichen unter anderem die Verbindungsherstellung.

```
1  public class literatursuche {
2      public static String XindiceAusg(String str) throws XMLDBException, Exception {
3          Collection col = null;
4          String erg = "";
5          try {
6              String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
7              Class c = Class.forName(driver);
8              Database database = (Database) c.newInstance();
9              DatabaseManager.registerDatabase(database);
10             col = DatabaseManager.getCollection("xmldb:xindice:///db/literatursuche");
```

Quelltext 23.2: Literatursuche: Web Service – Datenbankverbindung

Dieser erste Teil der Klasse „literatursuche“ stellt die Verbindung zur Datenbank her.

```
1            String xpath = "//literaturverzeichnis/buch[autor='"+str+"']";
2            XPathQueryService service =
3               (XPathQueryService) col.getService("XPathQueryService", "1.0");
4            ResourceSet resultSet = service.query(xpath);
5            ResourceIterator results = resultSet.getIterator();
6            while (results.hasMoreResources()) {
7               Resource res = results.nextResource();
8               erg = erg + (String)res.getContent();
9               //System.out.println((String) res.getContent());
10           }
11        }
```

Quelltext 23.3: Literatursuche: Web Service – XPath

Es wird ein XPath Pfad festgelegt, welcher durch einen String ergänzt wird, der vom Benutzer einzugeben ist. Dieser XPath-Pfad wird an einen XPathQueryService übergeben und das Ergebnis in einem ResultSet gespeichert. Dieser ResultSet wird in der Variable „erg“ gespeichert.

```
1
2         catch (XMLDBException e) {
3            System.err.println("XML:DB Exception occured " + e.errorCode);
4         }
5         finally {
6            if (col != null) {
7               col.close();
8            }
9         }
10        System.out.println(erg);
11        return erg;
12     }
13  }
```

Quelltext 23.4: Literatursuche: Web Service – Rückgabe

Die Variable „erg“ wird an das Programm das den Web-Service aufgerufen hat übergeben.

Bevor der Web-Service ausgeführt werden kann muss die XML-Datenbank Xindice eingerichtet werden. Hierfür habe ich mich eines GUI bedient mit dessen Hilfe ich XML-Dateien in die Datenbank einlesen und speichern kann. Man kann hier eine neue „Datenkollektion“ anlegen und diese dann mit Daten füllen, nachdem man die Xindice Datenbank gestartet und sich über das GUI mit dieser verbunden hat.

Die von mir in die Datenbank eingebundene XML-Datei wurde zuvor von mir erstellt und beinhaltet eine simple Auflistung mehrere Bücher mit ISBN, Autor, Titel und Erscheinungsjahr.

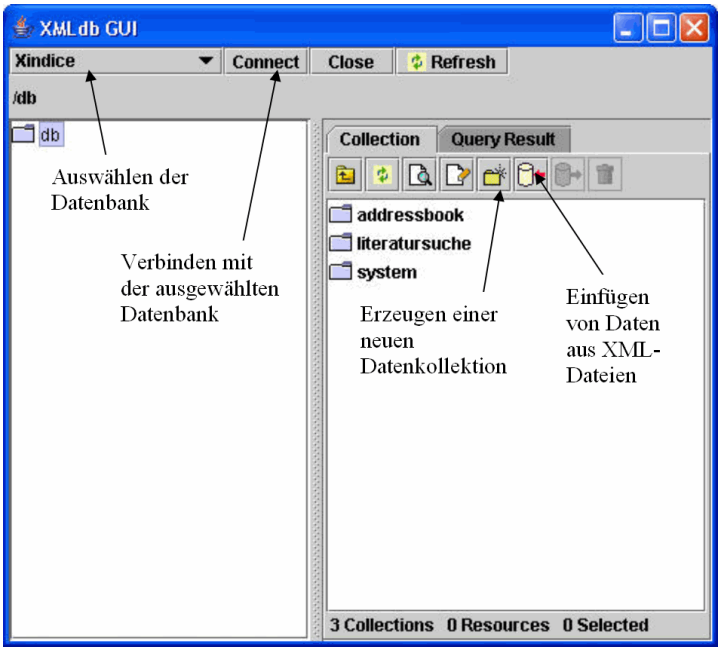


Abbildung 23.1.: Literatursuche: GUI für die Verwaltung der Datenbank Xindice

Sobald die Datenbank eingerichtet ist kommt die zuvor erwähnte Datei literatursuche.java zum tragen. Diese wird jetzt in das Verzeichnis `%AXIS_HOME%` kopiert und hier von *.java in *.jws umbenannt. Diese jws-Datei muss nun aufgerufen werden, z.B. mit Hilfe eines Browsers. Dieses geschieht in meinem Fall, nachdem der Tomcat gestartet wurde, über folgende URL:

http://localhost:8080/axis/literatursuche.jws

Dieselbe Anzeige erhält man auch, wenn man den Web-Service über den BizWeb Server mit folgender URL aufruft:

http://195.37.183.100/axis/literatursuche.jws

Der Web-Service ist außerdem über den Service Namen „Literatursuche“ oder folgenden Binding Key im Microsoft UDDI Verzeichnis zu finden:

```
315f7892-e764-4826-8e73-e8f1ba2a3e67
```

Man erhält folgende Anzeige in seinem Browser:

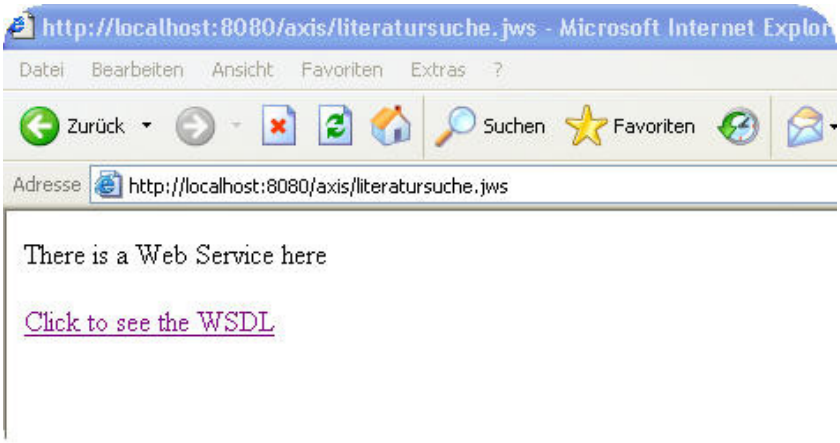


Abbildung 23.2.: Literatursuche: Aufrufen der *.jws über den Browser

Hier lässt man sich nun über den in Abbildung 2 zu sehenden Link den WSDL-Quellcode abzeigen. Dieser sieht für meine Datei wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/literatursuche.jws" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://localhost:8080/axis/literatursuche.jws"
    xmlns:intf="http://localhost:8080/axis/literatursuche.jws" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="http://base.api.xmldb.org"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <schema targetNamespace="http://base.api.xmldb.org" xmlns="http://www.w3.org/2001/XMLSchema">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      - <complexType name="XMLDBException">
        - <sequence>
            <element name="errorCode" type="xsd:int" />
            <element name="vendorErrorCode" type="xsd:int" />
          </sequence>
        </complexType>
      </schema>
    </wsdl:types>
  - <wsdl:message name="XMLDBException">
      <wsdl:part name="fault" type="tns1:XMLDBException" />
    </wsdl:message>
  - <wsdl:message name="XindiceAusgResponse">
      <wsdl:part name="XindiceAusgReturn" type="xsd:string" />
    </wsdl:message>
  - <wsdl:message name="XindiceAusgRequest">
      <wsdl:part name="str" type="xsd:string" />
    </wsdl:message>
  - <wsdl:portType name="literatursuche">
    - <wsdl:operation name="XindiceAusg" parameterOrder="str">
        <wsdl:input message="impl:XindiceAusgRequest" name="XindiceAusgRequest" />
        <wsdl:output message="impl:XindiceAusgResponse" name="XindiceAusgResponse" />
        <wsdl:fault message="impl:XMLDBException" name="XMLDBException" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="literatursucheSoapBinding" type="impl:literatursuche">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="XindiceAusg">
        <wsdlsoap:operation soapAction="" />
      - <wsdl:input name="XindiceAusgRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://DefaultNamespace" use="encoded" />
        </wsdl:input>
      - <wsdl:output name="XindiceAusgResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/axis/literatursuche.jws"
            use="encoded" />
        </wsdl:output>
      - <wsdl:fault name="XMLDBException">
          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/axis/literatursuche.jws"
            use="encoded" />
        </wsdl:fault>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="literatursucheService">
    - <wsdl:port binding="impl:literatursucheSoapBinding" name="literatursuche">
        <wsdlsoap:address location="http://localhost:8080/axis/literatursuche.jws" />
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Abbildung 23.3.: Literatursuche: WSDL-Quellcode der Datei literatursuche.jws

Diesen Quellcode hat man nun zu speichern. Es ist darauf zu achten das die Datei mit der Endung *.wsdl abgespeichert wird.

Diese *.wsdl-Datei ist nun mit Eclipse in Java-Code umzuwandeln. Dies geschieht über folgende Eclipse-Funktion:

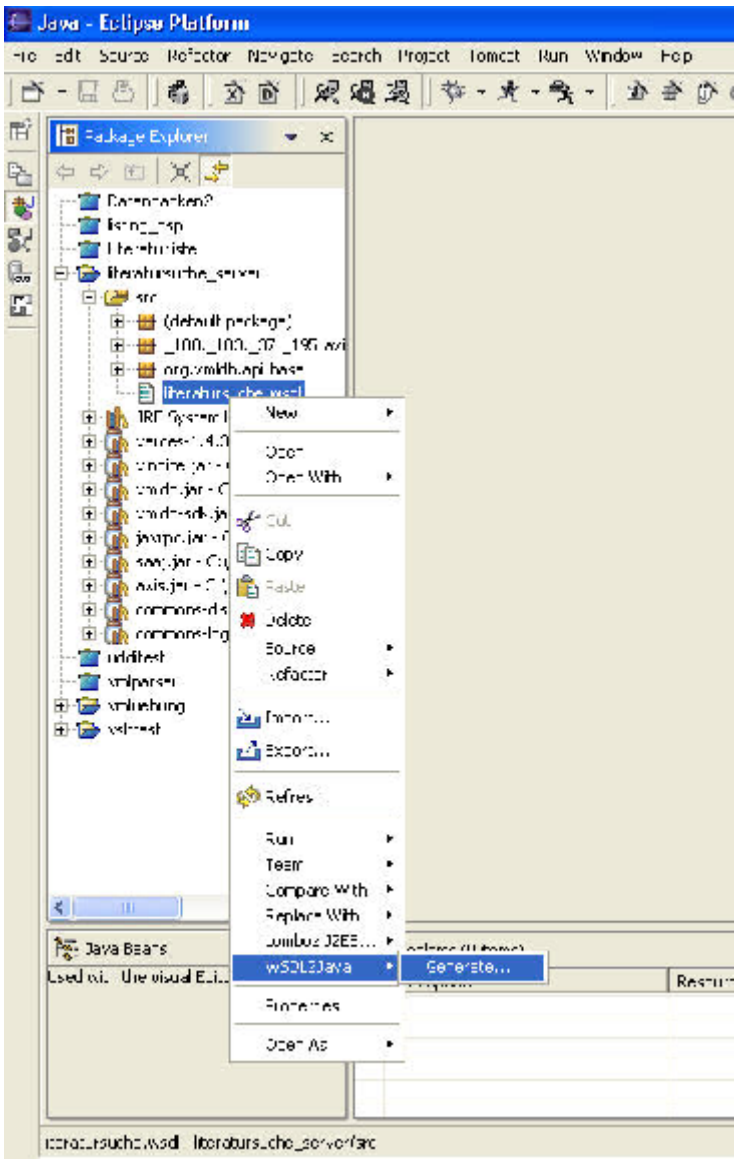


Abbildung 23.4.: Literatursuche: WSDL2Java-Funktion unter Eclipse

Der Web-Service kann nun bei laufender Xindice-Datenbank und Tomcat Server mit Hilfe des Java-Swing-GUI aufgerufen werden. Dieses wird im nächsten Abschnitt erläutert.

Auf die genaue Erklärung des Quellcodes der Ausgabe mittels DOM-Baum wird an dieser Stelle nicht genauer eingegangen. Er wird im Folgenden lediglich angehängt:

```
1  import org.apache.xerces.parsers.DOMParser;
2  import org.w3c.dom.Document;
3  import org.xml.sax.SAXException;
4  import java.io.IOException;
5
6  public class BasicDOM {
7
8     public BasicDOM (String xmlFile) {
9
10       DOMParser parser = new DOMParser();
11
12       try {
13          parser.parse(xmlFile);
14          Document document = parser.getDocument();
15          MyFrame MF = new MyFrame(document);
16       } catch (SAXException e) {
17          System.err.println (e);
18       } catch (IOException e) {
19          System.err.println (e);
20       }
21    }
22
23 }
```

Quelltext 23.5: Literatursuche: DOM-Baum – BasicDOM.java

```
1  import javax.swing.JFrame;
2  import javax.swing.JTree;
3  import org.w3c.dom.Document;
4  import org.w3c.dom.Node;
5
6  class MyFrame extends JFrame
7  {
8     MyFrame (Document doc)
9     {
10       Node docRoot = doc.getDocumentElement();
11
12       JTree tree = makeTree(docRoot);
13
14       getContentPane().add(tree);
15       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16       setSize(500,500);
17       show();
18    }
19
20    JTree makeTree(Node docRoot)
21    {
22       MyTreeNode treeRoot = new MyTreeNode();
23       new TreeExtender(treeRoot, docRoot);
24       return new JTree(treeRoot);
25    }
26 }
```

Quelltext 23.6: Literatursuche: DOM-Baum – MyFrame.java

```
1  import javax.swing.tree.DefaultMutableTreeNode;
2  class MyTreeNode extends DefaultMutableTreeNode {
3      /*Erbt alles, wie es ist. Bietet einen kürzeren Namen anstelle des unerträglich
4      lästigen Namens "DefaultMutableTreeNode."*/
5  }
```

Quelltext 23.7: Literatursuche: DOM-Baum – MyTreeNode.java

```
1  import org.w3c.dom.Node;
2  import org.w3c.dom.NamedNodeMap;
3
4  class TreeExtender {
5
6     MyTreeNode treeNode;
7     Node docNode;
8
9     TreeExtender (MyTreeNode treeNode, Node docNode) {
10       this.treeNode = treeNode;
11       this.docNode = docNode;
12
13       treeNode.setUserObject(getLabel());
14       addChildren();
15    }
16
17    void addChildren() {
18       Node docChild = docNode.getFirstChild();
19       while (docChild != null)
20       {
21          if (shouldDisplay(docChild))
22          {
23             MyTreeNode treeChild = new MyTreeNode();
24             treeNode.add(treeChild);
25             new TreeExtender(treeChild, docChild);
26          }
27          docChild = docChild.getNextSibling();
28       }
29    }
30
31    boolean shouldDisplay(Node docChild) {
32       if (docChild.getNodeType() == Node.TEXT_NODE)
33       {
34          String nodeValue = docChild.getNodeValue();
35          if (nodeValue != null &&
36              !nodeValue.trim().equals(""))
37             return true;
38          else
39             return false;
40       }
41       else
42          return true;
43    }
44
45    String getLabel() {
46       short nodeType = docNode.getNodeType();
47
48       if (nodeType == Node.TEXT_NODE)
49          return docNode.getNodeValue();
50       else if (nodeType == Node.ELEMENT_NODE)
51          return docNode.getNodeName() + stringOfAttribs();
52       else
53          return docNode.getNodeName();
54    }
55
56    String stringOfAttribs() {
57       NamedNodeMap attribs = docNode.getAttributes();
58       String label = "";
59
60       for (int i = 0; i < attribs.getLength(); i++)
61       {
62          Node item = attribs.item(i);
63          label += "    ";
64          label += item.getNodeName();
65          label += " = \"";
```

```
66              label += item.getNodeValue();
67              label += "\"";
68          }
69
70          return label;
71      }
72
73  }
```

Quelltext 23.8: Literatursuche: DOM-Baum – TreeExtender.java

23.3. Java-Programm zum Aufruf

Zum Aufruf des Web-Service wurde ein einfaches Java-Swing-GUI geschrieben. Dieses besteht aus einem JFrame mit einem (4,1) Grid Layout. Das Feld mit dem Hinweis „Bitte den Autoren eingeben:“ ist ein JLabel. Das Textfeld für die Eingabe des zu suchenden Autors ist ein JTextField. Die beiden restlichen Felder bestehen aus JButtons. Der Button „Buch ausgeben“ übergibt den eingegebenen String an den Web-Service und liefert das Ergebnis in Form einer XML-Datei an die Klasse „BasicDOM“. Diese wandelt die XML-Datei in einen DOM-Baum um und gibt diesen in einem separaten Fenster aus.

Abbildung 23.5.: Literatursuche: Java Swing GUI

Der Quellcode des Java-Swing-GUI sieht wie folgt aus:

```
1  //literaturgui.java
2
3  import javax.swing.JFrame;
4  import javax.swing.*;
5  import java.awt.event.*;
6  import java.awt.*;
7  import java.io.*;
8
9  import _100._183._37._195.axis.literatursuche_jws.*;
10
11 public class literaturgui extends JFrame {
12   private JPanel jContentPane = null;
13   private JLabel jLabel = null;
14   private JLabel jLabel1 = null;
15   private JButton jButton = null;
```

```
16    private JTextField jTextField = null;
17    private JButton jButton1 = null;
18
19    private PrintWriter g;
20
21    // Erzeugt das Fenster
22    public void init() {
23      this.setSize(300,200);
24      this.setContentPane(getJContentPane());
25      this.setTitle("Literaturausgabe");
26      this.setVisible(true);
27    }
28
29    // Erzeugt die ContentPane
30    private JPanel getJContentPane() {
31      if(jContentPane == null) {
32        jContentPane = new JPanel();
33        jContentPane.add(getJLabel1(), null);
34        jContentPane.add(getJTextField(), null);
35        jContentPane.add(getJButton(), null);
36        jContentPane.add(getJButton1(), null);
37        jContentPane.setLayout(new GridLayout(4,1));
38      }
39      return jContentPane;
40    }
41
42    // Erzeugt das Label mit der Aufforderung zur Eingabe der GID
43    private JLabel getJLabel1() {
44      if (jLabel1 == null) {
45        jLabel1 = new JLabel("                        Bitte den Autoren eingeben:");
46      }
47      return jLabel1;
48    }
49
50  //   Erzeugt das Label mit der Aufforderung zur Eingabe der GID
51    private JButton getJButton() {
52      try {
53      g = new PrintWriter(new FileWriter("Buch.xml"));
54      } catch(IOException d){
55        System.out.println(d);
56      };
57      if (jButton == null) {
58           jButton = new JButton();
59           jButton.setText("Buch ausgeben");
60           jButton.addActionListener(new ActionListener(){
61         public void actionPerformed(ActionEvent f){
62          try {
63               String t =  jTextField.getText();
64               LiteratursucheService myService = new  LiteratursucheServiceLocator();
65               Literatursuche myPort = myService.getliteratursuche();
66               g.println(myPort.xindiceAusg(t));
67               g.close();
68                BasicDOM basicDOM = new BasicDOM("Buch.xml");
69          } catch (Exception e){
70          System.out.println(e);
71          }
72        }
73           });
74      }
75      return jButton;
76    }
77
78    // Erzeugt das Feld zum Eingeben der GID
79    private JTextField getJTextField() {
80      if (jTextField == null) {
81        jTextField = new JTextField(10);
```

```
82      }
83      return jTextField;
84    }
85
86    // Erzeugt den Beenden Button
87    private JButton getJButton1() {
88      if (jButton1 == null) {
89        jButton1 = new JButton();
90        jButton1.setText("Beenden");
91        jButton1.addActionListener(new ActionListener(){
92          public void actionPerformed(ActionEvent e){
93            dispose();
94          }
95        });
96      }
97      return jButton1;
98    }
99
100   public static void main(String[] args) {
101     literaturgui libwebservice = new literaturgui();
102     libwebservice.init();
103   }
104
105 }
```

Quelltext 23.9: Literatursuche: Client – GUI

23.4. Beispielsitzung

23.4.1. Starten des Apache Tomcat, von Apache Xindice und Import der XML Daten in die Datenbank

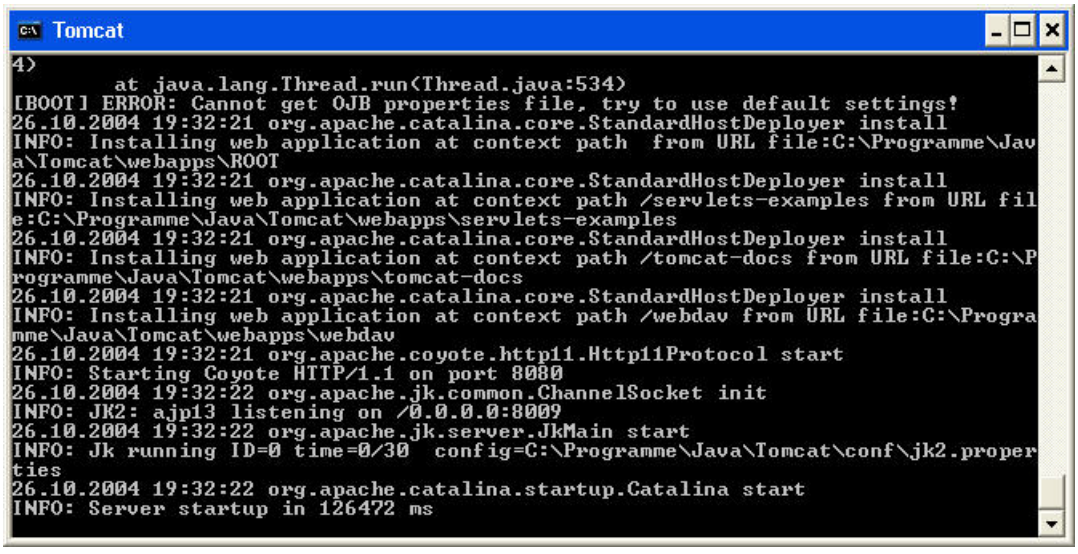


Abbildung 23.6.: Literatursuche: Gestarteter Apache Tomcat

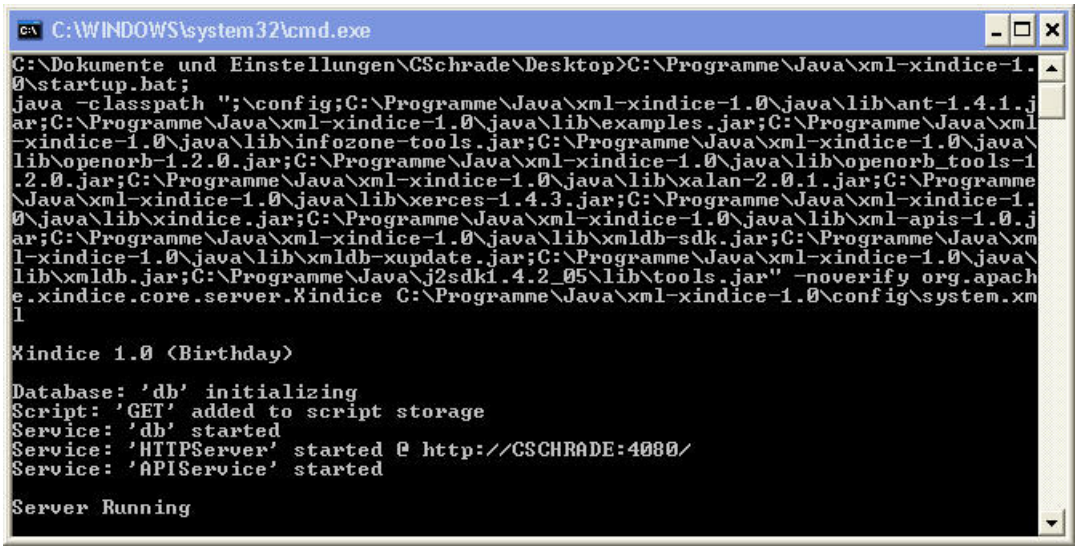


Abbildung 23.7.: Literatursuche: Gestartete Datenbank Apache Xindice

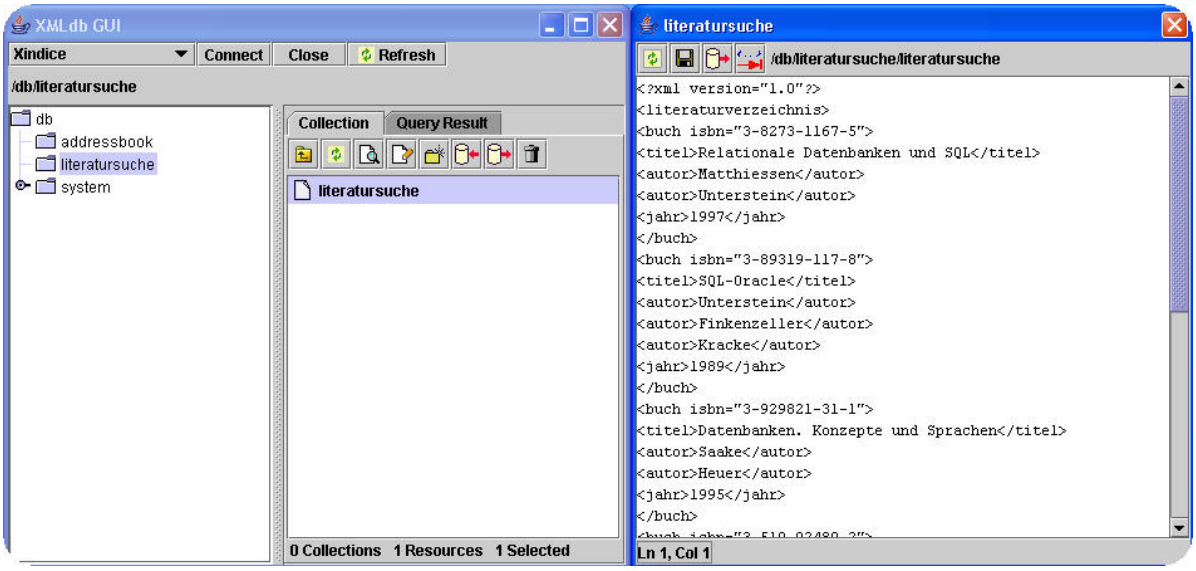


Abbildung 23.8.: Literatursuche: Datenbank GUI zur Bearbeitung der Xindice Datenbank

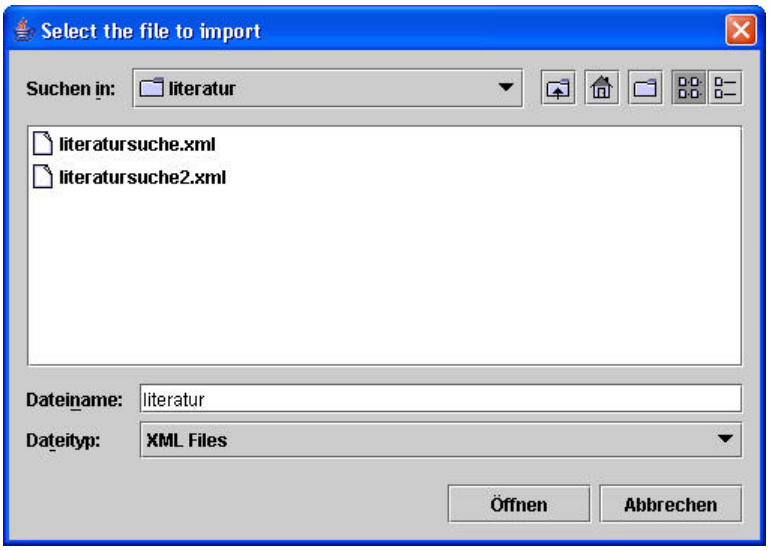


Abbildung 23.9.: Literatursuche: Fenster zum Importieren von XML-Dateien in die Datenbank

23.4.2. Starten des Java Swing GUI

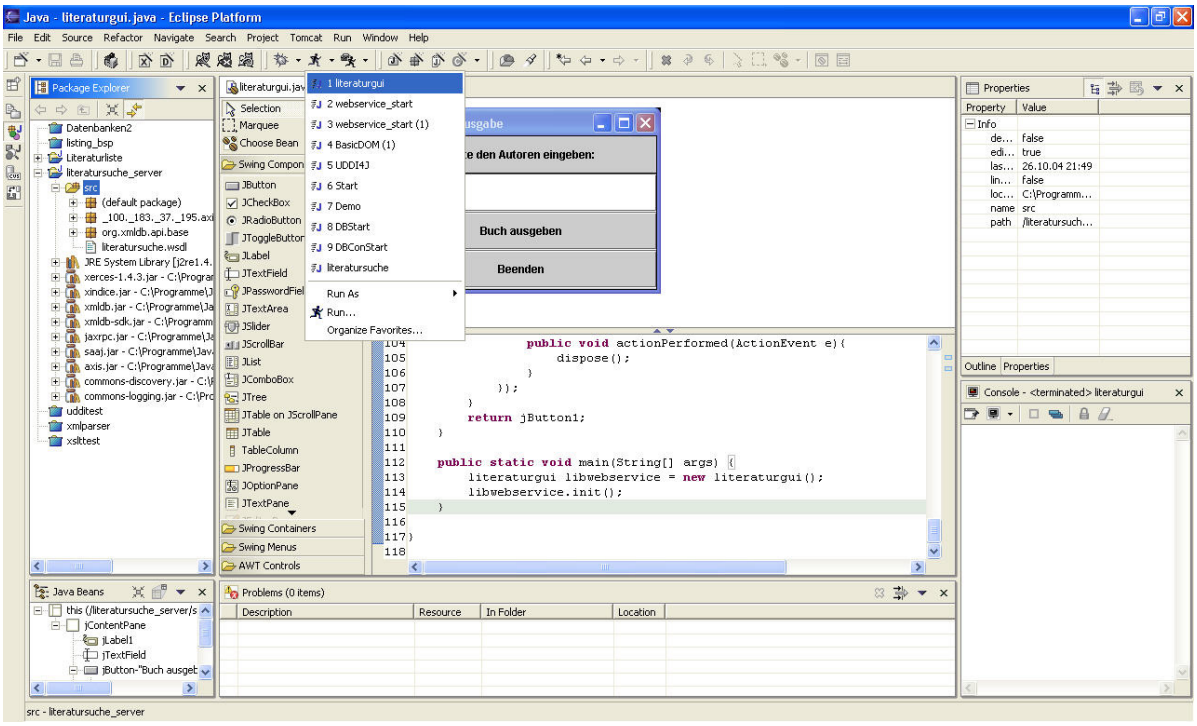


Abbildung 23.10.: Literatursuche: Starten des Java Swing GUI unter Eclipse

Abbildung 23.11.: Literatursuche: Java Swing GUI zur Literatursuche

23.4.3. Suche und Ausgabe eines Buches

Abbildung 23.12.: Literatursuche: Java Swing GUI mit Eingabe des Autors nachdem gesucht wird

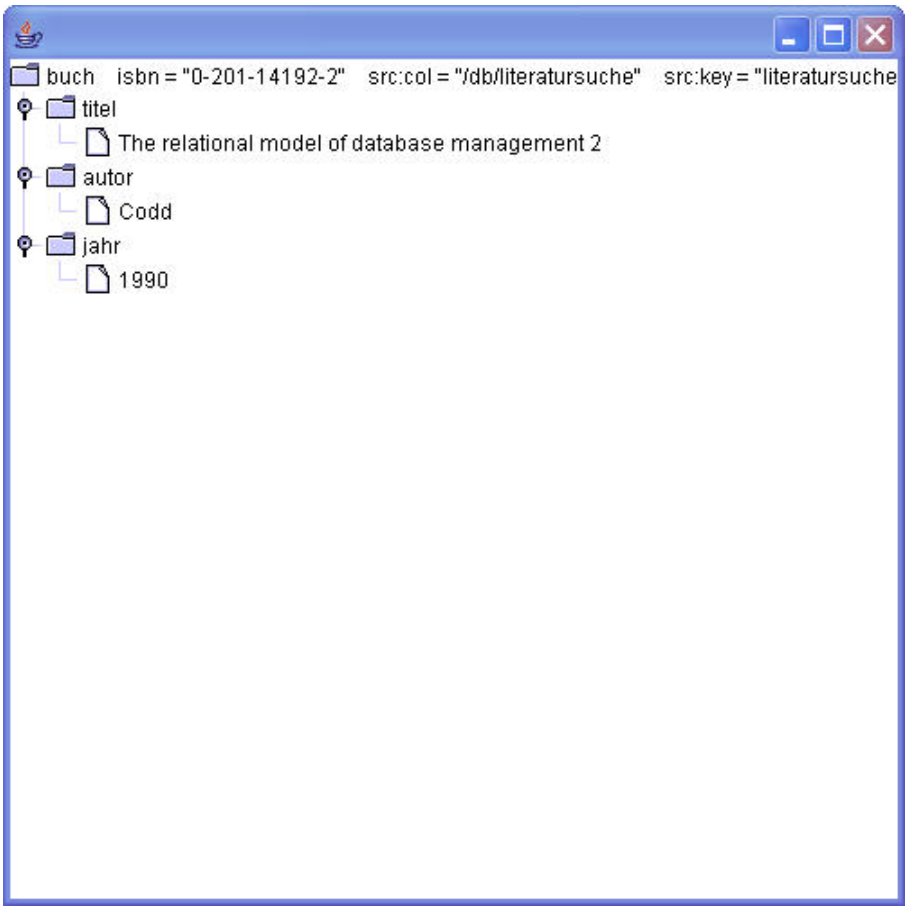


Abbildung 23.13.: Literatursuche: Ergebnis der Suche als DOM-Baum

23.5. Resümee

Diese Aufgabe einen eigenen Web-Service alleine auf die Beine stellen zu müssen war sehr hilfreich beim Verständnis der einzelnen Technologien eines Web-Service. Es wurde deutlich, dass durch die vorangegangenen Schulungen zwar ein grober Überblick vermittelt wurde, aber nur der eigene Schulungsteil wirklich so beherrscht wurde, dass er auch angewendet werden konnte.

Bezogen auf den eigenen Web-Service ist anzumerken, dass dieser zwar funktionsfähig ist, aber bei weitem in seinen Funktionen nicht das bietet was man von einem Web-Service erwarten würde. Dieses soll heißen, dass er in vielerlei Hinsicht noch erweiterbar wäre und in seinem aktuellen Zustand nur die durch die Aufgabe geforderten Basisfunktionen beinhaltet.

24. Produktkatalog

24.1. Ziel und Beschreibung des WebServices

Ziel ist es, einen WebService zu entwickeln und diesen auf dem BizWeb-Server zu implementieren. Der WebService „Produktsuche“ soll dabei als Funktionalität einen Kategorienamen als String übergeben bekommen, dabei auf eine Xindice-Datenbank[1] zugreifen und das Ergebnis in Form eines XML–Dokumentes zurückliefern. Ein GUI übernimmt die Darstellung des Ergebnis in Form eines Swing–Baumes.

Die WSDL-Beschreibung des WebService soll durch das Auto-Deployment des AXIS-Frameworks generiert werden.

Weiteres Ziel der Aufgabe ist es, jedem Projektteilnehmer die Möglichkeit zu geben, das durch die Schulungen angeeignete Wissen an einem Praxisbeispiel sinnvoll anzuwenden und dabei die Funktionalitäten und das Handhaben von WebServices zu verstehen.

24.2. Erstellen einer Xindice-Datenbank

Zunächst muss eine XML-Datei erstellt werden. Die `Hardware.xml` enthält die eigentlichen Daten, die in die Xindice-Datenbank eingepflegt werden. Enthalten sind die drei Kategorien Speicher, Laufwerk und Ausgabegerät mit drei Produkten. Die Datenbankabfrage bezieht sich dabei auf den Namen der Kategorie.

```
1  <?xml version="1.0"?>
2  <Produkte>
3    <Kategorie nr="1">
4      <Kategorie>Speicher</Kategorie>
5      <Artikel name="256 MB xD-Picture-Card">
6        <Beschreibung>Fuer ne Digicam</Beschreibung>
7        <Preis>150 EUR</Preis>
8      </Artikel>
9       </Kategorie>
10     <Kategorie nr="2">
11     <Kategorie>Laufwerk</Kategorie>
12     <Artikel name="CD-ROM">
13       <Beschreibung>Fuer den PC</Beschreibung>
14       <Preis>200 EUR</Preis>
15     </Artikel>
16      </Kategorie>
17      <Kategorie nr="3">
```

[1] Vgl. http://xml.apache.org/xindice

```
18      <Kategorie>Ausgabegerät</Kategorie>
19      <Artikel name="Monitor">
20        <Beschreibung>Fuer den PC</Beschreibung>
21        <Preis>200 EUR</Preis>
22      </Artikel>
23       </Kategorie>
24  </Produkte>
```

Quelltext 24.1: Produktkatalog: Produkte nach Kategorie im XML–Format

Nachdem die Produktliste erstellt wurde, kann diese in der Datenbank implementiert werden. Durch das Ausführen der „`startup.bat`“ wir die Datenbank gestartet und es erscheint folgendes Fenster:

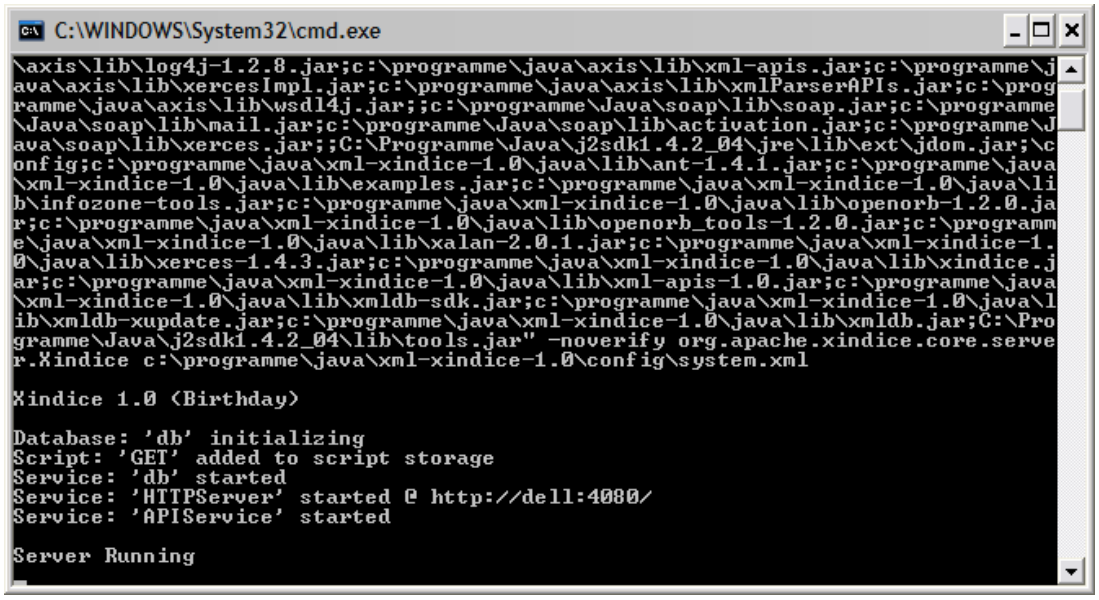


Abbildung 24.1.: Produktkatalog: Startup Xindice

In der Konsole werden danach folgende Befehle ausgeführt[2]:

Datenbankkollektion erstellen:

```
xindiceadmin ac -c /db -n produkte
```

Dokument der Datenbankkollektion hinzufügen:

```
xindiceadmin ad -c /db/produkte -f $XINDICE_HOME/java/xindice/hardware.xml -n produkte
```

Die Datenbank enthält nun die in der „`hardware.xml`“ eingegebenen Daten. Mit Hilfe von `XMLdbGUI`[3], einer grafischen Oberfläche für Xindice-Datenbanken, können die Inhalte der Datenbank angezeigt und verändert werden.

Abbildung 24.2 zeigt die geöffnete Datenbank mit deren implementierten Daten der „`hardware.xml`“.

[2] Hinweis: Die Befehle sind für den Fall ausgelegt, daß man sich im Ordner `$XINDICE_HOME/bin` befindet!

[3] titanium.dstc.edu.au/xml/xmldbgui

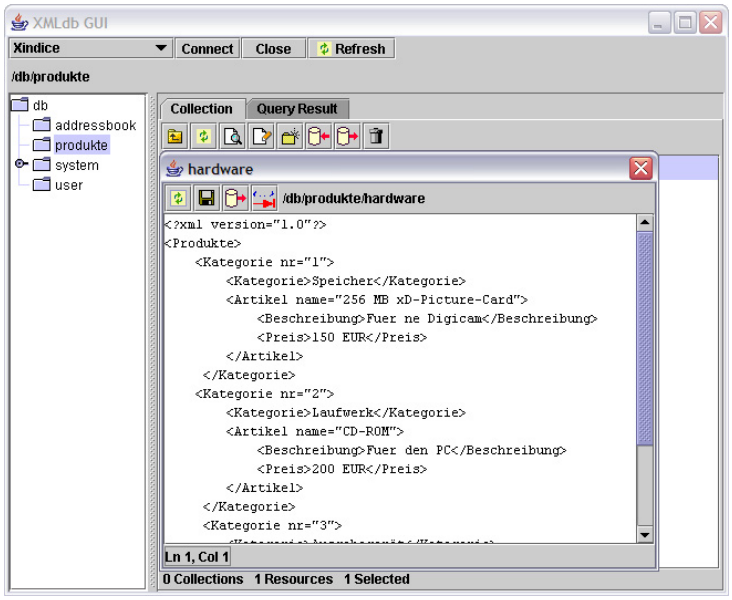


Abbildung 24.2.: Produktkatalog: Xindice–GUI

24.3. Beschreibung des WebService „Produktsuche“

Der WebService „Produktsuche“ besteht aus einer Klasse. Sie ist so konzipiert, dass sie einen „StringX“ übergeben bekommt. „StringX“ ist dabei die gewünschte Kategorie für die sich der Nutzer das vorhandene Produkt anzeigen lassen möchte.

```
1 public class Produktsuche
2 {
3    public static String ausgabe(String StringX) throws XMLDBException, Exception
4    {...}
5 }
```

Quelltext 24.2: Produktkatalog: Produktsuche.java

Zu Beginn wird eine neue Verbindung zur Datenbank hergestellt. Dazu wird der Treiber der Datenbank geladen.

```
1 /*Treiber für Xindice laden*/
2         String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
3         Class c = Class.forName(driver);
4
5         Database database = (Database) c.newInstance();
6         DatabaseManager.registerDatabase(database);
7
8         col = DatabaseManager.getCollection("xmldb:xindice:///db/produkte");
```

Quelltext 24.3: Produktkatalog: Verbindung zur Datenbank (1)

Wichtig dabei ist, dass folgende Bibliotheken der Klasse bekannt gemacht werden.

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
```

Quelltext 24.4: Produktkatalog: Verbindung zur Datenbank (2)

Anschließend wird eine Abfrage an die Datenbank gestellt, die den „StringX“ übermittelt.

```
String xpath = "//Produkte/Kategorie[Kategorie='"+StringX+"']";

         XPathQueryService service =
            (XPathQueryService) col.getService("XPathQueryService", "1.0");
         ResourceSet resultSet = service.query(xpath);
         ResourceIterator results = resultSet.getIterator();
         while (results.hasMoreResources()) {
            Resource res = results.nextResource();
           //System.out.println((String) res.getContent());
           erg = erg + (String)res.getContent();}
```

Quelltext 24.5: Produktkatalog: Abfrage an die Datenbank

Als Ergebnis wird ein String in Form eines XML-Dokumentes zurückgegeben, der den selektierten Bereich der Datenbank darstellt.

24.4. Vorgehensweise bei Erstellung des Webservice

Um einen WebService auf einem Server lauffähig zu machen, muss die oben beschriebene Java-Datei „`Produktsuche.java`“ in „`Produktsuche.jws`“ umbenannt werden und in das AXIS-Verzeichnis in „`Tomcat/webapps`“ kopiert werden. Das Auto-Deployment findet durch den Aufruf von http://195.37.183.100/axis/Produktsuche.jws über den Browser statt. Dadurch wird die WSDL-Beschreibung des Webservice erstellt. Folgendes Listing zeigt die `Produktsuche.wsdl`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://195.37.183.100/axis/Produktsuche.jws"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://195.37.183.100/axis/Produktsuche.jws"
  xmlns:intf="http://195.37.183.100/axis/Produktsuche.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns1="http://base.api.xmldb.org"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:types><schema
    targetNamespace="http://base.api.xmldb.org"
  xmlns="http://www.w3.org/2001/XMLSchema"><import
    namespace="http://schemas.xmlsoap.org/soap/encoding/"/><complexType
    name="XMLDBException"><sequence><element name="errorCode"
  type="xsd:int"/><element name="vendorErrorCode"
  type="xsd:int"/></sequence></complexType></schema></wsdl:types>
  <wsdl:message name="ausgabeResponse">
    <wsdl:part name="ausgabeReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="ausgabeRequest">
    <wsdl:part name="StringX" type="xsd:string"/>
```

```
23    </wsdl:message>
24    <wsdl:message name="XMLDBException">
25      <wsdl:part name="fault" type="tns1:XMLDBException"/>
26    </wsdl:message>
27    <wsdl:portType name="Produktsuche">
28      <wsdl:operation name="ausgabe" parameterOrder="StringX">
29        <wsdl:input message="impl:ausgabeRequest" name="ausgabeRequest"/>
30        <wsdl:output message="impl:ausgabeResponse" name="ausgabeResponse"/>
31        <wsdl:fault message="impl:XMLDBException" name="XMLDBException"/>
32      </wsdl:operation>
33    </wsdl:portType>
34    <wsdl:binding name="ProduktsucheSoapBinding" type="impl:Produktsuche">
35      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
36      <wsdl:operation name="ausgabe">
37        <wsdlsoap:operation soapAction=""/>
38        <wsdl:input name="ausgabeRequest">
39          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
40              namespace="http://DefaultNamespace" use="encoded"/>
41        </wsdl:input>
42        <wsdl:output name="ausgabeResponse">
43          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
44           namespace="http://195.37.183.100/axis/Produktsuche.jws" use="encoded"/>
45        </wsdl:output>
46        <wsdl:fault name="XMLDBException">
47          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
48           namespace="http://195.37.183.100/axis/Produktsuche.jws" use="encoded"/>
49        </wsdl:fault>
50      </wsdl:operation>
51    </wsdl:binding>
52    <wsdl:service name="ProduktsucheService">
53      <wsdl:port binding="impl:ProduktsucheSoapBinding" name="Produktsuche">
54        <wsdlsoap:address location="http://195.37.183.100/axis/Produktsuche.jws"/>
55      </wsdl:port>
56    </wsdl:service>
57  </wsdl:definitions>
```

Quelltext 24.6: Produktkatalog: WSDL–Beschreibung des WebService

24.5. Die Entwicklung des Clients

Mit der generierten WSDL-Beschreigung lässt sich nun ein Client entwickeln, mit dem es möglich ist die Funktionalitäten des WebService „Produktsuche“ in Anspruch zu nehmen. Als Entwicklungsplattform wurde Eclipse 3.0 ausgewählt, da ein Plugin verfügbar ist (`WSDL2JAVA`), mit dem auf einfachste Weise ein Client automatisiert generiert werden kann. `WSDL2JAVA` erzeugt dabei folgende Klassen:

- `Produktsuche.java`
- `ProduktsucheService.java`
- `ProduktsucheServiceLocator.java`
- `ProduktsucheSoapBindingStub.java`
- `XMLDBException.java`

Der eigentliche Aufruf, bzw. die Übergabe des Strings, geschieht über folgende Zeilen:

```
ProduktsucheService myService = new ProduktsucheServiceLocator();
Produktsuche myPort = myService.getProduktsuche();
System.out.println(myPort.ausgabe("gewünschte Kategorie"));
```

Quelltext 24.7: Produktkatalog: Web Service aufrufen

Diese Zeilen werden in den Quellcode für die entwickelte grafische Oberfläche implementiert. Diese, mit Swing entwickelte Oberfläche, ist unter dem Namen „ProduktsucheGUI.java“ verfügbar.

```
/*
 * Created on 30.09.2004
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author twaesch
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

import javax.swing.JFrame;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;

import _100._183._37._195.axis.Produktsuche_jws.*;

public class ProduktsucheGUI extends JFrame
{
  private JPanel jContentPane = null;
  private JLabel jLabel = null;
  private JLabel jLabel1 = null;
  private JButton jButton = null;
  private JTextField jTextField = null;
  private JButton jButton1 = null;
  private PrintWriter pw;

  // Erzeugt das Fenster
  public void init()
  {
    this.setSize(300,200);
    this.setContentPane(getJContentPane());
    this.setTitle("Produktsuche");
    this.setVisible(true);
  }

  // Erzeugt die ContentPane
  private JPanel getJContentPane()
  {
    if(jContentPane == null)
    {
      jContentPane = new JPanel();
      jContentPane.add(getJLabel1(), null);
      jContentPane.add(getJTextField(), null);
      jContentPane.add(getJButton(), null);
      jContentPane.add(getJButton1(), null);
      jContentPane.setLayout(new GridLayout(5,1));
    }
```

```
54      return jContentPane;
55    }
56
57    private JLabel getJLabel1()
58    {
59      if (jLabel1 == null)
60      {
61        jLabel1 = new JLabel("Bitte Kategorie eingeben:");
62      }
63      return jLabel1;
64    }
65
66    private JButton getJButton()
67    {
68      try
69      {
70        pw = new PrintWriter(new FileWriter("produktsuche.xml"));
71      }
72      catch(IOException d)
73      {
74        System.out.println(d);
75      };
76
77      if (jButton == null)
78      {
79        jButton = new JButton();
80        jButton.setText("Ausgabe des Produktes");
81        jButton.addActionListener(new ActionListener()
82        {
83          public void actionPerformed(ActionEvent f)
84          {
85            try
86            {
87              String t = jTextField.getText();
88              ProduktsucheService myService = new ProduktsucheServiceLocator();
89              Produktsuche myPort = myService.getProduktsuche();
90              String xQuerry = t;
91              pw.println(myPort.ausgabe(xQuerry));
92              pw.close();
93              BasicDOM basicDOM = new BasicDOM("produktsuche.xml");
94            }
95
96            catch (Exception e)
97            {
98                System.out.println(e.getMessage());
99              }
100         }
101       });
102     }
103     return jButton;
104   }
105
106
107   private JTextField getJTextField()
108   {
109     if (jTextField == null)
110     {
111       jTextField = new JTextField(10);
112     }
113     return jTextField;
114   }
115
116   // Erzeugt den Beenden Button
117   private JButton getJButton1()
118   {
119     if (jButton1 == null)
```

```
120     {
121       jButton1 = new JButton();
122       jButton1.setText("Beenden");
123       jButton1.addActionListener(new ActionListener()
124       {
125         public void actionPerformed(ActionEvent e)
126         {
127           dispose();
128         }
129       });
130     }
131     return jButton1;
132   }
133 }
```

Quelltext 24.8: Produktkatalog: Das GUI

Der zurückgelieferte String wird in der Datei „`produktsuche.xml`“ zwischengespeichert und mittels der Klasse „`BasicDOM.java`“ geparst.

```
1  /*
2   * Created on 10.04.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  /**
8   * @author twaesch
9   *
10  * To change the template for this generated type comment go to
11  * Window - Preferences - Java - Code Generation - Code and Comments
12  */
13
14 import org.apache.xerces.parsers.DOMParser;
15 import org.w3c.dom.Document;
16 //import org.w3c.dom.Element;
17
18 import org.xml.sax.SAXException;
19 import java.io.IOException;
20
21 //  A Simple DOM Application
22 public class BasicDOM {
23
24    // Constructor
25    public BasicDOM (String xmlFile) {
26
27       //  Create a Xerces DOM Parser
28       DOMParser parser = new DOMParser();
29
30       //  Parse the Document
31       //  and traverse the DOM
32       try {
33          parser.parse(xmlFile);
34          Document document = parser.getDocument();
35          MyFrame mf = new MyFrame(document);
36
37       } catch (SAXException e) {
38          System.err.println (e);
39       } catch (IOException e) {
40          System.err.println (e);
41       }
42    }
43 }
```

Quelltext 24.9: Produktkatalog: Die Klasse BasicDOM

Das so entstandene „Document“ wird an die Klasse MyFrame übergeben und grafisch aufbereitet, damit es in „Baumform“ dargestellt werden kann. Dazu bedient sie sich der Klassen „`MyTreeNode.java`“ und „`TreeExtender.java`“ die hier nicht näher erläutert werden.

```
1  import javax.swing.JFrame;
2  import javax.swing.JTree;
3  import org.w3c.dom.Document;
4  import org.w3c.dom.Node;
5  // import java.awt.BorderLayout;
6
7
8  class MyFrame extends JFrame
9  {
10    MyFrame (Document doc)
11    {
12       Node docRoot = doc.getDocumentElement();
13
14       JTree tree = makeTree(docRoot);
15
16       getContentPane().add(tree);
17       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18       setSize(500,500);
19       show();
20    }
21
22
23    JTree makeTree(Node docRoot)
24    {
25       MyTreeNode treeRoot = new MyTreeNode();
26       new TreeExtender(treeRoot, docRoot);
27       return new JTree(treeRoot);
28    }
29 }
```

Quelltext 24.10: Produktkatalog: Die Klasse MyFrame

24.6. Benötigte Bibliotheken

Damit der WebService mittels des Clients aufgerufen werden kann, müssen folgende Bibliotheken mit in das Projekt eingebunden werden:

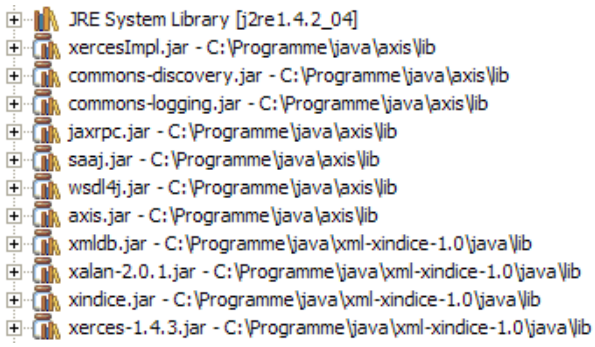


Abbildung 24.3.: Produktkatalog: Benötigte Bibliotheken

24.7. Beispielsitzung

Beim Ausführen der „`GUIstart.java`“ erscheint ein Fenster mit einem Eingabefeld sowie einem Button. Der Benutzer wird aufgefordert eine gewünschte Kategorie einzugeben und diese Eingabe mittels des Buttons „Ausgabe des Produktes“ an den WebService zu senden.

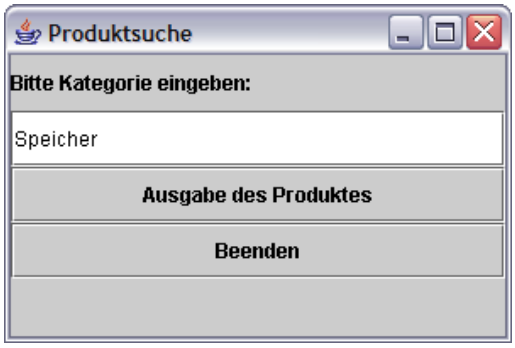


Abbildung 24.4.: Produktkatalog: Eingabemaske der Produktsuche

Das Ergebnis wird in Form eines Baumes dargestellt. Dieser enthält den Namen der Kategorie, den Namen des Produktes, dessen Beschreibung und den Preis.

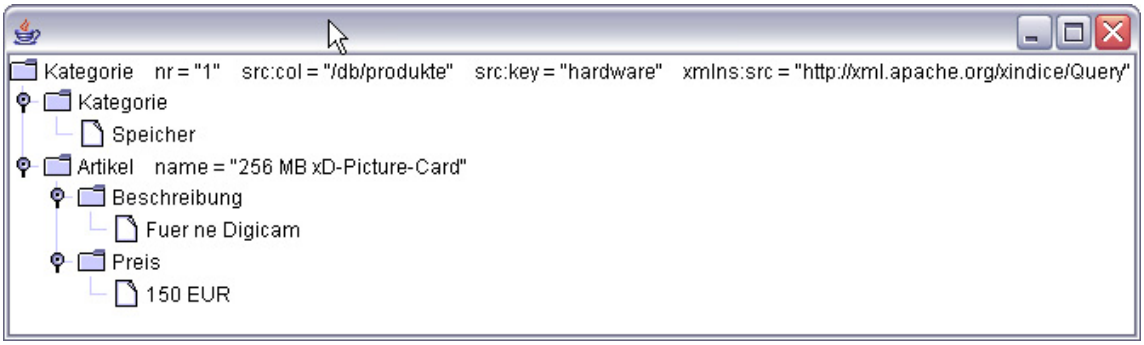


Abbildung 24.5.: Produktkatalog: Ergebnismaske der Produktsuche

24.8. Veröffentlichung des WebService im UDDI-Verzeichnis

Damit der WebService auch für die Öffentlichkeit zugänglich ist, wurde er in das öffentliche UDDI-Verzeichnis von Microsoft gestellt.

Hier ist er auf der Webseite http://uddi.microsoft.com/ unter dem Stichwort „Produktsuche“ hinterlegt worden und hat die ID:`d6e8b2ac-f0d9-4338-bd8c-82660cf99fb2`

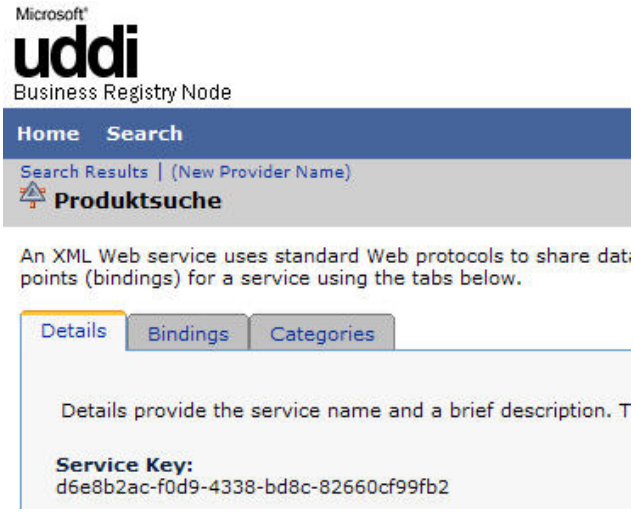


Abbildung 24.6.: Produktkatalog: UDDI MS

24.9. Direkter Aufruf des WebService über den Browser

Eine schnelle Möglichkeit seinen WebService auf Funktionalität zu prüfen besteht darin, ihn aus dem AXIS-Verzeichnis direkt über den Bowser aufzurufen. Dabei muß der `*.jws`-Datei bei der Ausführung ein Parameter übergeben werden. Für die „`Produktsuche.jws`“ sieht der Aufruf dabei folgendermaßen aus:

http://195.37.183.100/axis/Produktsuche.jws?method=ausgabe&xStringX=Speicher

Das Ergebnis wird in Form einer SOAP-Nachricht an den Browser zurückgeliefert. Dabei wird die aus der Abfrage entstandene XML-Datei als String übergeben.

```
<soapenv:Envelope>
  <soapenv:Body>
    <ausgabeResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <ausgabeReturn xsi:type="xsd:string">
        <?xml version="1.0"?>
        <Kategorie nr="1" xmlns:src="http://xml.apache.org/xindice/Query"
           src:col="/db/produkte" src:key="hardware">
          <Kategorie>Speicher</Kategorie>
          <Artikel name="256 MB xD-Picture-Card">
            <Beschreibung>Fuer ne Digicam</Beschreibung>
            <Preis>150 EUR</Preis>
          </Artikel>
        </Kategorie>
      </ausgabeReturn>
    </ausgabeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 24.11: Produktkatalog: SOAP–Nachricht

24.10. Resümee

Die Entwicklung dieses kleinen Webservices verlangte, dass man das in der Theorie erlernte Wissen nun in der Praxis umsetzen musste. Jeder im Projektteam ist damit auf dem gleichen Stand und in der Lage selbst einen WebService zu entwickeln. Außerdem wurde damit eine gute Basis für die eigentliche Projektaufgabe geschaffen: Die Umsetzung des Webservices „Kreuzfahrtbuchung“.

25. Stückliste

25.1. Ziel

Das Ziel des Web Service ist in erste Linie die praktische Umsetzung des im ersten Teil des Projektes gelernten Wissen. Dieses Wissen verbindet die Anwendung des eigentlichen Service mit den Technologien XML, WSDL, SOAP und UDDI.

Laut Aufgabenstellung ist eine Stückliste zu erstellen, welche ihre Daten aus einer Datenbank bezieht. Es soll möglich sein, in einem Client (hier: Java Swing GUI) das aufzulistende Produkt einzugeben und die daraus resultierende Stückliste graphisch (hier: DOM-Baum) darzustellen.

25.2. Beschreibung

Der dargebotene Service besteht aus zwei Elementen. Zum Einen die XML-Datenbank Xindice[1], welche die Produktdaten bereitstellt. Zum Anderen der eigentliche Web Service, welcher die Abfrage auf die Datenbank ausführt und das Ergebnis zurückliefert.

25.3. Eingebundene Bibliotheken

Um eine funktionstüchtige Entwicklungsumgebung zu haben, müssen verschiedene Programmbibliotheken in Tomcat (und später in Eclipse) eingebunden werden:

- axis.jar, axis-1_1.zip, activation.jar, commons-logging.jar, commons-discovery.jar, mail.jar, log4j-1.2.8.jar, xercesImpl.jar, xercesSamples.jar, xml-apis.jar, xmlParserAPIs.jar, xmlsec.jar, xmldb.jar, xindice.jar, wsdl4j.jar

[1] Vgl. http://xml.apache.org/xindice

25.3.1. Pfade

```
AXIS_HOME          =  C:\Java\axis
AXIS_LIB           =  %AXIS_HOME%\lib
AXIS_CLASSPATH     =  %AXIS_LIB%\axis.jar;
                      %AXIS_LIB%\commons-discovery.jar;
                      %AXIS_LIB%\commons-logging.jar;
                      %AXIS_LIB%\jaxrpc.jar;
                      %AXIS_LIB%\saaj.jar;
                      %AXIS_LIB%\log4j-1.2.8.jar;
                      %AXIS_LIB%\xml-apis.jar;
                      %AXIS_LIB%\xercesImpl.jar;
                      %AXIS_LIB%\xmlParserAPIs.jar;
                      %AXIS_LIB%\wsdl4j.jar
CATALINA_HOME      =  C:\JAVA\Tomcat
JAVA_HOME          =  C:\Programme\Java\j2sdk1.4.1_02
SOAP_HOME          =  C:\Java\soap
SOAP_LIB           =  %SOAP_HOME%\lib
SOAPCLASSPATH      =  %SOAP_LIB%\soap.jar;
                      %SOAP_LIB%\xerces.jar
XINDICE_HOME       =  C:\JAVA\xindice
XINDICECLASSPATH   =  C:\JAVA\xindice\java\lib
CLASSPATH          =  %AXISCLASSPATH%;
                      %SOAPCLASSPATH%;
                      C:\JAVA\jaf\activation.jar;
                      C:\JAVA\javamail\mail.jar;
                      C:\JAVA\jdom\build\jdom.jar;
                      C:\JAVA\xindice\java\lib\Xindice.jar
PATH               =  C:\Programme\Java\j2sdk1.4.1_02\jre\bin;
                      C:\JAVA\xindice\bin;
                      C:\Programme\Java\j2sdk1.4.1_02\bin
```

25.3.2. Stückliste

Diese beispielhafte Stückliste besteht aus drei Artikeln. Die Elemente dieser Artikel teilen sich auf in den Namen und die entsprechenden Teile. Eine Artikelnummer ist auch vorhanden, dient aber eher der Optik als dem eigentlichen Zweck. Wichtig für den Service ist das Element <name> oder besser gesagt, dessen Inhalt. Die darin enthaltene Zeichenkette ist für die Abfrage mittels der Datenbank notwendig.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <stueckliste>
3   <artikel nr.="1000">
4     <name>Panzer</name>
5     <teile>
6       <Turm/>
```

```
7         <Wanne/>
8         <Ketten/>
9       </teile>
10    </artikel>
11    <artikel nr.="1001">
12      <name>Kugelschreiber</name>
13      <teile>
14        <Mine/>
15        <Huelle/>
16        <Feder/>
17        <Minenbewegungsvorrichtung/>
18      </teile>
19    </artikel>
20    <artikel nr.="1002">
21      <name>Fleischwurst</name>
22      <teile>
23        <Pelle/>
24        <Fleisch/>
25      </teile>
26    </artikel>
27  </stueckliste>
```

Quelltext 25.1: Stückliste: Stückliste im XML-Format

Nachdem die Stückliste erstellt wurde, muss diese in die Datenbank geschrieben werden. In der Konsole sind (nachdem die Datenbank-Engine gestartet wurde) folgende Befehle notwendig[2]:

Datenbankkollektion erstellen:

```
sh xindiceadmin ac -c /db -n stueckliste
```

Dokument der Datenbankkollektion hinzufügen:

```
sh xindiceadmin ad
-c /db/stueckliste
-f $XINDICE_HOME/java/bizweb-aufgaben/xml/stueckliste.xml
-n stueckliste
```

Danach ist das Dokument in der Datenbank eingetragen und kann abgefragt werden.

25.3.3. Java-Programm

Der Service wurde minimal gehalten, von daher existiert nur eine Java-Klasse: „stueck-listedb“.

```
1
2  import org.xmldb.api.base.*;
3  import org.xmldb.api.modules.*;
4  import org.xmldb.api.*;
```

Quelltext 25.2: Stückliste: Klasse stuecklistedb – Import

[2]Hinweis: Die Befehle sind für den Fall ausgelegt, daß man sich im Ordner *$XINDICE_HOME/bin* befindet!

Der Import muss da Paket xmldb.jar eingebunden werden. Es enthält die notwendigen Klassen, um u.a. eine Verbindung zwischen Datenbank und Java-Klasse(n) herzustellen.

```
1
2  public class stuecklistedb
3  {
4     public static String ausgabe(String StringX) throws XMLDBException, Exception
5     {
6        Collection col = null;
7        String erg = "";
8        try
9        {
10             String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
11             Class c = Class.forName(driver);
12
13             Database database = (Database) c.newInstance();
14             DatabaseManager.registerDatabase(database);
15
16             col = DatabaseManager.getCollection("xmldb:xindice:///db/stueckliste");
```

Quelltext 25.3: Stückliste: Klasse stuecklistedb – Datenbankanbindung

In diesem Teil des Programms wird die Verbindung zur Datenbank hergestellt.

```
1
2              String xpath = "//stueckliste/artikel[name='"+StringX+"']";
3              XPathQueryService service =
4                 (XPathQueryService) col.getService("XPathQueryService", "1.0");
5              ResourceSet resultSet = service.query(xpath);
6              ResourceIterator results = resultSet.getIterator();
7              while (results.hasMoreResources())
8              {
9                 Resource res = results.nextResource();
10                erg = erg + (String)res.getContent();
11             }
12       }
```

Quelltext 25.4: Stückliste: Klasse stuecklistedb – Abfrage

Hier wird ein XPath (Pfad) festgelegt. Diese Abfrage enthält den String „StringX“. Dieser wird später vom Benutzer über den Client an den Service weitergegeben.

Dieser XPath wird einem XPathQueryService übergeben. Das Ergebnis des Querys wird in einem ResultSet gespeichert. Dieses wird wiederum in die Variable `erg`.

```
1
2         catch (XMLDBException e)
3         {
4            System.err.println("XML:DB Exception occured ");
5         }
6         finally
7         {
8            if (col != null)
9            {
10              col.close();
11           }
12        }
13        return erg;
14     }
15  }
```

Quelltext 25.5: Stückliste: Klasse stuecklistedb – Rückgabe

Zum Schluss wird `erg` zurückgeliefert und steht dem Client zu Verfügung.

25.3.4. WSDL

Zum Aufrufen der WSDL-Beschreibung für den Service „StuecklisteDB“ wird folgende URL benötigt:

http://195.37.183.100/axis/stuecklistedb.jws?wsdl

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <wsdl:definitions
4    targetNamespace="http://195.37.183.100/axis/stuecklistedb.jws"
5    xmlns="http://schemas.xmlsoap.org/wsdl/"
6    xmlns:apachesoap="http://xml.apache.org/xml-soap"
7    xmlns:impl="http://195.37.183.100/axis/stuecklistedb.jws"
8    xmlns:intf="http://195.37.183.100/axis/stuecklistedb.jws"
9    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
10   xmlns:tns1="http://base.api.xmldb.org"
11   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
12   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
13   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
14   <wsdl:types>
15   <schema targetNamespace="http://base.api.xmldb.org"
16       xmlns="http://www.w3.org/2001/XMLSchema">
17      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
18      <complexType name="XMLDBException">
19      <sequence>
20        <element name="errorCode" type="xsd:int"/>
21           <element name="vendorErrorCode" type="xsd:int"/>
22        </sequence>
23       </complexType>
24     </schema>
25   </wsdl:types>
26   <wsdl:message name="ausgabeResponse">
27     <wsdl:part name="ausgabeReturn" type="xsd:string"/>
28   </wsdl:message>
29   <wsdl:message name="ausgabeRequest">
30     <wsdl:part name="StringX" type="xsd:string"/>
31   </wsdl:message>
32   <wsdl:message name="XMLDBException">
33     <wsdl:part name="fault" type="tns1:XMLDBException"/>
```

```
34    </wsdl:message>
35    <wsdl:portType name="stuecklistedb">
36      <wsdl:operation name="ausgabe" parameterOrder="StringX">
37        <wsdl:input message="impl:ausgabeRequest" name="ausgabeRequest"/>
38        <wsdl:output message="impl:ausgabeResponse" name="ausgabeResponse"/>
39        <wsdl:fault message="impl:XMLDBException" name="XMLDBException"/>
40      </wsdl:operation>
41    </wsdl:portType>
42    <wsdl:binding name="stuecklistedbSoapBinding" type="impl:stuecklistedb">
43      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
44      <wsdl:operation name="ausgabe">
45        <wsdlsoap:operation soapAction=""/>
46        <wsdl:input name="ausgabeRequest">
47          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
48              namespace="http://DefaultNamespace"
49              use="encoded"/>
50        </wsdl:input>
51        <wsdl:output name="ausgabeResponse">
52          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
53              namespace="http://195.37.183.100/axis/stuecklistedb.jws"
54              use="encoded"/>
55        </wsdl:output>
56        <wsdl:fault name="XMLDBException">
57          <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
58               namespace="http://195.37.183.100/axis/stuecklistedb.jws"
59               use="encoded"/>
60        </wsdl:fault>
61      </wsdl:operation>
62    </wsdl:binding>
63    <wsdl:service name="stuecklistedbService">
64      <wsdl:port binding="impl:stuecklistedbSoapBinding" name="stuecklistedb">
65        <wsdlsoap:address location="http://195.37.183.100/axis/stuecklistedb.jws"/>
66      </wsdl:port>
67    </wsdl:service>
68  </wsdl:definitions>
```

Quelltext 25.6: Stückliste: WSDL–Beschreibung

25.3.5. UDDI

Ein Web Service verliert seinen Sinn, wenn er nicht von der Allgemeinheit gefunden werden kann. Verschiedene namhafte Firmen bieten ihrerseits solche sogenannten Verzeichnisdienste an, in dem die Anbieter von Web Services ihre Dienste mit Beschreibung und Link(s) hinterlegen können.

Der Stücklisten - Web Service wurde bei Microsoft[3] hinterlegt. Nach der Auswahl eines „Providers“ (z.B. Firmenname des Anbieters, Gruppenname der Services etc.) können der bzw. die Web Services eingetragen werden.

[3]Vgl. http://uddi.microsoft.com

Er hat den Service Key `ada4dafb-950d-4ca9-a9a7-c98683d39d7d`.

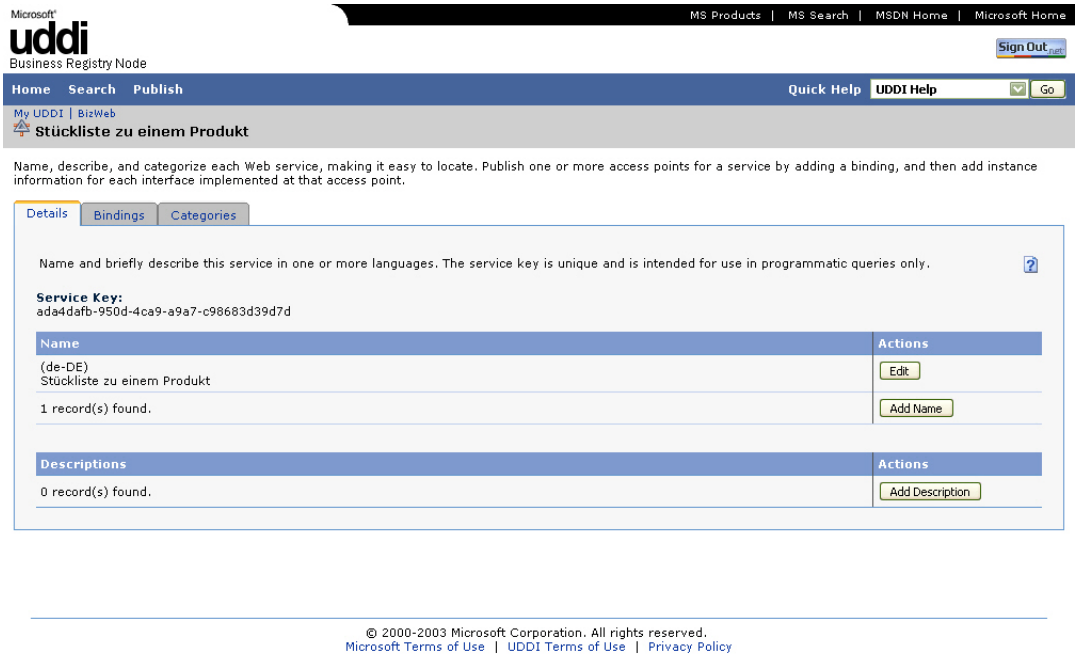


Abbildung 25.1.: Stückliste: UDDI-Eintrag bei Microsoft

25.3.6. Client

Als Client wurde ein einfaches Java Swing GUI entwickelt. Dieses besteht aus einem JFrame (Grind Layout). Es existiert eine JTextField, in dem der Name des zu ermittelnden Produkts eingegeben werden kann. Nach drücken der Schaltfläche „Ausgabe der Teile“ wird der Artikel an den Web Service übergeben. Nach Ausführung wird das Ergebnis in eine XML-Datei geschrieben. Die Klasse „BasicDOM“ wandelt diese Datei um und gibt sie in einem neuen Fenster mit einem DOM-Baum aus. (Siehe Beispielsitzung auf Seite 314)

25.4. Beispielsitzung

25.4.1. Mittels Java-Client

Nach dem Aufruf des Programms erscheint folgendes Fenster:

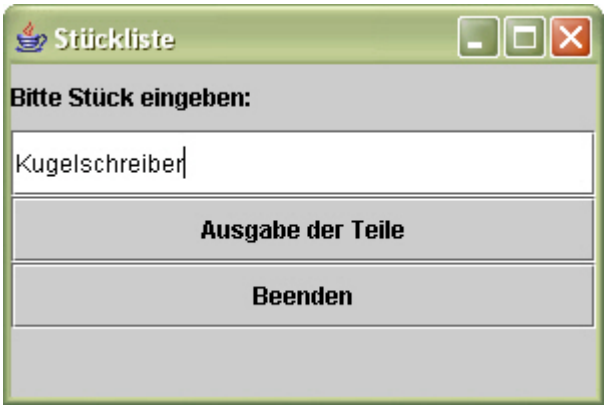


Abbildung 25.2.: Stückliste: Eingabemaske

Der Anwender hat hier die Möglichkeit, ein in der Datenbank vorhandes Produkt in das entsprechende Feld einzutragen. Mit einem Klick auf die Schaltfläche „Ausgabe der Teile“ wird ein neues Fenster aufgerufen:

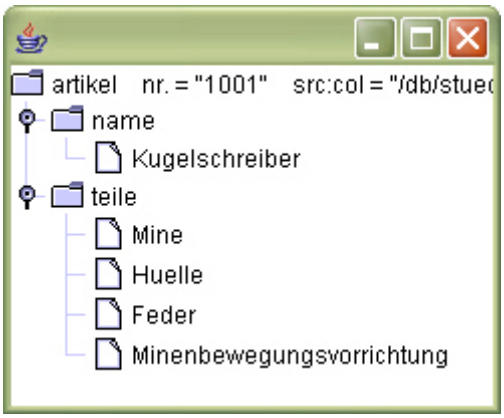


Abbildung 25.3.: Stückliste: Ausgabemaske

Hier zu sehen ist die ausgeklappte Stückliste. Es ist eine sehr primitive Darstellungsweise, welche sich nicht an die gängigen Normen von Stücklisten hält, zur beispielhaften Demonstration des Web Service allerdings ausreichend.

25.4.2. Quelltext

Vorbereitend, um das Programm in „Baumstruktur“ darzustellen, müssen einige Klassen geschrieben werden. Nachfolgend der Quelltext.

```
1  import org.apache.xerces.parsers.DOMParser;
2  import org.w3c.dom.Document;
3  import org.xml.sax.SAXException;
4  import java.io.IOException;
5
6  //  A Simple DOM Application
7  public class BasicDOM {
8
9     // Constructor
10    public BasicDOM (String xmlFile)
11    {
12       //  Create a Xerces DOM Parser
13       DOMParser parser = new DOMParser();
14
15       //  Parse the Document
16       try
17     {
18          parser.parse(xmlFile);
19          Document document = parser.getDocument();
20          MyFrame mf = new MyFrame(document);
21       }
22       catch (SAXException e)
23     {
24          System.err.println (e);
25       }
26       catch (IOException e)
27     {
28          System.err.println (e);
29       }
30    }
31 }
```

Quelltext 25.7: Stückliste: BasicDOM.java

```
1  import javax.swing.JFrame;
2  import javax.swing.JTree;
3  import org.w3c.dom.Document;
4  import org.w3c.dom.Node;
5
6  class MyFrame extends JFrame
7  {
8     MyFrame (Document doc)
9     {
10       Node docRoot = doc.getDocumentElement();
11       JTree tree = makeTree(docRoot);
12       getContentPane().add(tree);
13       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14       setSize(500,500);
15       show();
16    }
17    JTree makeTree(Node docRoot)
18    {
19       MyTreeNode treeRoot = new MyTreeNode();
20       new TreeExtender(treeRoot, docRoot);
21       return new JTree(treeRoot);
22    }
23 }
```

Quelltext 25.8: Stückliste: MyFrame.java

```
1  import org.w3c.dom.Node;
2  import org.w3c.dom.NamedNodeMap;
3
4
5  class TreeExtender
6  {
7     MyTreeNode treeNode;
8     Node docNode;
9
10
11    TreeExtender (MyTreeNode treeNode, Node docNode)
12    {
13       this.treeNode = treeNode;
14       this.docNode = docNode;
15
16       treeNode.setUserObject(getLabel());
17       addChildren();
18    }
19
20
21    void addChildren()
22    {
23       Node docChild = docNode.getFirstChild();
24       while (docChild != null)
25       {
26          if (shouldDisplay(docChild))
27          {
28             MyTreeNode treeChild = new MyTreeNode();
29             treeNode.add(treeChild);
30             new TreeExtender(treeChild, docChild);
31          }
32          docChild = docChild.getNextSibling();
33       }
34    }
35
36
37    boolean shouldDisplay(Node docChild)
38    {
39       if (docChild.getNodeType() == Node.TEXT_NODE)
40       {
41          String nodeValue = docChild.getNodeValue();
42          if (nodeValue != null &&
43              !nodeValue.trim().equals(""))
44             return true;
45          else
46             return false;
47       }
48       else
49          return true;
50    }
51
52
53    String getLabel()
54    {
55       short nodeType = docNode.getNodeType();
56
57       if (nodeType == Node.TEXT_NODE)
58          return docNode.getNodeValue();
59       else if (nodeType == Node.ELEMENT_NODE)
60          return docNode.getNodeName() + stringOfAttribs();
61       else
62          return docNode.getNodeName();
63    }
64
65
```

```
66     String stringOfAttribs()
67     {
68        NamedNodeMap attribs = docNode.getAttributes();
69        String label = "";
70
71        for (int i = 0; i < attribs.getLength(); i++)
72        {
73           Node item = attribs.item(i);
74           label += "    ";
75           label += item.getNodeName();
76           label += " = \"";
77           label += item.getNodeValue();
78           label += "\"";
79        }
80
81        return label;
82     }
83
84  }
```

Quelltext 25.9: Stückliste: TreeExtender.java

Nun kann mit dem GUI begonnen werden, welches Benutzereingaben ermöglicht (siehe Abbildung 25.2).

```
1  import javax.swing.JFrame;
2  import javax.swing.*;
3  import java.awt.event.*;
4  import java.awt.*;
5  import java.io.*;
6
7  import _100._183._37._195.axis.stuecklistedb_jws.*;
8
9  public class StuecklisteDBGUI extends JFrame
10 {
11   private JPanel jContentPane = null;
12   private JLabel jLabel = null;
13   private JLabel jLabel1 = null;
14   private JButton jButton = null;
15   private JTextField jTextField = null;
16   private JButton jButton1 = null;
17   private PrintWriter pw;
18
19   // Erzeugt das Fenster
20   public void init()
21   {
22     this.setSize(300,200);
23     this.setContentPane(getJContentPane());
24     this.setTitle("Stückliste");
25     this.setVisible(true);
26   }
27
28   // Erzeugt die ContentPane
29   private JPanel getJContentPane()
30   {
31     if(jContentPane == null)
32     {
33       jContentPane = new JPanel();
34       jContentPane.add(getJLabel1(), null);
35       jContentPane.add(getJTextField(), null);
36       jContentPane.add(getJButton(), null);
37       jContentPane.add(getJButton1(), null);
38       jContentPane.setLayout(new GridLayout(5,1));
39     }
40     return jContentPane;
```

```
41    }
42
43    private JLabel getJLabel1()
44    {
45      if (jLabel1 == null)
46      {
47        jLabel1 = new JLabel("Bitte Stück eingeben:");
48      }
49      return jLabel1;
50    }
51
52    private JButton getJButton()
53    {
54      try
55      {
56        pw = new PrintWriter(new FileWriter("stueckliste.xml"));
57      }
58      catch(IOException d)
59      {
60        System.out.println(d);
61      };
62
63      if (jButton == null)
64      {
65        jButton = new JButton();
66        jButton.setText("Ausgabe der Teile");
67        jButton.addActionListener(new ActionListener()
68        {
69          public void actionPerformed(ActionEvent f)
70          {
71            try
72            {
73              String t = jTextField.getText();
74              StuecklistedbService myService = new StuecklistedbServiceLocator();
75              Stuecklistedb myPort = myService.getstuecklistedb();
76              String xQuerry = t;
77              pw.println(myPort.ausgabe(xQuerry));
78              pw.close();
79              BasicDOM basicDOM = new BasicDOM("stueckliste.xml");
80            }
81
82            catch (Exception e)
83            {
84                System.out.println(e.getMessage());
85              }
86          }
87        });
88      }
89      return jButton;
90    }
91
92
93    private JTextField getJTextField()
94    {
95      if (jTextField == null)
96      {
97        jTextField = new JTextField(10);
98      }
99      return jTextField;
100   }
101
102   // Erzeugt den Beenden Button
103   private JButton getJButton1()
104   {
105     if (jButton1 == null)
106     {
```

```
107       jButton1 = new JButton();
108       jButton1.setText("Beenden");
109       jButton1.addActionListener(new ActionListener()
110       {
111         public void actionPerformed(ActionEvent e)
112         {
113           dispose();
114         }
115       });
116     }
117     return jButton1;
118   }
```

Quelltext 25.10: Stückliste: StuecklisteDBGUI.java

25.4.3. Mittels Aufruf über URL

Es ist möglich, eine Anfrage über die URL-Zeile des Browsers zu starten. Dabei wird auf eine graphisches Oberfläche verzichtet. Die URL lautet:

http://195.37.183.100/axis/stuecklistedb.jws?method=ausgabe&value=Kugelschreiber

Folgendes wird im Browser ausgegeben:

```
1
2   <soapenv:Envelope>
3     <soapenv:Body>
4       <ausgabeResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
5         <ausgabeReturn xsi:type="xsd:string">
6           <?xml version="1.0"?>
7           <artikel nr.="1001" xmlns:src="http://xml.apache.org/xindice/Query"
8           src:col="/db/stueckliste" src:key="stueckliste">
9           <name>Kugelschreiber</name>
10            <teile>
11              <Mine />
12              <Huelle />
13              <Feder />
14              <Minenbewegungsvorrichtung />
15            </teile>
16          </artikel>
17        </ausgabeReturn>
18      </ausgabeResponse>
19    </soapenv:Body>
20  </soapenv:Envelope>
```

Quelltext 25.11: Stückliste: Methodenaufruf über URL

Kundige Benutzer können so ohne die Entwicklung eines Clients den Web Service aufrufen und ihn verwenden. Diese Methode lässt sich meist aber nur für einfach gestrickte Web Services verwenden, da für Umfangreichere entweder die Zeichenbeschränkung der URL greift oder die Metohden einen solchen Aufruf als Parameter nicht zulassen bzw. erschweren.

25.5. Resümee

Zum Abschluss der Teildokumentation wird eine kurze Übersicht über noch mögliche Verbesserungen in der nächsten Programmversion gegeben:

- Der GUI sollte die in der Datenbank gelisteten Artikel in einer Drop-Down-Box zur Auswahl stehen haben
- Die Art der Ausgabe sollte gewählt werden können
  - DOM-Baum
  - Reiner XML-Code
  - XML-Datei
  - ...
- Die Optik des DOM-Baumes sollte verbessert werden

Für die weitere praktische Arbeit war es von Vorteil, eine solche Aufgabe gestellt zu bekommen, war man doch noch nie zuvor praktisch mit Web Services in Verbindung geraten. Die hier erlernten Fähigkeiten werden im weiteren Projektverlauf von großem Nutzen sein.

26. Weltzeituhr

26.1. Ziel Web – Service

Das Ziel des Webservices ist die Ermittlung der aktuellen Uhrzeit in einem bestimmten Ort bzw. in einer bestimmten Region auf der Welt. Aufgrund der geforderten Komplexität der Aufgabe gibt der Webservice nur Informationen zu einigen vorgegebenen Orten / Regionen. Der Service demonstriert auf einfache Weise die Funktionalität der zu verwendenden Techniken die für den Aufbau eines Webservices notwendig sind.

26.2. Beschreibung Web – Service

Beim Aufruf des Web – Services bekommt der Benutzer ein GUI (Swing GUI oder in Form eines Servlets) angezeigt. Das GUI besteht aus einem Eingabefeld und einem Button. Im Eingabefeld wird der Ort bzw. die Region eingegeben zu der die aktuelle Uhrzeit gewünscht ist. Durch Klicken des Buttons wird geprüft ob der Ort (bzw. die Region) in der Liste vorhanden ist und die gewünschte Uhrzeit ausgegeben. Ist der Ort nicht verfügbar wird eine Fehlermeldung ausgegeben. Eingabe von Orten kann beliebig vorgenommen werden, der Service wird durch den Benutzer, zum Beispiel durch Schließen des Fensters beendet.

26.3. Java – Programm zum Aufruf

Siehe Punkt 26.5.3 auf Seite 325.

26.4. Beispielsitzung

26.4.1. Sitzung mit Swing GUI

Bei Start des GUI bekommt der Benutzer ein Eingabefeld und ein Button angezeigt. In das Eingabefeld kann nun ein beliebiger Ort eingeben werden.

Abbildung 26.1.: Weltuhrzeit: GUI 1

Durch Klicken des Buttons „Uhrzeit ermitteln“ wird die Anfrage an den Web Service gestartet. Ist der Ort bzw. diese Region in der vorher erstellte Liste enthalten wird der Ort mit der dazugehörigen Ortszeit zurückgeliefert.

Abbildung 26.2.: Weltuhrzeit: GUI 2

Ist der Ort bzw. die Region nicht enthalten so bekommt der Benutzer eine Fehlermeldung vom Web Service zurückgeliefert. In der Fehlermeldung wird zusätzlich der eingegebene Ort ausgegeben, so dass der Benutzer einen möglichen Tippfehler erkennen kann.

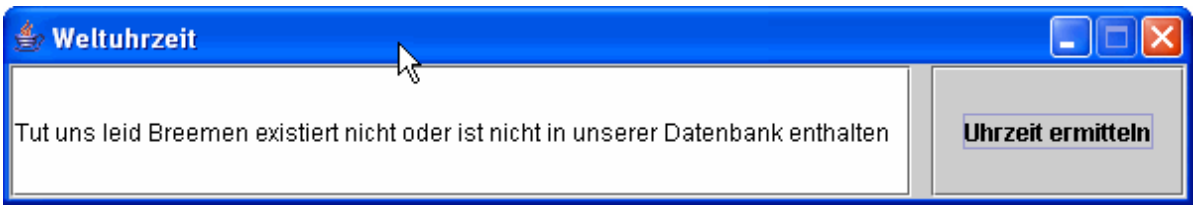


Abbildung 26.3.: Weltuhrzeit: GUI 3

26.4.2. Sitzung mit JAVA Servlet

Die Sitzung unter einem JAVA Servlet läuft genauso ab wie unter einem Swing Gui. Lediglich die Ausrichtung des Eingabefeldes und des Buttons ist ein wenig anders.

Abbildung 26.4.: Weltuhrzeit: Servlet 1

Die Ausgabe erfolgt beim Servlet unter dem Button „Uhrzeit ermitteln“

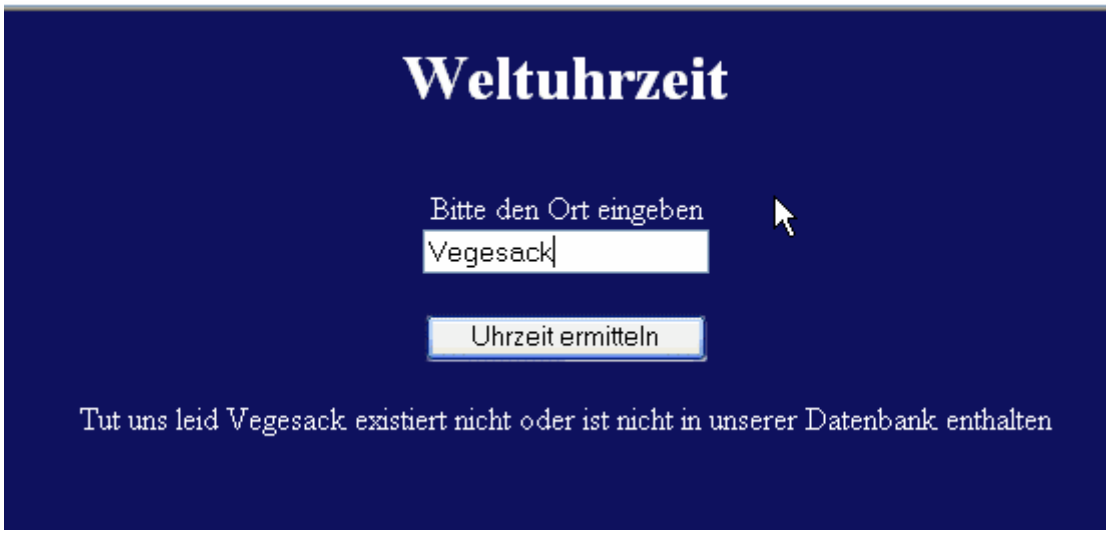


Abbildung 26.5.: Weltuhrzeit: Servlet 2

Auch hier wird eine Fehlermeldung ausgeben. In diesem Fall handelt es sich nicht um einen Schreibfehler, sondern der eingegebene Ort ist nicht in der Liste enthalten.

26.5. Beantwortung der Fragen

26.5.1. Siehe Dir die WSDL – Beschreibung Deines Web – Services an. Welche URL musst Du dazu verwenden?

Folgende URL ist zum Aufruf der WDSL – Beschreibung des Web – Services notwendig:

http://localhost/axis/weltuhrzeit.jws?wsdl

Liegt der Web – Service auf einem Server muss das „localhost“ entsprechend angepasst werden.

26.5.2. Erstelle ein Eclipse Projekt und generiere mittels eines WDSL2JAVA – Plugins einen Client für Deinen Webservice.

Die erstellte WDSL Datei wird nun in das erstellte Projekt „weltuhrzeit_clientV01“ kopiert. Mittels Rechtsklick kann über das WSDL2JAVA Plugin der Client generiert werden.

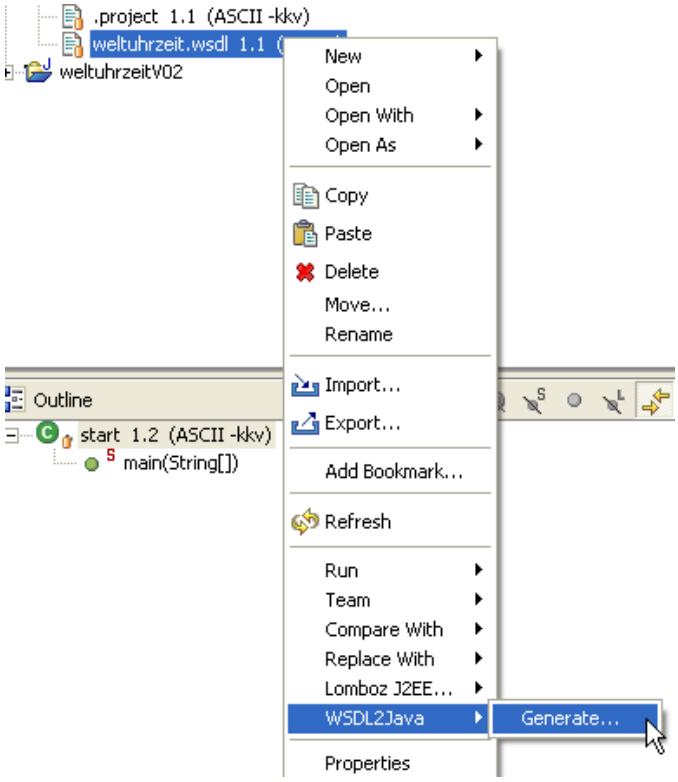


Abbildung 26.6.: Weltuhrzeit: WSDL2JAVA 1

Das Plugin erstellt nun automatisch ein Verzeichnis mit folgenden Dateien:

- Weltuhrzeit.java, WeltuhrzeitService.java, WeltuhrzeitServiceLocator.java, WeltuhrzeitSoapBindingStub.java

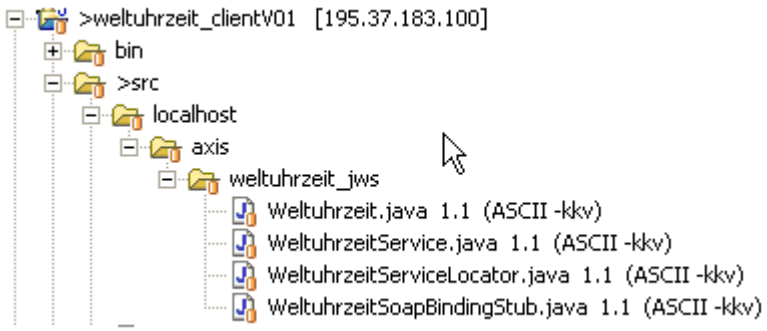


Abbildung 26.7.: Weltuhrzeit: WSDL2JAVA 2

26.5.3. Erstelle ein JAVA Programm und rufe Deinen Web Service auf

Folgendes Programm ruft den Web Service auf und übergibt als Anfrage die Stadt New York.

```
1  /*
2   * Created on 21.09.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7
8  import localhost.axis.weltuhrzeit_jws.*;
9  /**
10  * @author Jan Stelmaszek
11  *
12  * Dies ist einfaches Programm zum Aufruf des Webservices Weltuhrzeit
13  */
14 public class Client_zum_Aufruf {
15
16   public static void main (String [] args) throws Exception {
17
18     WeltuhrzeitService myService = new WeltuhrzeitServiceLocator();
19       Weltuhrzeit myPort = myService.getweltuhrzeit();
20
21       System.out.println(myPort.getTime("New York"));
22     }
23
24 }
```

Quelltext 26.1: Weltuhrzeit: Swing – GUI

Das Ergebnis wird zunächst nur in der Konsole ausgegeben:

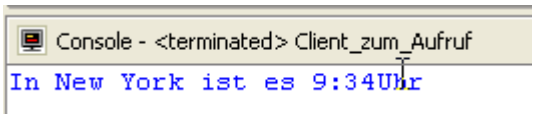


Abbildung 26.8.: Weltuhrzeit: Aufruf des Clients

26.5.4. Erstelle ein Swing – GUI für Deinen Client

Im Nachfolgenden ist der Quellcode für das Swing GUI abgebildet:

```
1  import javax.swing.JFrame;
2  import java.awt.BorderLayout;
3  import javax.swing.JPanel;
4  import javax.swing.*;
5  import localhost.axis.weltuhrzeit_jws.*;
6  import java.awt.event.*;
7  /*
8   * Created on 06.10.2004
9  /**
10  * @author Jan
11  *
12  * GUI für den Webservice Weltuhrzeit ausgeben,
13  * besteht aus einem Textfeld und einem Button
14  */
15  public class gui  extends JFrame
16  {
17   JTextField text = new JTextField();
18
19   /**
20    * Standard Konstruktor
21    */
22   public gui() {
23     super();
24     initialize();
25   }
26   /**
27    * erstellt Frame und setzt entsprechene Attribute
28    *
29    * @return void
30    */
31   private void initialize() {
32     this.setContentPane(getJPanel());
33     this.setSize(600, 100);
34     this.setTitle("Weltuhrzeit");
35     this.setVisible(true);
36     this.setLocation(300, 300);
37     this.setResizable(false);
38
39     //EventHandler für Fenster schliessen einrichten
40     this.addWindowListener(new WindowAdapter() {
41       public void windowClosing(WindowEvent e) {
42         setVisible(false);
43         dispose();  }
44     });
45
46
47   }
48   /**
49    * Erstellt JPanel und setzt das Layout für
```

```
50     * den JButton und JTextField
51     * @return JPanel
52     */
53    private JPanel getJPanel() {
54
55      JPanel panel = new JPanel(new BorderLayout(10,10) );
56      panel.add(getJButton(), java.awt.BorderLayout.EAST);
57      panel.add(text, java.awt.BorderLayout.CENTER);
58      pack();
59
60      return panel;
61    }
62    /**
63     * erstellt Jbutton und entsprechenden
64     * Action Listener der bei Betätigung
65     * den Webservice aufruft und den
66     * eingegebenen String vom JTextfield
67     * übergibt und die entsprechende Uhrzeit
68     * zurückliefert
69     *
70     * @return JButton
71     */
72    private JButton getJButton() {
73
74      JButton jButton = new JButton();
75      jButton = new javax.swing.JButton();
76      jButton.setText("Uhrzeit ermitteln");           jButton.setSize(10, 10)
77      jButton.addActionListener(new ActionListener()
78        {public void actionPerformed (ActionEvent event){
79
80          String scommand = event.getActionCommand();
81          System.out.println(scommand);
82
83          Object obj = event.getSource();
84
85      if(scommand.equals("Uhrzeit ermitteln"))
86      {
87  try{              WeltuhrzeitService myService =
88        new WeltuhrzeitServiceLocator();
89        Weltuhrzeit myPort = myService.getweltuhrzeit();
90
91        String ort = new String(text.getText());
92                                  text.setText(myPort.getTime(ort));
93      }catch(Exception er){};
94          }
95          }
96      });
97      return jButton;
98    }
99
100 }
```

Quelltext 26.2: Weltuhrzeit: WebService – Aufruf

Abbildung des GUI:

Abbildung 26.9.: Weltuhrzeit: GUI – Abbildung

Das GUI ist sehr einfach gehalten und besteht nur aus einem Panel in dem wiederum ein Eingabefeld und ein Button vorhanden sind. Das Panel ist in einem JFrame eingebunden.

26.5.5. Dokumentiere den Informationsfluss zwischen Client und Server

Versucht mit folgendem Aufruf in der Konsole:
`javaw org.apache.axis.utils.tcpmon 5555 localhost 8080`

Fehlermeldung. Java class main nicht gefunden

26.5.6. Veröffentliche Deinen Webservice in einem UDDI – Verzeichnis

Der Web Service wurde beim UDDI Test Service von Microsoft veröffentlicht

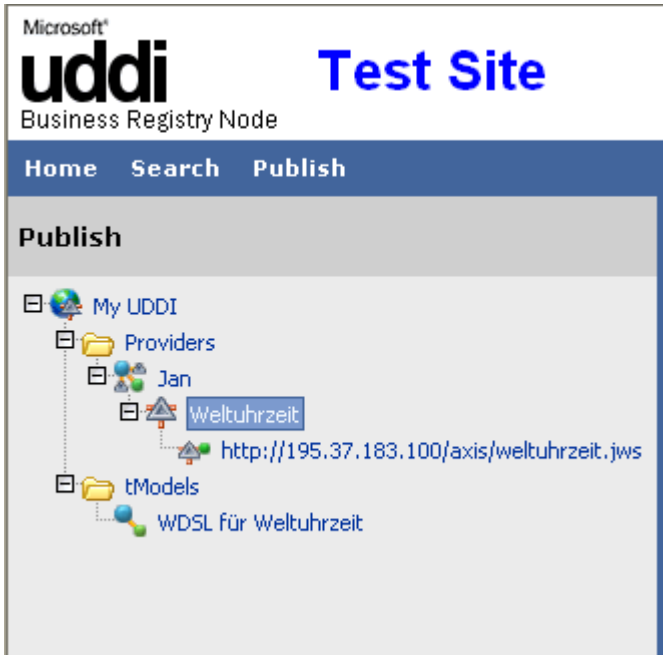


Abbildung 26.10.: Weltuhrzeit: UDDI 1

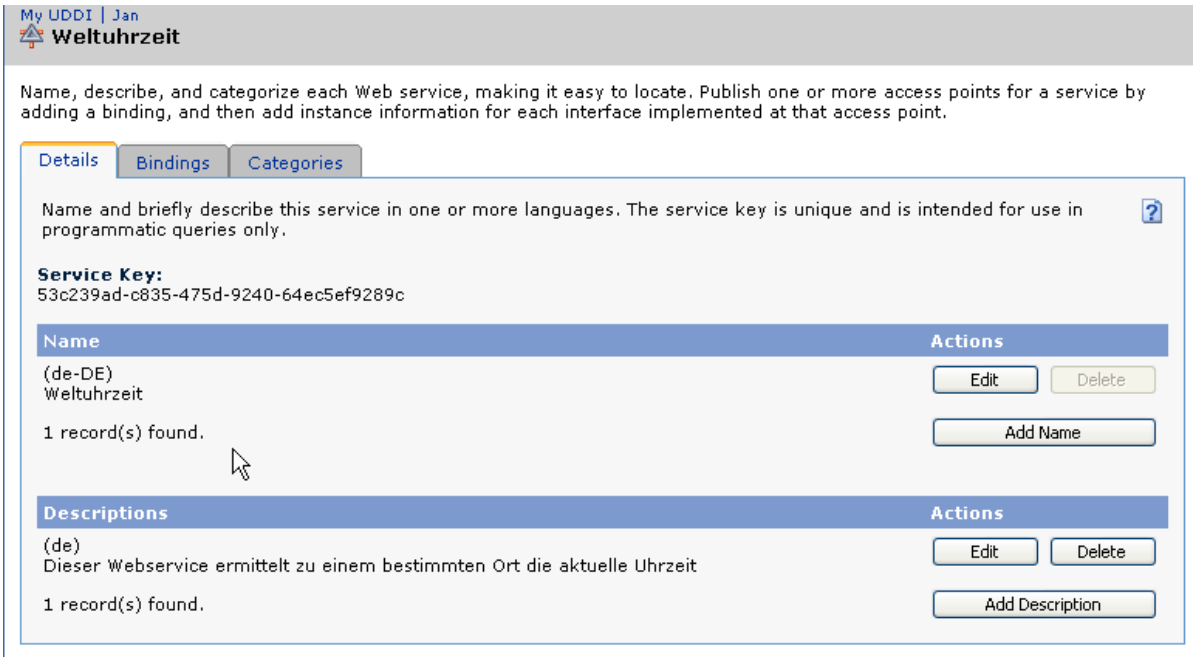


Abbildung 26.11.: Weltuhrzeit: UDDI 2

Der veröffentlichte Dienst wird nun gesucht und auch gefunden:

Abbildung 26.12.: Weltuhrzeit: UDDI – Suche

26.5.7. Idee: Kannst Du Deinen Web Service mit Parametern und einem darauf folgenden Ergebnis über eine URL im Web Browser aufrufen?

Mit der Angabe der unter JAVA implementierten Methode (method = getTime) und der Angabe eines Parameters (ort = Tokio) kann der Webservice über einen Browser aufgerufen werden.

Aufruf lokal:
http://localhost/axis/weltuhrzeit.jws?method=getTime&ort=Tokio

Aufruf auf dem BizWeb – Server:
http://195.37.183.100/axis/weltuhrzeit.jws?method=getTime&region=Bremen

Der Browser liefert folgende SOAP Nachricht zurück:

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5  <soapenv:Body>
6    <getTimeResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
7      <getTimeReturn xsi:type="xsd:string">In Tokio ist es 22:54Uhr</getTimeReturn>
8        </getTimeResponse>
9        </soapenv:Body>
10       </soapenv:Envelope>
```

Quelltext 26.3: Weltuhrzeit: WebService – SOAP-Nachricht

26.6. Programmdokumentation

Die Dokumentation wurde mit JAVADoc vorgenommen, so dass es keiner weiteren Erläuterung bedarf:

```
/*
 * Created on 05.10.2004
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
/**
 * @author Jan Stelmaszek
 *Diese Klasse gibt zu einem vorgegebenen Ort die aktuelle Uhrzeit zurück
 */
import java.util.*;
public class weltuhrzeit
{
/**
 * @param String region
 *
 * Diese Klasse bekommt eine Region (String)übergeben
 * so fern diese Region / Ort vorhanden ist gibt es
 * eine Zeitzone (timezone).
 * Wird ein Ort gefunden wird auch die dazugehörige
 * Zeitzone gesetzt.
 * Die Klasse Calender gibt die gewünschte Stunde und
 * Minute der aktuellen Uhzeit als Integer zurück. Die
 * Integer werden in Strings umgewandelt.
 * Die Stunden und Minuten werden nun als ein gemeinsamer
 * String mit dem dazugehörigen Ort zurückgegeben.
 *
 * Ist der Ort / die Region nicht vorhanden. Liefert das
 * Programm eine Fehlermedlung zurück.
 * @return String Uhrzeit
 */
 public String getTime(String region){

  Calendar cal = new GregorianCalendar();
  String time = new String();

  if (region.equals("Bremen"))
  cal.setTimeZone( TimeZone.getTimeZone("GMT+2") );
  else if (region.equals("Hawaii"))
   cal.setTimeZone( TimeZone.getTimeZone("GMT-9") );
  else if (region.equals("Alaska"))
   cal.setTimeZone( TimeZone.getTimeZone("GMT-7") );
  else if (region.equals("Chicago"))
   cal.setTimeZone( TimeZone.getTimeZone("GMT-5") );
  else if (region.equals ( "New York" ))
   cal.setTimeZone( TimeZone.getTimeZone("GMT-4") );
  else if (region.equals("Brasilien") )
   cal.setTimeZone( TimeZone.getTimeZone("GMT-2") );
  else if (region.equals("Athen") )
   cal.setTimeZone( TimeZone.getTimeZone("GMT+3") );
  else if (region.equals("Moskau"))
   cal.setTimeZone( TimeZone.getTimeZone("GMT+4") );
  else if (region.equals("Peking") )
   cal.setTimeZone( TimeZone.getTimeZone("GMT+8") );
  else if (region.equals("Tokio") )
   cal.setTimeZone( TimeZone.getTimeZone("GMT+9") );
  else if (region.equals("Fidschi"))
   cal.setTimeZone( TimeZone.getTimeZone("GMT+12") );
  else
```

```
60   {
61    time = "Tut uns leid " + region + " existiert nicht oder
62             ist nicht in unserer Datenbank enthalten";
63   System.out.print(time);
64   return (time);
65   }
66
67   final Integer h  = new Integer (cal.get(Calendar.HOUR_OF_DAY));
68   final Integer m = new Integer (cal.get(Calendar.MINUTE));
69
70 //Die Stunden und Minutenangabe wird hier in einen String umgwandelt
71   String hour    = h.toString();
72   String minute  = m.toString();
73
74 time = "In " + region + " ist es " + hour + ":"+ minute + "Uhr";
75 System.out.println(time);
76
77 return time;
78 }
```

Quelltext 26.4: Weltuhrzeit: Programmdokumention

26.7. Resümee

Die Entwicklung dieses kleinen Webservices verlangte, dass man das in der Theorie erlernte Wissen nun in der Praxis umsetzen musste. So konnte man sich intensiv mit dem Thema Web Services auseinandersetzen und von den auftretenden Problemen(z.B. fehlende importierte JAR Files etc.) lernen. So wurde eine gute Basis für die Umsetzung des Webservices „Kreuzfahrtbuchung“ im nächsten Semester geschaffen.

27. Kreuzfahrtbuchung

27.1. Ziel des Web Service

Der Webservice soll einem Benutzer oder Kunden die Möglichkeit bieten eine Kreutzfahrt, die er nach belieben aussuchen kann, über das Internet buchen zu können. Nach der Buchung erhält der Kunde eine Buchungsbestätigung z.B. eine Buchungsnummer mit der der Kunde dann seine Reise antreten kann. Falls der Benutzer bereits ein Kunde des Reiseveranstalters ist soll der Kunde die Möglichkeit haben sich über ein Account zu authentifizieren und auf diese Weise die mehrfache Eingabe seiner kompletten Daten vermeiden.

27.2. Beschreibung des Web Service

Der Web-Service ist eine Java-Klasse die mehrere Methoden enthält, diese Methoden erfüllen vier grundlegende Aufgaben:

- Login
  - Der Benutzer muss sich bei dieser Ausarbeitung des Web Services auf jeden Fall einloggen. Dazu muss er einen Benutzernamen und ein Passwort eingeben. Bei falschen Angaben muss dieser seine Eingabe wiederholen.
- Reisedaten
  - Hat sich der Benutzer korrekt Authentifiziert kann er nun über seine gewünschte Reise mit Schiff, Basis- und Zielhafen sowie Abfahrts- und Ankunftsdatum auswählen.
- Buchung
  - Hat der Benutzer alle Eingaben getätigt, muss er nun mit einem Klick auf OK die Auswahl abschließen. Die Eingaben werden zurückgegeben und im System gespeichert.

- Buchungsbestätigung
  - Zum Schluss erhält der Benutzer eine Bestätigung, dass die Buchung erfolgreich getätigt wurde und eine Buchungsnummer über die er diese Bezahlen und später in Anspruch nehmen kann.

Die Datei des Java-Programms für die Buchung einer Kreuzfahrt wurde *Kreuzfahrtbuchung_test.java* genannt. Für die Bekanntmachung des Web-Service auf dem Server muss die Datei im Ordner *%TOMCAT_HOME%/webapps/axis* abgelegt und in *Kreuzfahrtbuchung_test.jws* umbenannt werden. Die WSDL-Datei des Web-Service kann nun unter http://195.37.183.100/axis/Kreuzfahrtbuchung_test.jws?wsdl aufgerufen und als XML-Datei gespeichert werden. Diese muss dann in *Kreuzfahrtbuchung_test.wsdl* umbenannt und in ein Java-Projekt in Eclipse importiert werden. Jetzt wird das wsdl2java-Plugin genutzt um aus der WSDL-Datei die Java-Klassen und -Interfaces für die Kommunikationslogik zu generieren.

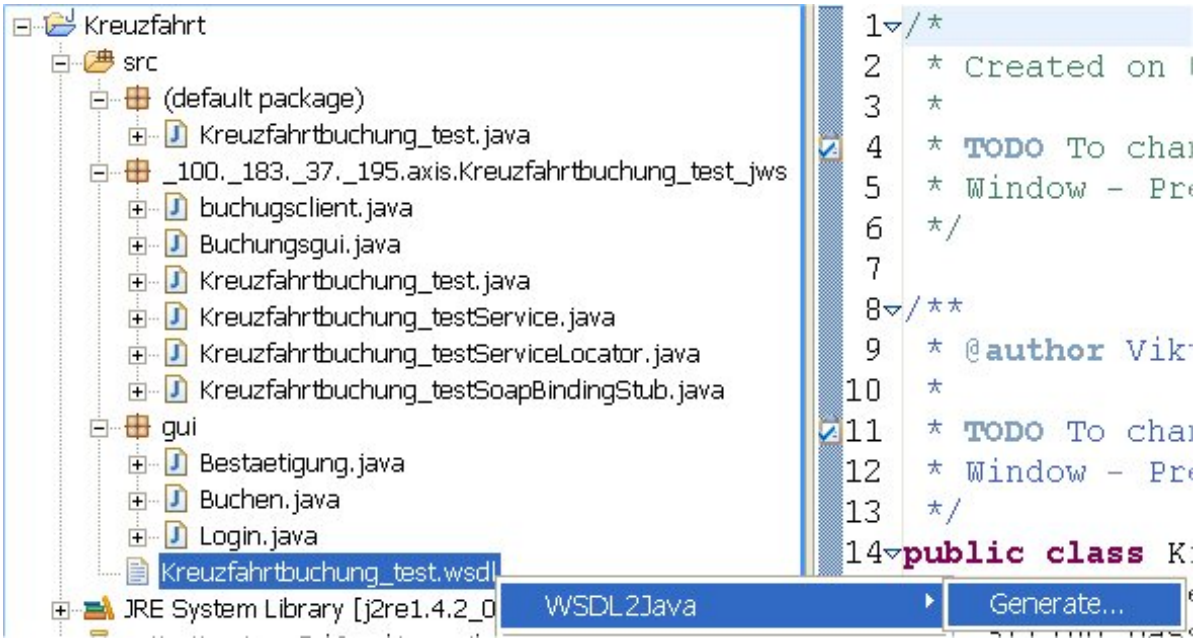


Abbildung 27.1.: Kreuzfahrtbuchung: WSDL2Java (1)

Folgende Klassen und Interfaces wurden durch das wsdl2java-Plugin erstellt:

- Kreuzfahrtbuchung_test.java
- Kreuzfahrtbuchung_testService.java
- Kreuzfahrtbuchung_testServiceLocator.java
- Kreuzfahrtbuchung_testSoapBindingStub.java

27.3. Java Programm für den Aufruf des Web Service

Folgendes Programm ruft den Web - Service auf. Es wird zweimal ein Account geprüft, eine Buchung vorgenommen, mit der Rückgabe einer Buchungsnummer und einige Daten abgefragt.

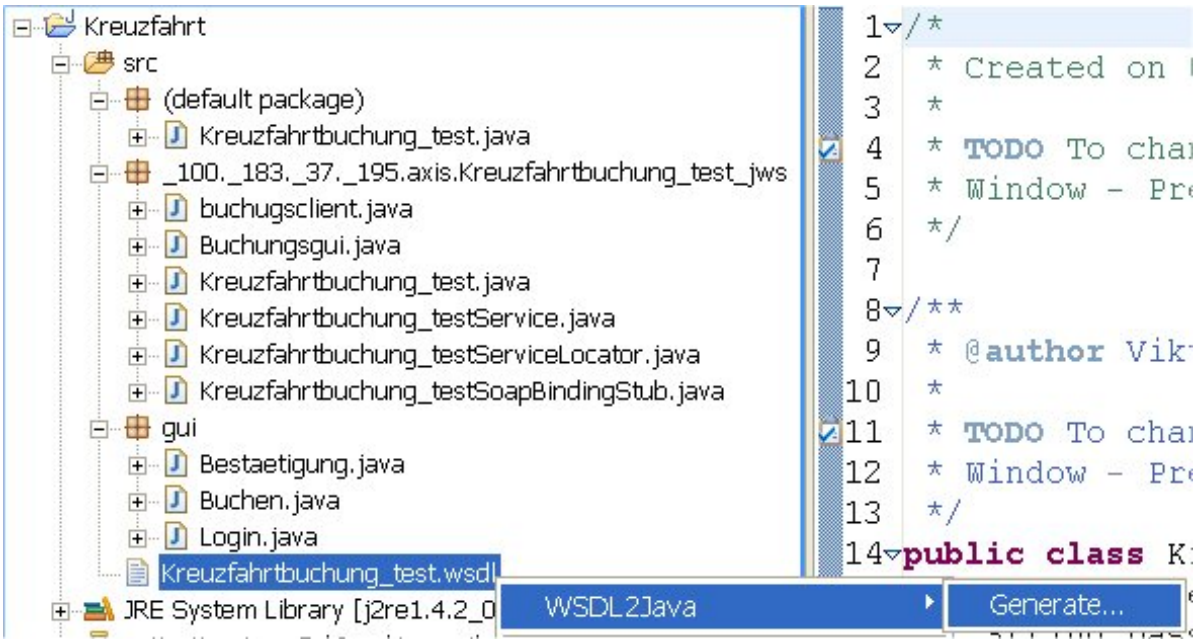


Abbildung 27.2.: Kreuzfahrtbuchung: WSDL2Java (2)

```
1  package _100._183._37._195.axis.Kreuzfahrtbuchung_test_jws;
2  /*
3   * Created on 08.11.2004
4   *
5   * TODO To change the template for this generated file go to
6   * Window - Preferences - Java - Code Style - Code Templates
7   */
8  public class buchugsclient {
9   public static void main(String[] args) throws Exception{
10   Kreuzfahrtbuchung_testService myService = new Kreuzfahrtbuchung_testServiceLocator();
11   Kreuzfahrtbuchung_test myPort = myService.getKreuzfahrtbuchung_test();
12    try{
13   int i = myPort.checkUser("Hans","123");
14   if ( i == 0){
15    System.out.println("User OK");
16   } else {
17    System.out.println("User oder PW falsch");
18   }
19   i = myPort.checkUser("Hans","321");
20   if ( i == 0){
21    System.out.println("User OK");
22   } else {
23    System.out.println("User oder PW falsch");
24   }
25   int bn = myPort.setBuchung("Prince","Karibik","1.2.2004","Bremerhaven","New York");
26   System.out.println("Ihre Buchung ist unter der Buchungsnummer: "
27             +bn+" im System abgelegt.");
28   System.out.println("Schiff: "+myPort.getShip(1));
29   System.out.println("Region: "+myPort.getRegion(1));
```

```
30    System.out.println("BasisHafen: "+myPort.getstartHafen(1));
31    System.out.println("ZielHafen: "+myPort.getzielHafen(1));
32    System.out.println("Datum: "+myPort.getDate(1));
33    System.out.println("Buchungsnummer: "+myPort.getBNr(1));
34    }catch(Exception e){
35     System.out.println("Fehler: " + e);
36    }
37   }
38  }
```

Quelltext 27.1: Kreuzfahrtbuchung: Buchung_Client

Die Ausgabe des Programms:

```
User OK
User oder PW falsch
Ihre Buchung ist unter der Buchungsnummer: 39475 im System abgelegt.
Schiff: Bremen
Region: Balearen
BasisHafen: New York
ZielHafen: Sydney
Datum: 3.4.2004
Buchungsnummer: 0
```

Abbildung 27.3.: Kreuzfahrtbuchung: Buchung_Antwort

27.4. Beispielsitzung

Nach dem Starten der Java - GUI erhält der Benutzer in diesem Beispielprogramm als erstes das Login - Fenster zu sehen.

Abbildung 27.4.: Kreuzfahrtbuchung: Login

Nach der Eingabe des Benutzernamens und des dazu korrekten Passwortes kann der Benutzer die gewünschte Reise buchen indem er ein Schiff, die Region, ein Basis- und Zielhafen und das Datum für die Reiseabfahrt auswählt.

Abbildung 27.5.: Kreuzfahrtbuchung: Auswahl

Ist der Benutzer mit seiner Auswahl zufrieden schließt er diesen Vorgang mit einem Klick auf den OK - Button abschließen und erhält eine Buchungsbestätigung mit seiner Buchungsnummer für die gewünschte Reise.

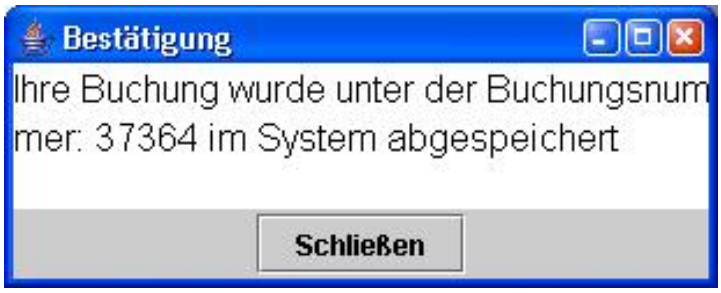


Abbildung 27.6.: Kreuzfahrtbuchung: Bestätigung

27.5. Fragen

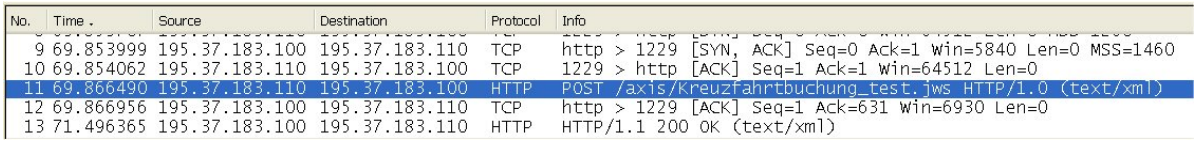
27.5.1. Welche URL ist notwendig um die WSDL-Beschreibung des Web-Services zu erhalten?

Die WSDL des Web - Services ist unter folgender URL erreichbar:

http://195.37.183.100/axis/Kreuzfahrtbuchung_test.jws?wsdl

27.5.2. Der Informationsfluss zwischen Client und Server

Der Informationsfluss zwischen dem Client und dem Server BizWeb mit der IP: 195.37.183.100 auf dem sich der Web-Service befindet wurde mit dem Tool „Ethereal“ aufgezeichnet. Dazu wurde vorher der Web-Service auf dem Server im Order *%TOMCAT_HOME%/webapps/axis* abgelegt. Anschließend wurde auf den Service mit dem GUI - Client zugegriffen. Die ersten beiden Anfragen an den Server sind, nach einem Klick auf den OK - Button im Login - Frame, nach dem Benutzernamen und dem Passwort um die Eingabe des Benutzers zu überprüfen.



No.	Time	Source	Destination	Protocol	Info
9	69.853999	195.37.183.100	195.37.183.110	TCP	http > 1229 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
10	69.854062	195.37.183.110	195.37.183.100	TCP	1229 > http [ACK] Seq=1 Ack=1 Win=64512 Len=0
11	69.866490	195.37.183.110	195.37.183.100	HTTP	POST /axis/Kreuzfahrtbuchung_test.jws HTTP/1.0 (text/xml)
12	69.866956	195.37.183.100	195.37.183.110	TCP	http > 1229 [ACK] Seq=1 Ack=631 Win=6930 Len=0
13	71.496365	195.37.183.100	195.37.183.110	HTTP	HTTP/1.1 200 OK (text/xml)

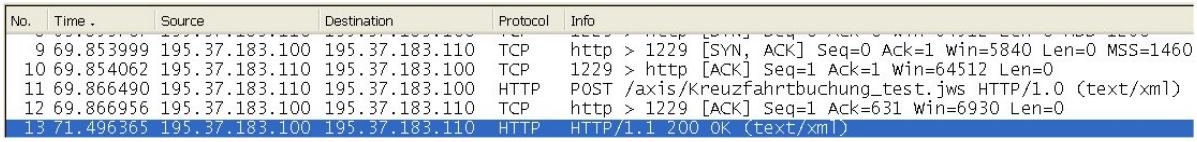
Abbildung 27.7.: Kreuzfahrtbuchung: POST_getName_Q-Z

Das POST - Packet enthält eine SOAP - Nachricht an den Web-Service, die die Methode *getName()* aufruft.

```
..SOAPAc tion: ""
..Conten t-Length
: 339... .<soapen
v:Envelo pe xmlns
:soapenv ="http:/
/schemas .xmlsoap
.org/soa p/envelo
pe/" xml ns:xsd="
http://w ww.w3.or
g/2001/X MLSchema
" xmlns: xsi="htt
p://www. w3.org/2
001/XMLS chema-in
stance"> <soapenv
:Body><n s1:getNa
me soape nv:encod
ingStyle ="http:/
/schemas .xmlsoap
.org/soa p/encodi
ng/" xml ns:ns1="
http://D efaultNa
mespace" /></soap
env:Body ></soape
nv:Envel ope>
```

Abbildung 27.8.: Kreuzfahrtbuchung: POST_getName_Inhalt

Als Antwort kommt ebenfalls eine SOAP - Nachricht mit dem Benutzernamen.



No.	Time	Source	Destination	Protocol	Info
9	69.853999	195.37.183.100	195.37.183.110	TCP	http > 1229 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
10	69.854062	195.37.183.110	195.37.183.100	TCP	1229 > http [ACK] Seq=1 Ack=1 Win=64512 Len=0
11	69.866490	195.37.183.110	195.37.183.100	HTTP	POST /axis/Kreuzfahrtbuchung_test.jws HTTP/1.0 (text/xml)
12	69.866956	195.37.183.100	195.37.183.110	TCP	http > 1229 [ACK] Seq=1 Ack=631 Win=6930 Len=0
13	71.496365	195.37.183.100	195.37.183.110	HTTP	HTTP/1.1 200 OK (text/xml)

Abbildung 27.9.: Kreuzfahrtbuchung: HTTP_NameResponse

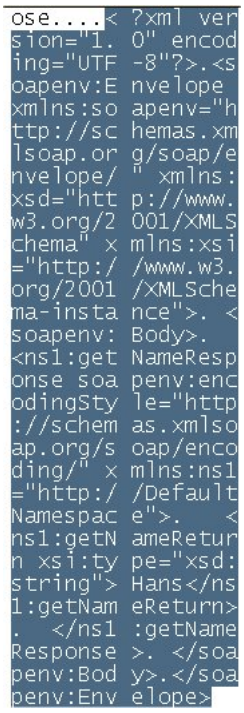


Abbildung 27.10.: Kreuzfahrtbuchung: HTTP_NameResponse_Inhalt

Danach wurde die Session geschlossen und dann mit der nächsten Anfrage nach dem Passwort von neuem aufgemacht. Dieser Vorgang wiederholt sich beim jedem Aufruf einer Methode vom Webservice.

27.5.3. Veröffentlichung in einem UDDI-Verzeichnis

Der WebService wurde auf dem Test-UDDI-Server von Microsoft veröffentlicht:

http://test.uddi.microsoft.com/

Dazu muss erst der Provider angegeben werden. Als Provider habe ich BizWeb angegeben und als Beschreibung, dass es sich um ein Projekt der HS - Bremerhaven mit dem Thema WebServices handelt. Daraufhin habe ich einen Provider - Key erhalten, der für jeden Provider einmalig ist.

Abbildung 27.11.: Kreuzfahrtbuchung: UDDI-Provider

Nachdem ich den Provider angelegt hatte, konnte ich den WebService veröffentlichen, indem ich die URL, unter der die WSDL erreichbar ist, angab.

Abbildung 27.12.: Kreuzfahrtbuchung: UDDI-Service

Testweise habe ich dann den Web-Service mit dem Suchbegriff „kreuz“ gesucht. Daraufhin wurde mir eine Liste von Ergebnissen angezeigt unter der auch mein Web-Service befand.

Abbildung 27.13.: Kreuzfahrtbuchung: UDDI-Suche

27.5.4. Kannst Du Deinen Web-Service mit Parametern und darauf folgendem Ergebnis über eine URL im Web-Browser aufrufen?

Beispiel 1

Mit Folgender URL wird die Methode *checkUser()* aufgerufen. An diese werden die Attribute user mit dem Inhalt „Hans“ und pw mit dem Inhalt „123“ übergeben.

http://195.37.183.100/axis/Kreuzfahrtbuchung_test.jws?method=checkUser&user=Hans&pw=123

Antwort:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <checkUserResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <checkUserReturn xsi:type="xsd:int">0</checkUserReturn>
  </checkUserResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 27.14.: Kreuzfahrtbuchung: SOAP - Antwort - checkUser

Das wichtigste an der langen Antwort ist *<checkUserReturn xsi:type="xsd:int">0</checkUserReturn>*. Die Methode gibt bei korrektem Benutzernamen und Passwort „0“ zurück und sonst „1“.

Beispiel 2

Die Methode *getShip()* erwartet keine Attribute und wird mit der folgenden URL aufgerufen.

http://195.37.183.100/axis/Kreuzfahrtbuchung_test.jws?method=getShip

Antwort:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <getShipResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getShipReturn xsi:type="soapenc:Array" soapenc:arrayType="xsd:string[3]"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <item>Queen Elizabeth</item>
    <item>Bremen</item>
    <item>Pride of America</item>
   </getShipReturn>
  </getShipResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 27.15.: Kreuzfahrtbuchung: SOAP - Antwort - getShip

Als Antwort erhält man ein Array, der die gewünschte Liste der Schiffsnamen enthält.

27.6. Quellcode des Web Service

```
1  /*
2   * Created on 08.11.2004
3   *
4   * TODO To change the template for this generated file go to
5   * Window - Preferences - Java - Code Style - Code Templates
6   */
7
8  /**
9   * @author Viktor
10  *
11  * TODO To change the template for this generated type comment go to
12  * Window - Preferences - Java - Code Style - Code Templates
13  */
14 public class Kreuzfahrtbuchung_test {
15   // Die Attribute in dem Web-Service werden normalerweise
```

```
16    // von einer Datenbank abgefragt. Da dies nicht in der
17    // Aufgabe gefordert war sind die entsprechenden Daten
18    // fest in den Web-Service integriert worden.
19    String name = "Hans";
20    String passwort = "123";
21    int bnr[] = new int[10];
22    String[] schiffe = {"Queen Elizabeth","Bremen","Pride of America"};
23    String[] region = {"Karibik","Balearen","Mittelmeer"};
24    String[] datum = {"1.2.2004","3.4.2004","5.6.2004"};
25    String[] startHafen = {"Bremerhaven","New York","Lissabon"};
26    String[] zielHafen = {"Honkong","Sydney","Rio de Janeiro"};
27
28    // Prüfung des Accounts, bestehend aus Benuzternamen
29    // und Passwort. Bei übereinstimmung wird 0 sonst 1
30    // zurückgegeben.
31    public int checkUser(String user, String pw){
32      if ( user.intern() == name.intern()
33        && pw.intern() == passwort.intern() ){
34        return 0;
35      } else {
36        return 1;
37      }
38    }
39
40    // Anlegen einer Buchung dazu werden Schiff,Region,
41    // Starthafen,Zielhafen und Datum erwartet. Die Methode
42    // gibt eine Buchungsnummer zurück, die normalerweise
43    // von einer Datenbank abgefragt und an den Client
44    // weiterleitet.
45    public int setBuchung(String s, String r, String sh,
46                          String zh,  String d){
47      schiffe[2] = s;
48      datum[2] = d;
49      region[2] = r;
50      startHafen[2] = sh;
51      zielHafen[2] = zh;
52      return (bnr[0] = 39475);
53    }
54
55    // Nun folgen einige get-Methoden, mit denen die
56    // Clients arbeiten können um z.b. die GUI für den
57    // Client mit allen nötigen Informationen zu füllen.
58    public String getShip(int i){
59      return schiffe[i];
60    }
61    public String getDate(int i){
62      return datum[i];
63    }
64    public String getRegion(int i){
65      return region[i];
66    }
67    public int getBNr(int i){
68      return bnr[i];
69    }
70    public String getstartHafen(int i){
71      return startHafen[i];
72    }
73    public String getzielHafen(int i){
74      return zielHafen[i];
75    }
76    public String getName(){
77      return name;
78    }
79    public String getPW(){
80      return passwort;
81    }
```

```
82   public String[] getShip(){
83     return schiffe;
84   }
85   public String[] getRegion(){
86     return region;
87   }
88   public String[] getDate(){
89     return datum;
90   }
91   public String[] getstartHafen(){
92     return startHafen;
93   }
94   public String[] getzielHafen(){
95     return zielHafen;
96   }
97 }
```

Quelltext 27.2: Kreuzfahrtbuchung: Web Service

27.7. Resümee

Die Aufgabe hat mir einen guten Überblick über das Thema Web-Services verschafft. Dadurch, dass die Implementierung einer lauffähigen Umbebung, des Clients samt GUI und des Web-Services selber gefordert waren kenne ich nun die Thematik aus allen Perspektiven. Die gute Vorbereitung durch Schulungen zu jedem Programm und Protokoll usw. die damit zusammenhängen konnte ich schnell Fehler auffinden und beseitigen. Bei schwierigeren Fällen war das Team ebenfalls sehr wichtige Stütze. Dabei war die größte Schwierigkeit eine lauffähige Umgebung zu schaffen, von da an ging alles, zusammen mit den Schulungen, wie von selbst.

Teil IV.

Projektdurchführung

28. Einleitung

Nach dem Abschluss des ersten Projektteils – der Schulungen sowie der praktischen Aufgaben – war es nun Zeit, mit der eigentlichen Projektarbeit zu beginnen. Die Mitglieder waren nun auf die anstehenden Aufgaben und ihre Schwierigkeiten vorbereitet. Die Mitglieder wurden in einzelnen Teilgruppen eingeteilt, die ähnlich den Schulungsgruppen aufgebaut waren.

Auf den folgenden Seiten ist das Resultat des Projekts beschrieben. Alle Teilgruppen haben ihre Arbeit sowie die geschaffenen Ergebnisse, aber auch Probleme, welche in der Entwicklung auftauchten, dokumentiert.

29. Datenbank

29.1. Einleitung

Das Projekt BizWeb des Fachbereichs Informatik/Wirtschaftsinformatik begann im Frühjahr 2004. Die Namensgebung erfolgte nach Abwandlung des Namens Business WebServices = BizWeb. Es beinhaltet die Web Service Entwicklung eines Geschäftsprozesses im Szenario der Kreuzfahrtbuchung. Die Datenbasis soll auf einem SAP-System verwaltet werden. Die Datenanbindung an die SAP Datenbank erfolgt über den SAP-Java-Connector.

Weitere standardisierte Techniken werden genutzt: XML, SOAP, WSDL, UDDI, Java, Apache Tomcat-Server, Java-Server-Page-Engine, AXIS, JAVA-Servlets, SAP-Java-Connector, ABAP, HTTP-Protokoll und TCP-Monitoring. Das Projekt wurde in zwei Phasen unterteilt die folgend erläutert werden.

29.1.1. Phase I

In der ersten Phase hat sich jedes Projektmitglied mit einem Thema auseinandersetzen müssen, welches im späteren Verlauf benötigt wurde. Die Wissensweitergabe jedes einzelnen erfolgte darauf durch Schulungen in der Gesamtgruppe.

Themengebiete der einzelnen Schulungen:

- Eclipse, Einführung in Eclipse
- XML Grundlagen Teil 1, Einführung in XML, Validierung (DTD, Schema), CSS, XSL
- XML Grundlagen (DTD, Schema, XSL, CSS, Parser, XPath, XQuery)
- XML Grundlagen Teil 2, XML Parser (SAX, DOM, JDOM)
- XML Grundlagen Teil 3, Abfragetools (XPath, XQuery)
- Tomcat und Servlets, Einführung in Tomcat und Servlets
- Java Server Pages, Einführung in JSP

- WSDL, Einführung in WSDL
- SOAP, Einführung in SOAP und AXIS
- UDDI, Einführung in UDDI

Nach Abschluss aller Schulungen sollte ein eigenständiger WebService erstellt werden.

Themengebiete der einzelnen WebServices:

- Taschenrechner
- Währungsrechner
- Flugbestätigung
- Wertpapierkurs
- Kreuzfahrtliste
- Hotelbuchung
- Textanalysierer
- Literatursuche
- Produktkatalog
- Stückliste
- Weltuhrzeit
- Kreuzfahrtbuchung (Test)

29.1.2. Phase II

In der zweiten Phase sollte die Realsierung des WebServices Kreuzfahrtbuchungen mit allenTeilprozessen erfolgen. Hierbei wurde das Projekt in einzelne Teilaufgaben geteilt. Die Projektmitglieder arbeiteten in kleinen Gruppen zusammen.

Themengbiete der einzelnen Teilprojekte:

- Datenbankmodell SAP
- SAP-Java-Connector
- WebService „Kreuzfahrtbuchung“ und Servlet

- Mehrstufigkeit, Intermediaries
- Cocoon-Portal

Unsere Aufgabe in diesem Gesamtprojekt ist der Entwurf und die Integration eines Datenbankmodells für das Projektszenario Kreuzfahrtbuchung in das SAP System. Weiterhin müssen auch Vorkehrungen für die Datensicherheit vorgenommen werden z. B. eine Datensicherung.

Unser Teilprojekt „der Entwurf des Datenbankmodells“ legt den Grundstein für alle weiteren Teilbereiche des BizWeb Projekts. Alle Teilprojekte haben einen engen Zusammenhang, wobei die Integration der Datenbank am meisten mit dem „SAP-Java-Connector“ und dem „WebService „Kreuzfahrtbuchung“ und Servlet“ im Zusammenhang stand.

29.2. Aufgabenstellung

Die Aufgabenstellung unseres Teilprojekts ist der Entwurf und die Abbildung eines Datenbankmodells im Szenario Kreuzfahrtbuchung auf das SAP System.

Der Datenbankentwurf sollte die folgenden gewünschten Merkmale, die später auf der GUI Oberfläche dargestellt werden, enthalten:

Daten (nach außen sichtbare Felder):

- Reise-Nr.
- Anbieter
- Schiff
- Zeit Abfahrt (ohne Zwischenstationen)
- Zeit Ankunft (ohne Zwischenstationen)
- Startort
- Zielort
- Fahrbereich/Region
- Art
- Zwei Preisklassen (Innenkabine / Außenkabine)

Aus diesen groben Anforderungen soll ein Modell in der 3. Normalform entworfen werden. Alle weiteren notwendigen Bestandteile und das Wissen dazu müssen von uns selbst erarbeitet werden.

Funktionen – Liste nach unterschiedlichen Kriterien sortiert aufrufen (Selektionen):

- Schiff
- Datum von
- Datum von/bis
- Preisklasse 1 (aufsteigend)
- Preisklasse 2 (aufsteigend)
- Fahrbereich/Region (nach Region)
- Kapazität Innenkabinen (frei/belegt)
- Kapazität Außenkabinen (frei/belegt)
- Abrufen bestimmter Kreuzfahrten (bspw. nur Kreuzfahrten, die unter 1000,- € kosten)

Die Funktionen / Methoden für die Sortierung nach den einzelnen Kriterien müssen vom Service realisiert werden. Als Basis müssen wir dazu die passenden Views / Ansichten der Stammdaten bereitstellen.

29.3. Entwurf des Datenbankmodells

29.3.1. Vorgehensweise

In den ersten Schritten haben wir den Zweck der Datenbank überlegt. Wodurch wir per Mindmapping alle zugehörigen Inhalte und Aufgaben der Datenbank festgehalten haben. Für den Entwurf unseres Datenbankmodells haben wir die UML[1] zur Hilfe genommen. Die Modell Entwürfe der Tabelle 29.1 sowie der Abbildung 29.1 sind mit den den Power Designer von Sybase erstellt worden.

Es wurde hierbei versucht die Realität in ein konzeptionellen und das daraus resultierende physikalischen Modell abzubilden. Dieser erste Entwurf des Datenbankmodells diente

[1] Unified Modeling Language

als Grundlage zur Diskussion mit den Projektmitgliedern, um Verständnisprobleme oder fehlende Daten bzw. Beziehungen zu ermitteln.

Im weiteren Verlauf wurde mit den Projektmitgliedern das Datenbankmodell für alle relevanten Geschäftsobjekte weiter verfeinert.

29.3.2. Erstellung des ConceptualModels

Aufbau der Tabellen

Das Datenbankmodell beinhaltet neun Klassen (Objekttypen), welche nur die für unser Projekt bedeutsamsten Attribute/ Informationen enhalten. Dieses bedeutet, dass die Datenbank weiter ausbaufähig wäre, wir aber nur die Grundzüge für die Umsetzung der Kreuzfahrtbuchung berücksichtigt haben.

Die erste Tabelle ist die Tabelle „**Kreditkarten**“ – *zbw_kreditkarten* (schräggestellt: Umsetzung in der SAP-Datenbank): Diese beinhaltet den Primärschlüssel „KreditkartenID“ – *zbw_kkid* und die Attribute „Kreditkarten Pruefnummer“ – *zbw_kkpruefnummer*, „KundenID“ – *zbw_kundenid*, „Kreditkarten Herausgeber“ – *zbw_kkherausgeber*, „Kreditkarten Gueltigkeit“ – *zbw_kkgueltigkeit*.

Die Tabelle „**Kunden**“ – *zbw_kunden* (schräggestellt: Umsetzung in der SAP-Datenbank): Diese beinhaltet den Primärschlüssel „KundenID“ – *zbw_kundenid* und die Attribute „Kundenname“ – *zbw_kname*, „Kundenvorname“ – *zbw_kvorname*, „Geburtsdatum des Kunden“ – *zbw_kgebdatum*, „Strasse und Hausnummer des Kunden“ – *zbw_kstrasse*, „PLZ des Kunden“ – *zbw_kplz*, „Wohnort des Kunden“ – *zbw_kort*, „Heimatland des Kunden“ - *zbw_land*, „Passwort des Kunden“ – *zbw_kpasswort*.

Die Tabelle „**Schiffe**“ – *zbw_schiffe* (schräggestellt: Umsetzung in der SAP-Datenbank): Diese beinhaltet den Primärschlüssel „SchiffsID“ – *zbw_schiffsid* und das Attribut „Schiffsname“ – *zbw_schiffsname*.

Die Tabelle „**Kabinentypen**“ – *zbw_kabinentypen* (schräggestellt: Umsetzung in der SAP-Datenbank): Diese beinhaltet den Primärschlüssel „KabinenID“ – *zbw_kabinenid* und die Attribute „Kabinenname“ – *zbw_kabinenname* „Kabinenpreis“ – *zbw_kabinenpreis* „Standardwaehrung“ – *mwaer*.

Die Tabelle „**Schiffskabinen**“ – *zbw_skabinen* (schräggestellt: Umsetzung in der SAP-Datenbank): Diese beinhaltet die Primärschlüssel „KabinenID“ – *zbw_kabinenid* und „SchiffsID“ – *zbw_Schiffsid* mit dem Attribut „Kabinenanzahl“ – *zbw_kabinenanzahl*.

Die Tabelle „**Anbieter**“ – *zbw_anbieter* (schräggestellt: Umsetzung in SAP-Datenbank): Diese beinhaltet den Primärschlüssel „AnbieterID“ – *zbw_anbieterid* und die Attribute „Name des Anbieter“ – *zbw_aname*, „Strasse und Hausnummer des Anbieters“ – *zbw_astrasse*, „PLZ des Anbieters“ – *zbw_aplz*, „Ort des Anbieters“ – *zbw_aort*, „Tele-

fon des Anbieters“ – *zbw_atelefon*, „Email des Anbieters“ – *zbw_aemail*, „Webseite des Anbieters“ – *zbw_awebseite*.

Die Tabelle „**Regionen**“ – *zbw_regionen* (schräggestellt: Umsetzung in SAP-Datenbank): Diese beinhaltet den Primärschlüssel „RegionID“ – *zbw_regionid* mit dem Attribut „Name der Region“ – *zbw_regionname*.

Die Tabelle „**Reisen**“ – *zbw_reisen* (schräggestellt: Umsetzung in SAP-Datenbank): Diese beinhaltet den Primärschlüssel „ReiseID“ – *zbw_reiseid* und die Attribute „AnbieterID“ – *zbw_anbieterid*, „RegionID“ – *zbw_regionid*, „SchiffsID“ – *zbw_schiffsid*, „Beginn der Reise“ – *zbw_reiseabfahrt*, „Ende der Reise“ – *zbw_reiseankunft*, „Basishafen“ – *zbw_basishafen*, „Zielhafen“ – *zbw_zielhafen*.

Die Tabelle „**Buchung**“ – *zbw_buchung* (Kursivdruck: Umsetzung in SAP-Datenbank): Diese beinhaltet den Primärschlüssel „BuchungsID“ – *zbw_buchungsid* und die Attribute „KundenID“ – *zbw_kundenid*, „ReiseID“ – *zbw_reiseid*, „Buchungsdatum“ – *zbw_buchungsdatum*.

Im folgenden eine Übersicht in tabellarischer Form der Inhalte der einzelnen Objekttypen der Datenbank.

Objekttyp	**Attribute**
Kreditkarten	zbw_kkid, zbw_kkpruefnummer, zbw_kundenid, zbw_kkherausgeber, zbw_kkgueltigkeit
Kunden	zbw_kundenid, zbw_kname, zbw_kvorname, zbw_kgebdatum, zbw_kstrasse, zbw_kplz, zbw_kort, zbw_land, zbw_kpasswort
Schiffe	zbw_schiffsid, zbw_schiffsname
Kabinentypen	zbw_kabinenid, zbw_kabinenname, zbw_kabinenpreis, mwaer
Skabinen	zbw_kabinenid, zbw_Schiffsid, zbw_kabinenanzahl
Anbieter	zbw_anbieterid, zbw_aname, zbw_astrasse, zbw_aplz, zbw_aort, zbw_atelefon, zbw_aemail, zbw_awebseite
Regionen	zbw_regionid, zbw_regionname
Reisen	zbw_reiseid, zbw_anbieterid, zbw_regionid, zbw_schiffsid, zbw_reiseabfahrt, zbw_reiseankunft, zbw_basishafen, zbw_zielhafen
Buchung	zbw_buchungsid, zbw_kundenid, zbw_reiseid, zbw_buchungsdatum

Tabelle 29.1.: Datenbank: Objekte und Attribute

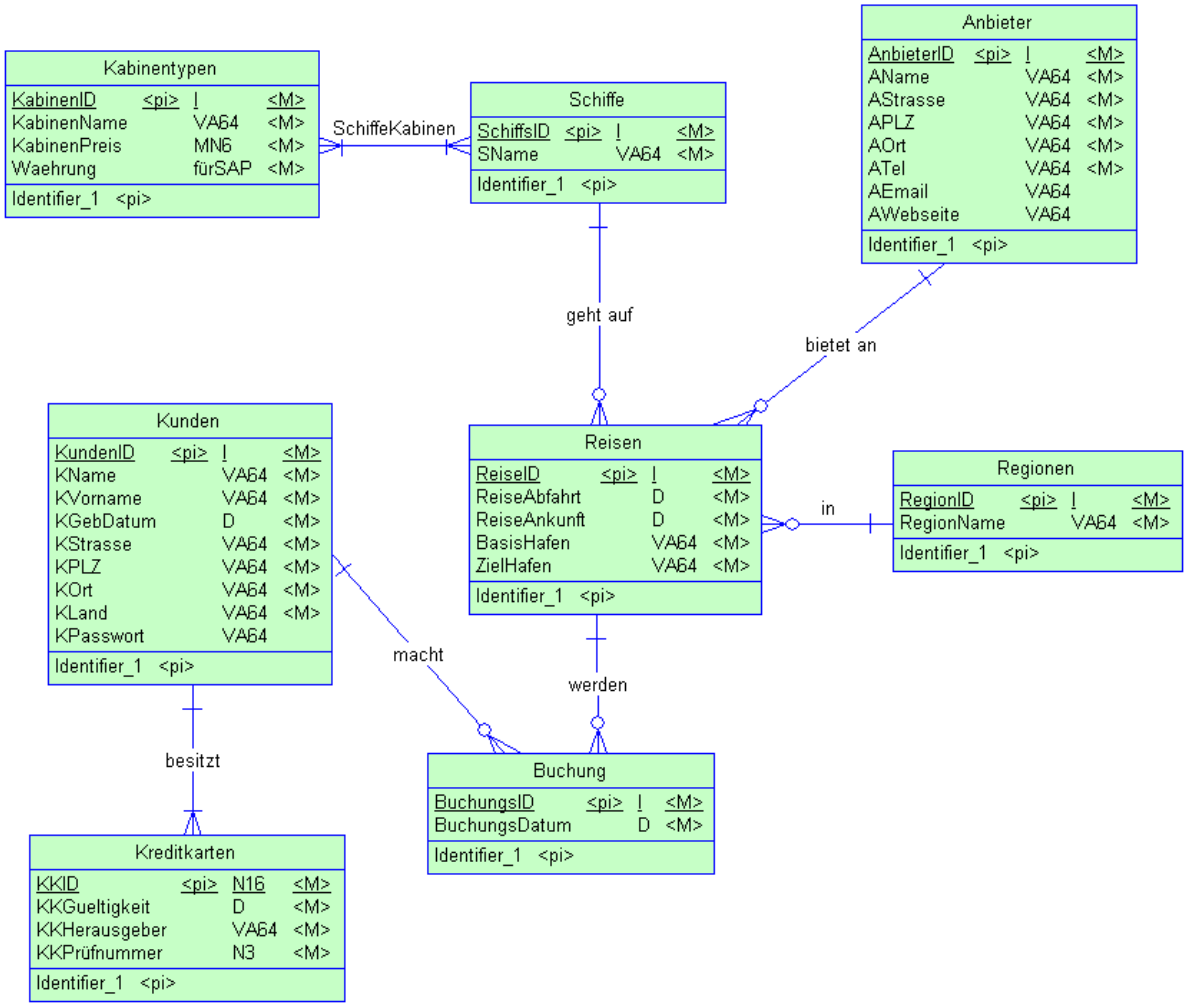


Abbildung 29.1.: Datenbank: ConceptualModel

29.3.3. Erstellung des PhysicalModels

Aufbau der Tabellen

Wie in Abbildung 29.2 zu sehen, ist das PhysicalModel ähnlich dem ConceptualModel. Das PhysicalModel stellt die Datenbankstruktur in ihrer Endversion dar.

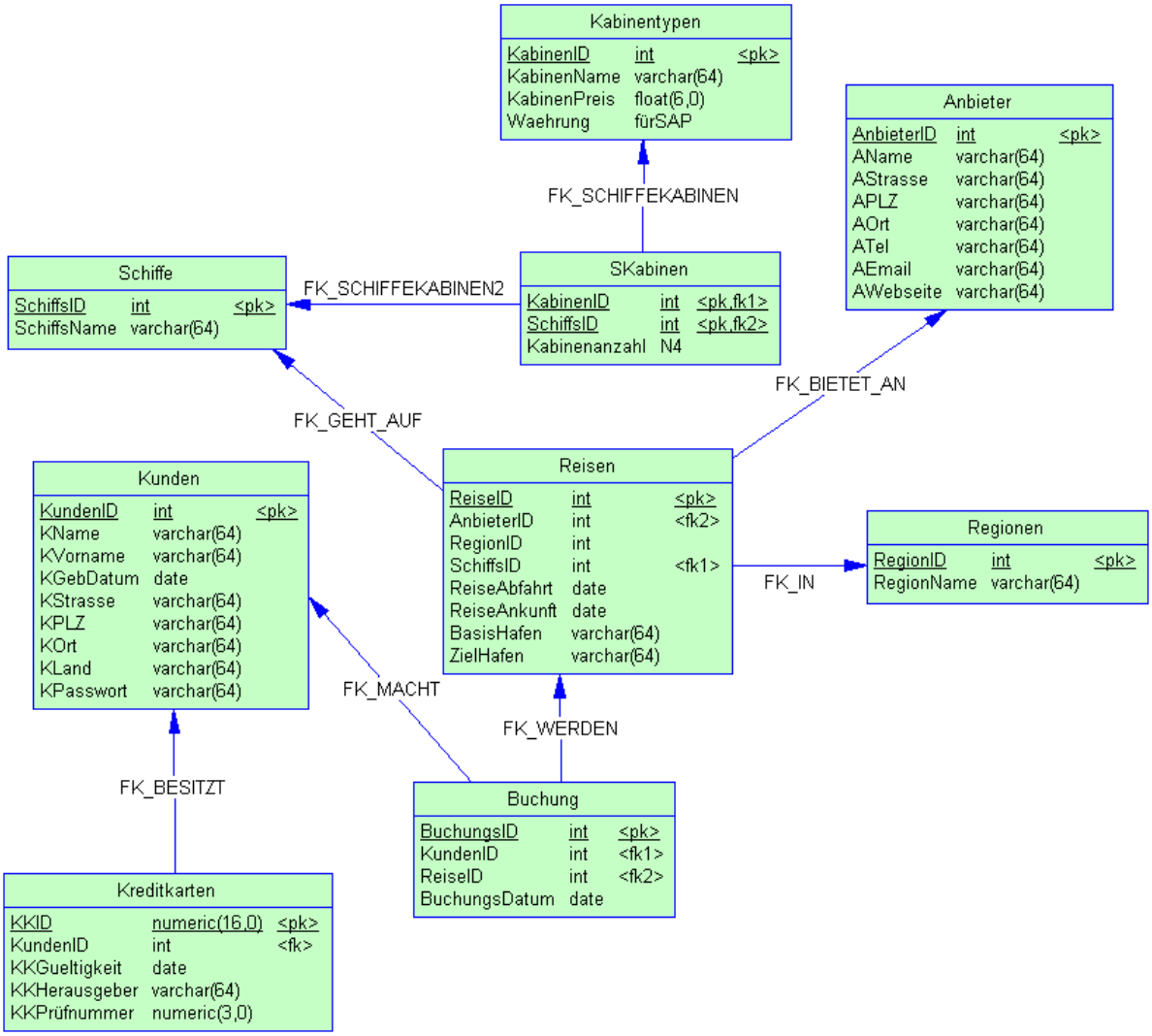


Abbildung 29.2.: Datenbank: PhysicalModel

29.3.4. Beziehungen

- 0 - optional (ein oder kein Objekt)
- 1 - genau ein Objekt
- * - beliebig viele Objekte, mindestens eins

Folgend eine Tabelle über alle Beziehungen des Datenbankmodells:

Beziehungen	
Eine oder beliebig viele Kreditkarten besitzt genau ein Kunde Kreditkarten - Kunden	*..1
Genau ein Kunde besitzt eine oder mehrere Kreditkarten Kunden - Kreditkarten	1..*
Genau ein Kunde macht keine, eine oder mehrere Buchungen Kunden - Buchung	1 : 0..*
Genau ein Schiff geht auf keine, eine oder mehrere Reisen Schiffe - Reisen	1 : 0..*
Ein oder mehrere Schiffe haben ein oder mehrere Kabinentypen Schiffe - Kabinentypen	*..*
Ein oder mehrere Kabinentypen sind auf einem oder mehreren Schiffen vorhanden Kabinentypen - Schiffe	*..*
Genau ein Anbieter bietet keine, eine oder mehrere Reisen an Anbieter - Reisen	1 : 0..*
Genau eine Region besitzt keine, eine oder mehrere Reisen Regionen - Reisen	1 : 0..*
Genau eine Reise besitzt keine, eine oder mehrere Buchungen Reisen - Buchung	1 : 0..*
Keine, eine oder mehrere Reisen besitzen genau eine Region Reisen - Regionen	0..* : 1
Keine, eine oder mehrere Reisen bietet genau ein Anbieter an Reisen - Anbieter	0..* : 1
Keine, eine oder mehrere Reisen besitzen genau ein Schiff Reisen - Schiffe	0..* : 1
Keine, eine oder mehrere Buchungen macht genau ein Kunde Buchung - Kunde	0..* : 1
Keine, eine oder mehrere Buchungen besitzen genau eine Reise Buchung - Reise	0..* : 1

Tabelle 29.2.: Datenbank: Beziehungen

29.4. Realisierung in SAP

Die Realisierung des Modells in SAP weicht etwas vom Datenbank Entwurf in Sybase PowerDesigner ab, da es in SAP spezielle Konventionen gibt die zwingend eingehalten werden müssen. So muss bspw. beim Preis der Datentyp „CURR“ (Währung) verwendet werden, dabei muss zwingend in derselben Tabelle ein Währungsfeld namens „Waehrung“ dem Feldtyp „Mwaer“ und dem daraus resultierenden Datentyp „CUKY“ angelegt werden. Des Weiteren gibt es in SAP einen so genannten „Kundennamensraum“, der

dem Kunden vorschreibt, dass die eigenen Entwicklungen mit Y oder Z beginnen müssen. Daher beginnen unsere Tabellen-, Datenelement- und Domänennamen mit „ZBW_“. Weiterhin kennt SAP nur den Datentyp „NUMC“ statt Integer.

29.5. Integration des Datenbank Szenarios Kreuzfahrtbuchung in das SAP System

Um in SAP etwas programmieren zu dürfen benötig man eine Entwicklungsklasse, um diese anzulegen wird ein Auftrag benötigt.

Unsere Entwicklungsklasse: ZB_BIZWEB
Unser Auftrag: BHVK900440

29.5.1. Starten von SAP R/3

Wir starten SAP/R3 mit den Logon Screen, indem wir den „Logon“ Button bestätigen (Abbildung 29.3).

Abbildung 29.3.: Datenbank: SAP R/3 Logon – Start

Folgend erscheint der Startbildschirm. Hier müssen Mandant: „`312`“, Benutzername: „`bizweb`“ und Kennwort eingegeben werden (Abbildung 29.4).

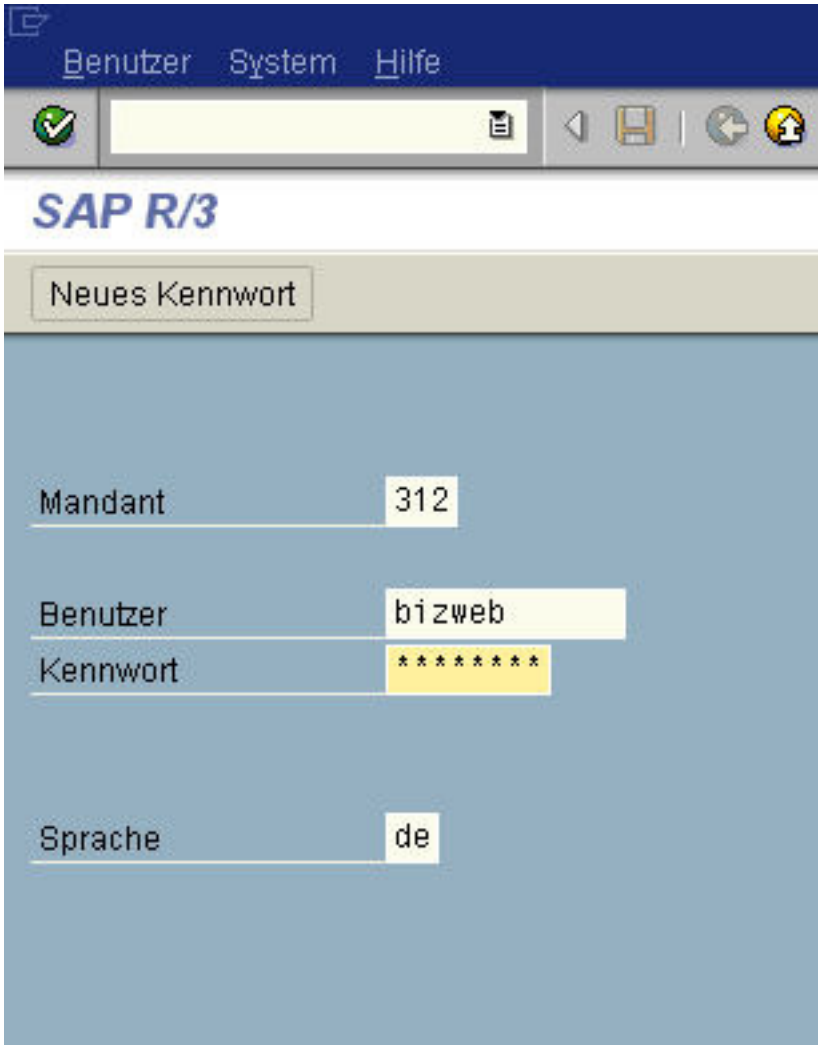


Abbildung 29.4.: Datenbank: SAP R/3 einloggen auf Mandant

29.5.2. Anlegen der Entwicklungsklasse ZB_BIZWEB

Zu Beginn benötigen wir eine Entwicklungsklasse. Diese erstellen wir im Modul SE80 Object Navigator, wobei wir die Namenskonventionen beachten müssen. Alle eigenen Namen müssen hierbei mit Y oder Z beginnen. Unsere Entwicklungsklasse besitzt den Namen: ZB_BIZWEB.

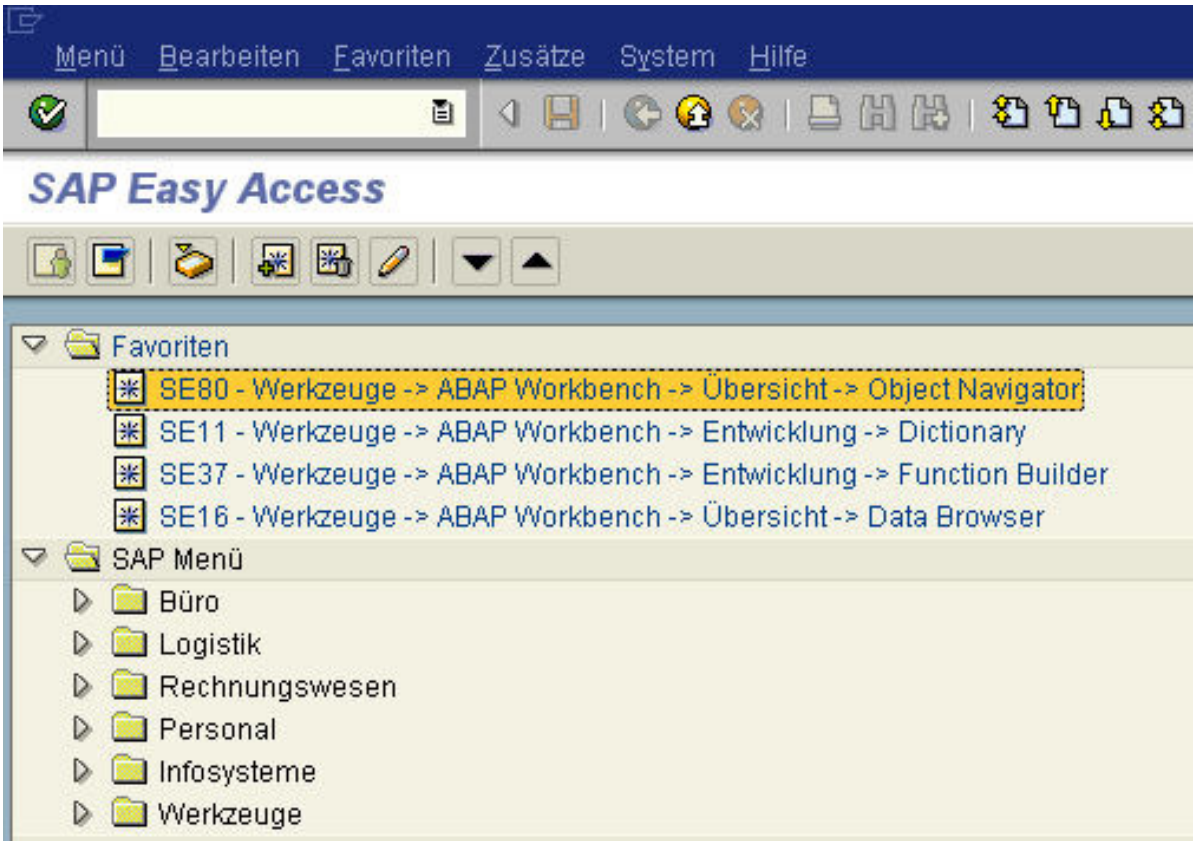


Abbildung 29.5.: Datenbank: Übersicht SAP Start Menü

Falls die SE80 in den Favoriten nicht vorhanden ist, wird im weissen Suchfeld „n/SE80“ eingegeben. Dies ist eine weitere Möglichkeit um die Entwicklungsklasse SE80 zu öffnen (Abbildung 29.5).

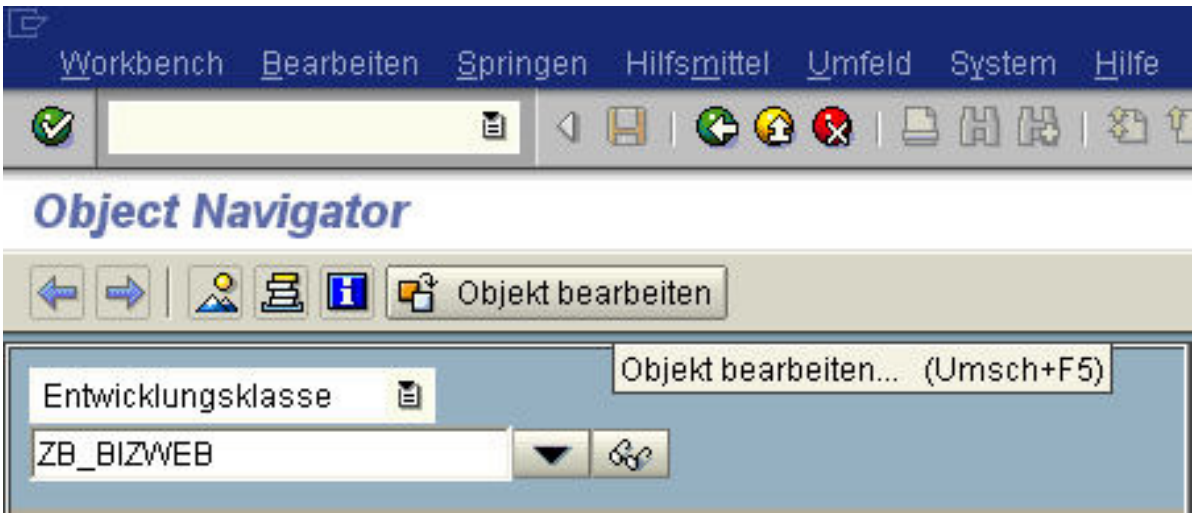


Abbildung 29.6.: Datenbank: Anlegen einer Entwicklungsklasse SE80

Wählen Sie in der Objektliste den Menüpunkt „Entwicklungsklasse“ aus und geben im darunterstehenden Feld Ihren gewüschten Namen ein. Daraufhin erscheint nach Bestätigung einer Meldung ob die Entwicklungsklasse angelegt werden soll, falls sie noch nicht vorhanden ist (Abbildung 29.6).

Wir erstellen die Entwicklungsklasse indem wir auf den Button „Objekt bearbeiten“ klicken. Daraufhin erscheint ein neues Fenster, wo in der obersten Zeile der Radio-Button Entwicklungsklasse ausgewählt wird und im nebenstehenden Feld nun unser gewählter Name „ZB_BIZWEB“ eingetragen werden kann (Abbildung 29.7).

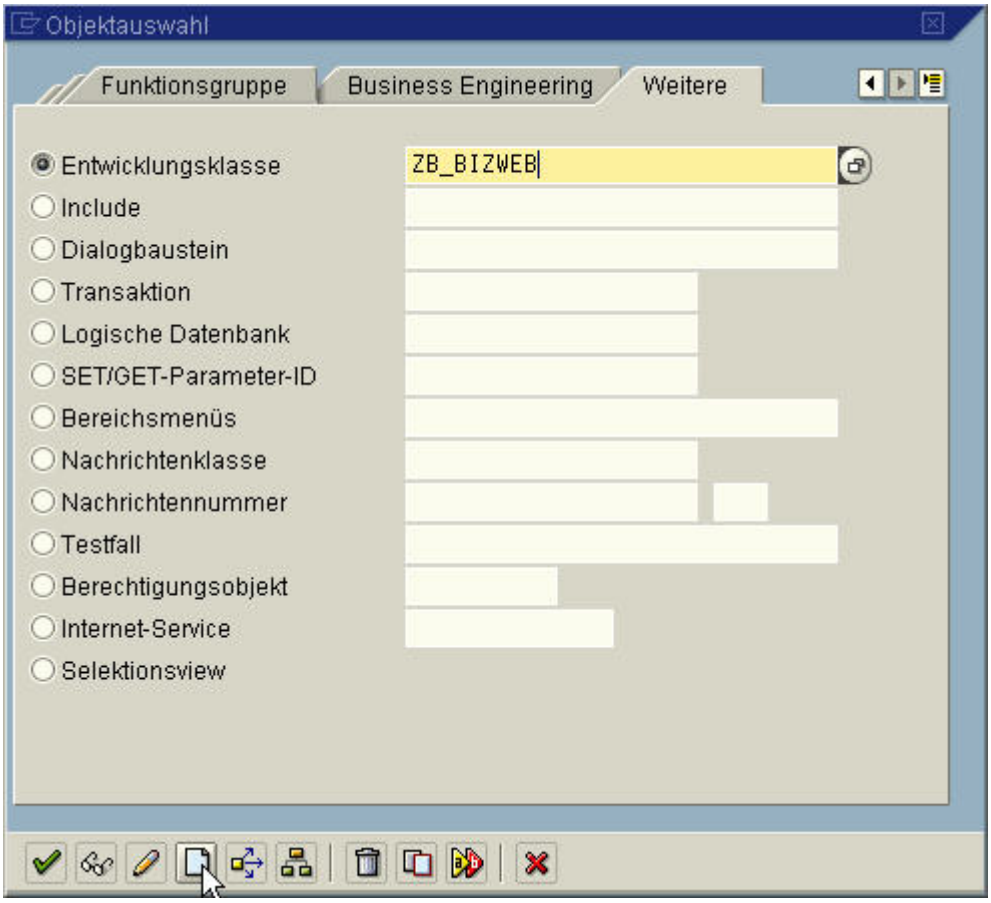


Abbildung 29.7.: Datenbank: Neuanlegung der Entwicklungsklasse mit Namensvergabe

Durch betätigen des Symbols „anlegen“ erscheint ein neues Fenster (Abbildung 29.7). Die Entwicklungsklasse kann durch klicken auf das Diskettensymbol gesichert werden (Abbildung 29.8).

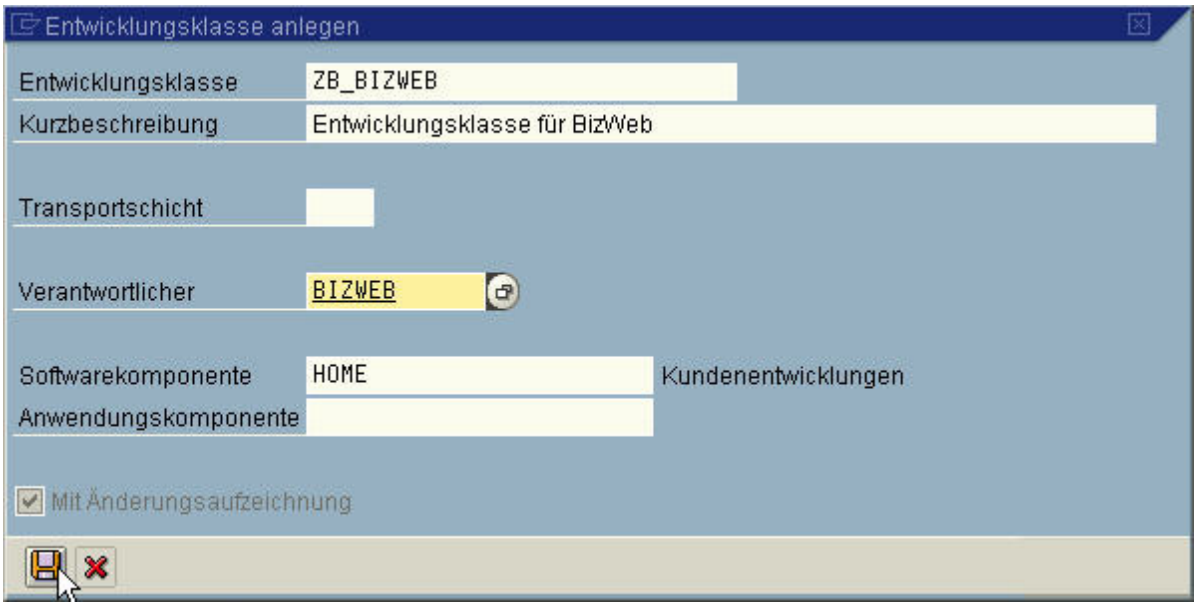


Abbildung 29.8.: Datenbank: Neuanlegung der Entwicklungsklasse

Zum weiteren Vorgehen benötigen wir einen Workbench Auftrag (Abbildung 29.9).

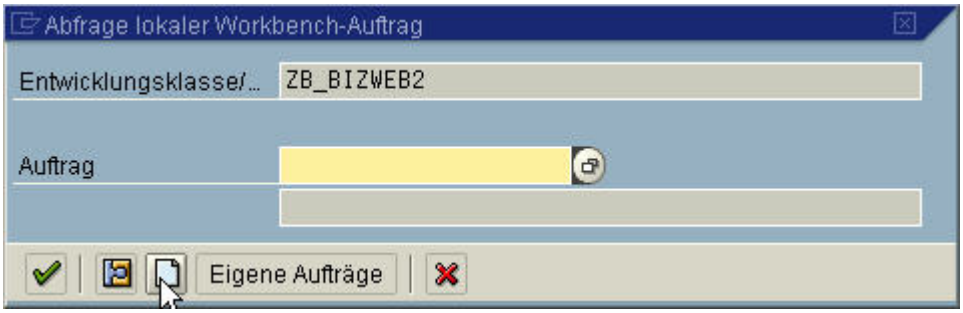


Abbildung 29.9.: Datenbank: Anlegen eines Workbench-Auftrags

Durch betätigen des Symbols „anlegen“ gelangen wir in den Modus „Auftrag anlegen“. Unter Kurzbeschreibung kann eine Bezeichnung der Auftrags erfolgen. Worauf nun der Workbench-Auftrag mit dem Diskettensymbol abgespeichert werden kann (Abbildung 29.10).

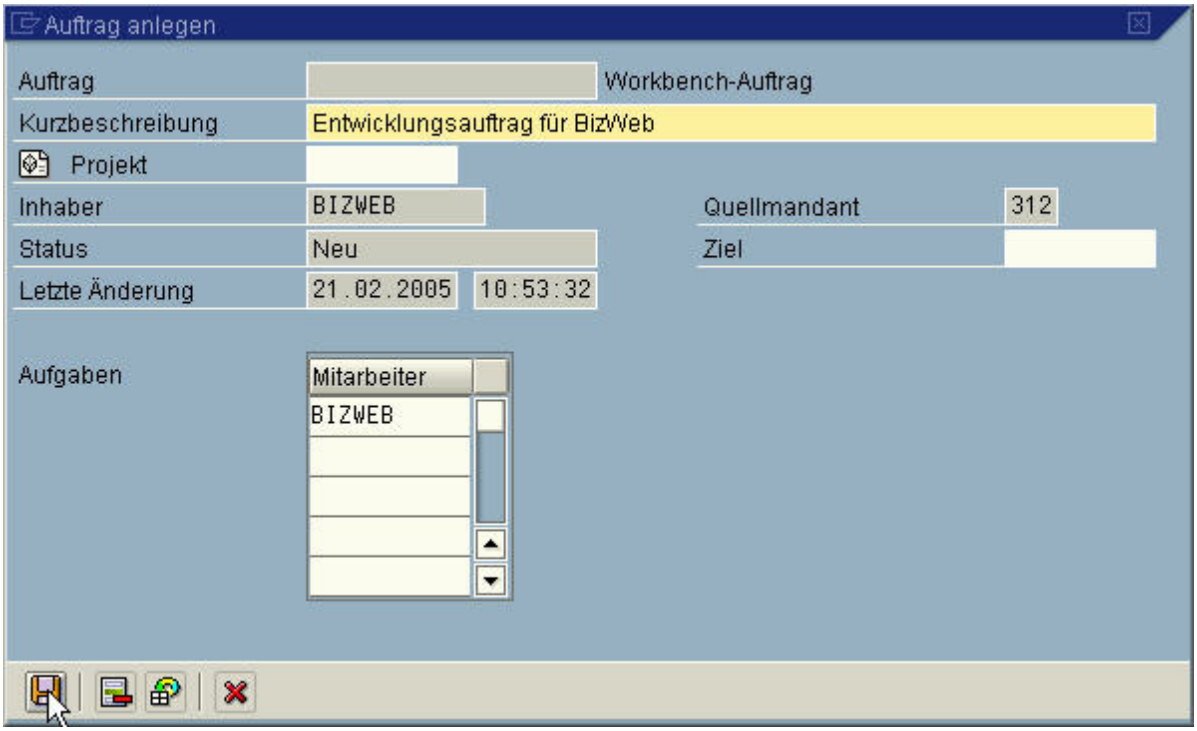


Abbildung 29.10.: Datenbank: Anlegen eines Workbench-Auftrags - Eigenschaften

Nach der Abspeicherung erscheint die Auftragsnummer, in unserem Beispiel „BHVK9004 40“. Zum Abschluss wird das Häckchen-Symbol „Weiter“ geklickt (Abbildung 29.11).

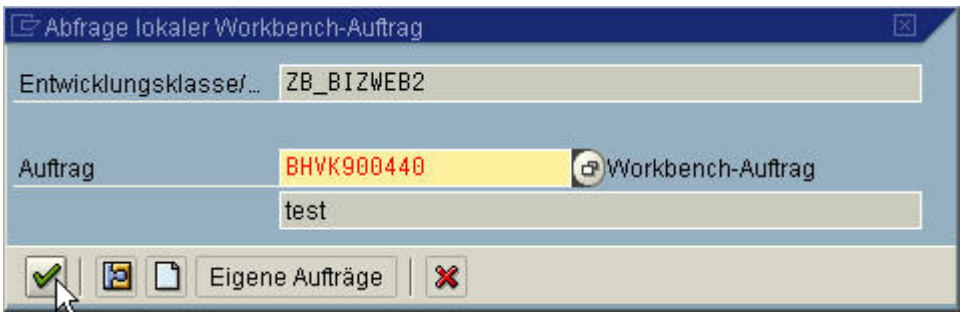


Abbildung 29.11.: Datenbank: Anlegen eines Workbench-Auftrags - Systemnummernvergabe

29.5.3. Anlegen der Datenbanktabellen

In SAP R/3 gibt es drei Grundtypen von Tabellen, wobei die „Pool-“ und „Clustertabellen“ allgemein nicht mehr verwendet werden. Der Standard ist hierbei die transparente Tabelle. Die transparente Tabelle ist eine eins-zu-eins-Abbildung von der tatsächlichen Datenbank in SAP R/3.

Im Folgenden eine Übersicht der einzelnen Dictionary Elemente, die in SAP R/3 definiert sind (Abbildung 29.12).

Dict. Element	Beschreibung
Datenbanktabelle	transparente Tabellen, Pool- und Clustertabellen
View	Sicht auf mehrere Tabellen
Datentyp	Datenelemente, Strukturen, Tabellentypen
Domäne	technische Beschreibung eines Datenelements
Suchhilfe	Wertehilfen für Datenelemente und Tabellen
Sperrobjekt	Funktionsbausteine: Ent-/Sperren von Tabellen

Abbildung 29.12.: Datenbank: Auflistung der Dictionary Elemente (ABAP Skript Alfred Schmidt)

Wir befinden uns im Modul SE80 Object Navigator. Unsere Entwicklungsklasse ist bereits angelegt und ausgewählt. Zum Anlegen der Tabellen wählen Sie mit Rechtsklick den Namen der Entwicklingsklasse an und wählen Sie folgenden Menüpfad: „Anlegen“ → „DDIC-Object“ → „Datenbanktabellen“ (Abbildung 29.13).

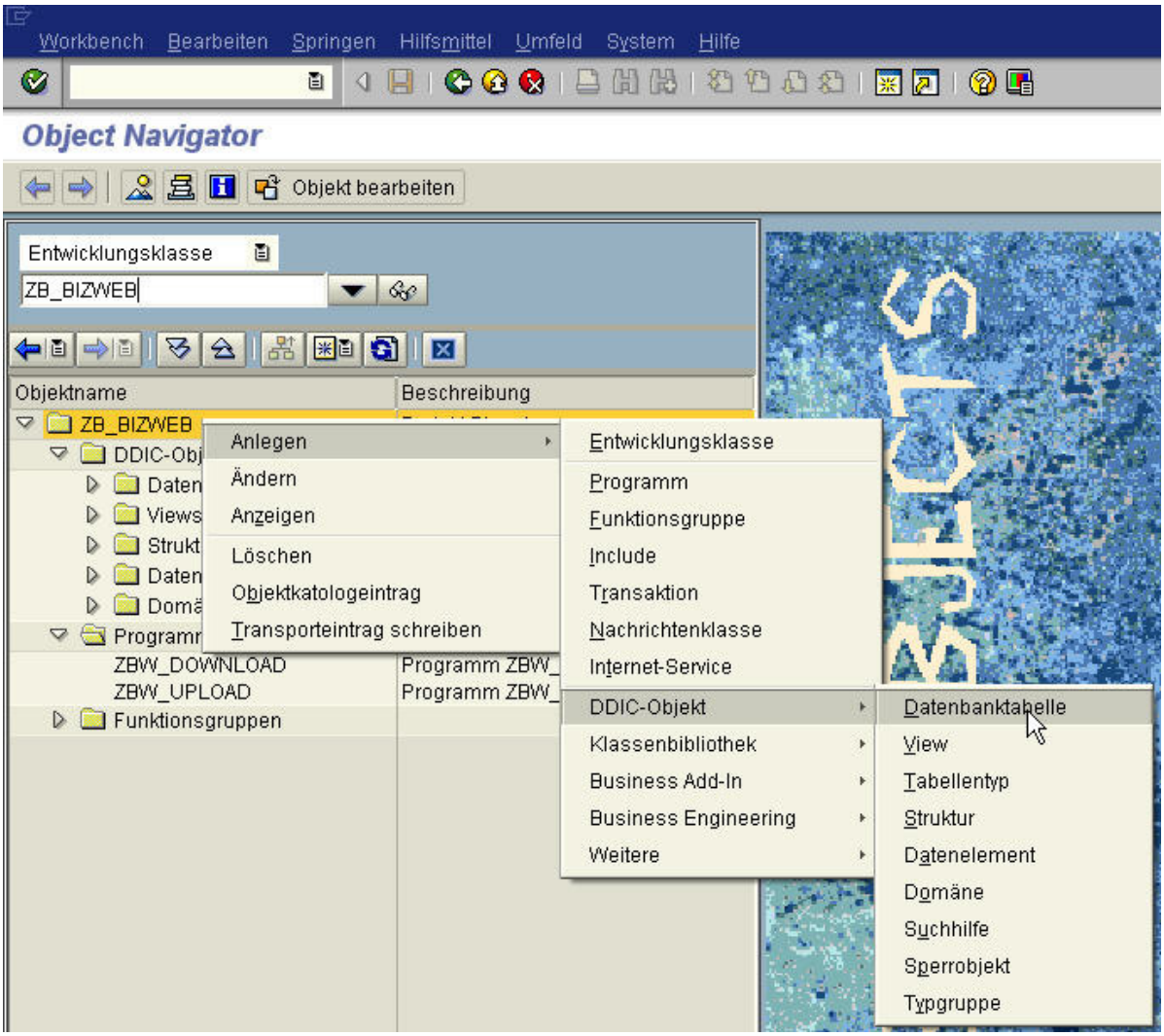


Abbildung 29.13.: Datenbank: Anlegen der Datenbanktabellen

Im nachfolgenden Fenster kann nun ein Name eingegeben werden, wobei die Namenskonventionen beachtet werden müssen (eine genauere Beschreibung der Namenskonventionen befinden sich im Kapitel 29.5.2: Anlegen der Entwicklungsklasse ZB_BIZWEB). Wir vergeben für unsere Kundentabelle den Namen „ZBW_KUNDEN“ (Abbildung 29.14).

Abbildung 29.14.: Datenbank: Anlegen einer Tabelle mit Namenvergabe

Durch Klicken des Häckchen-Symbols „Weiter“ müssen nun die Eigenschaften der Kundentabelle festgelegt werden (Abbildung 29.14). Es müssen definiert werden: „Kurzbeschreibung“, „Auslieferungsklasse“ und ob die Tabellenpflege erlaubt sein soll oder nicht.

Auslieferungsklasse: „A“ (Anwendungstabelle (Stamm- und Bewegungsdaten)) (Abbildung 29.15)

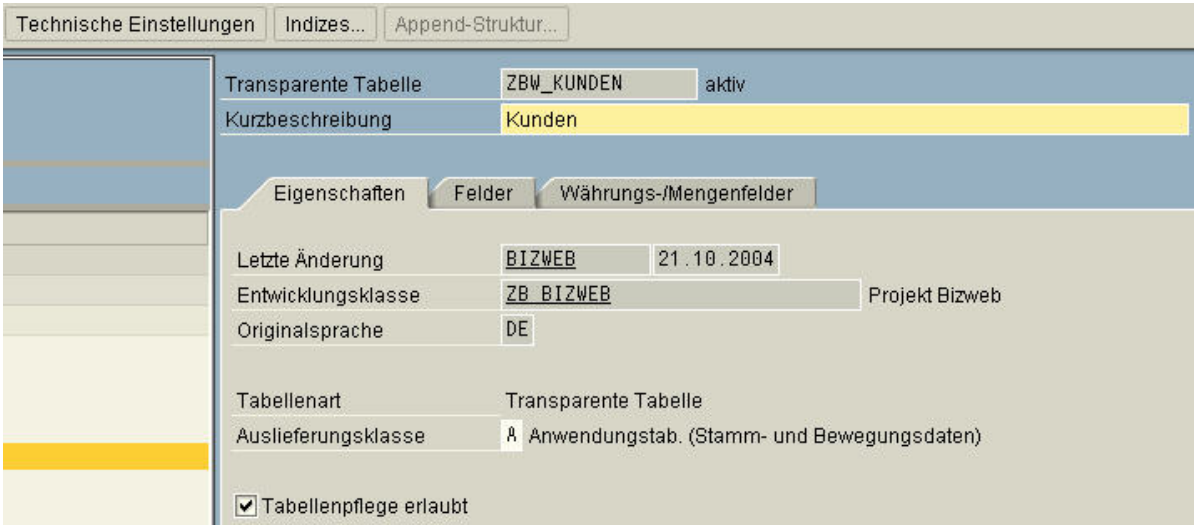


Abbildung 29.15.: Datenbank: Eigenschaften von Datenbanktabellen

Nach dem Speichern der Eigenschaften müssen nun die Technischen Einstellungen gepflegt werden. Es müssen definiert werden „Datenart“, „Größenkategorie“ und die „Pufferung“ (Abbildung 29.16).
Datenart: „APPL0“ (Stammdaten, transparente Tabellen)
Größenkategorie: „0“ (Erwartete Datensätze 0 bis 1.400)
Die Pufferung muss mit den Radio-Button „nicht erlaubt“ belegt werden, da kein großes Datenaufkommen erwartet wird.

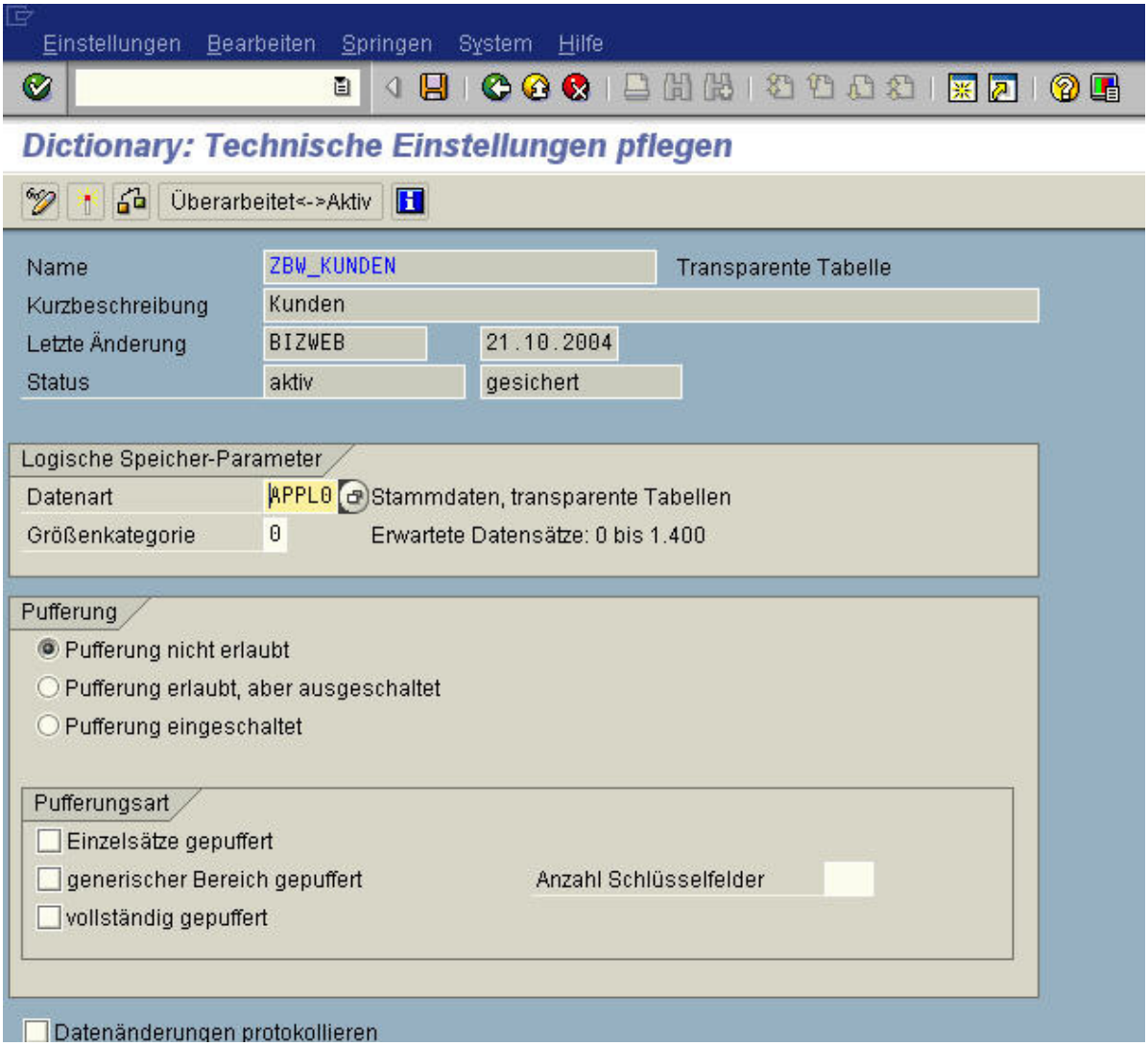


Abbildung 29.16.: Datenbank: Technische Einstellungen einer Datenbanktabelle

Nach Abspeicherung und „zurück“ erhalten wir unter dem Reiter „Felder“ die Eingabefelder für die einzelnen Attribute der Tabelle. Der Begriff „Mandt“ für Mandant muss immer mitangelegt werden. Dieser Schlüssel beschreibt die Abhängigkeit zum Mandaten. In unserem Fall Mandant 312 (Mandant = Client).

Die Spalte "Key“ beschreibt den „Primary Key“. Der „Primary Key“ ist der Schlüssel der die Tabelle eindeutig identifiziert und wovon jedes Attribut abhängig ist.

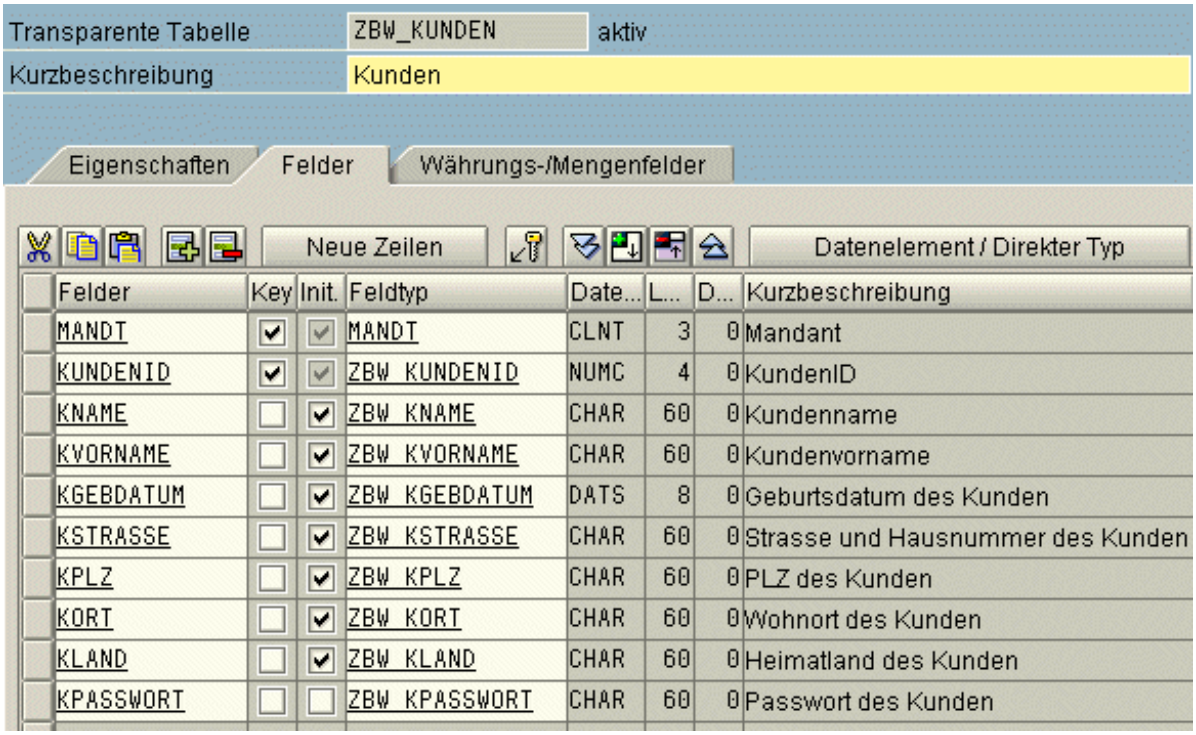
Durch Auswahl von „Init.“ (initialisieren) wird die Eingabe der Attribute zu einer Pflichteingabe (Abbildung 29.17).

Folgende Reihenfolge sollte beachtet werden:

- 1. Zeile: Setzung von „MANDT“
- 2. Zeile: Setzung des „Primary Key“ (oder mehrere)
- ff. Zeilen: Setzung von weiteren Attributen

Vorgehensweise zur Erstellung der Attribute:

1. In die Spalte „Felder“ den Attributnamen eintragen
2. „Key“ und „Init.“ festlegen
3. In „Feldtyp“ den Datenelementnamen eintragen



Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	☑	☑	MANDT	CLNT	3	0	Mandant
KUNDENID	☑	☑	ZBW_KUNDENID	NUMC	4	0	KundenID
KNAME	☐	☑	ZBW_KNAME	CHAR	60	0	Kundenname
KVORNAME	☐	☑	ZBW_KVORNAME	CHAR	60	0	Kundenvorname
KGEBDATUM	☐	☑	ZBW_KGEBDATUM	DATS	8	0	Geburtsdatum des Kunden
KSTRASSE	☐	☑	ZBW_KSTRASSE	CHAR	60	0	Strasse und Hausnummer des Kunden
KPLZ	☐	☑	ZBW_KPLZ	CHAR	60	0	PLZ des Kunden
KORT	☐	☑	ZBW_KORT	CHAR	60	0	Wohnort des Kunden
KLAND	☐	☑	ZBW_KLAND	CHAR	60	0	Heimatland des Kunden
KPASSWORT	☐	☐	ZBW_KPASSWORT	CHAR	60	0	Passwort des Kunden

Abbildung 29.17.: Datenbank: Technische Einstellungen einer Datenbanktabelle

29.5.4. Erstellung von Datenelementen und Domänen

Das Datenelement ist die semantische Beschreibung eines Feldes oder einer Komponente. Jedes Datenelement ist an einer Domäne gekoppelt. Die Domäne enthält die technische Beschreibung des Datenelements. SAP nennt das ganze zweistufiges Domänenkonzept (Abbildung 29.18).

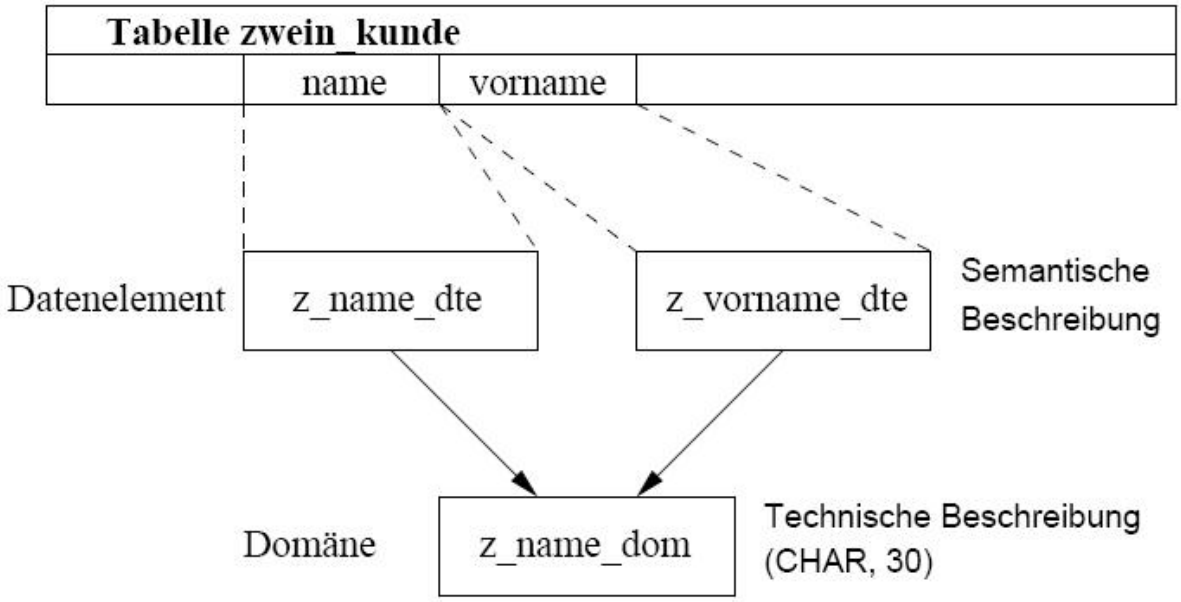


Abbildung 29.18.: Datenbank: Zweistufiges Domänenkonzept SAP R/3 (ABAP Skript, Alfred Schmidt)

Falls das Datenelement nicht vorhanden ist kann man mit einen Doppelklick auf den Namen das Datenelement angelegt werden (Abbildung 29.19).

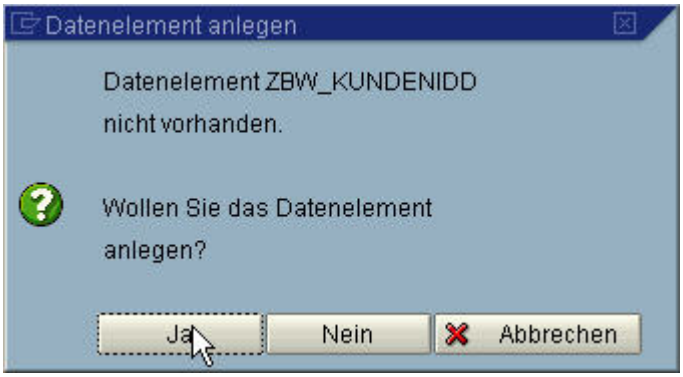


Abbildung 29.19.: Datenbank: Datenelemente anlegen

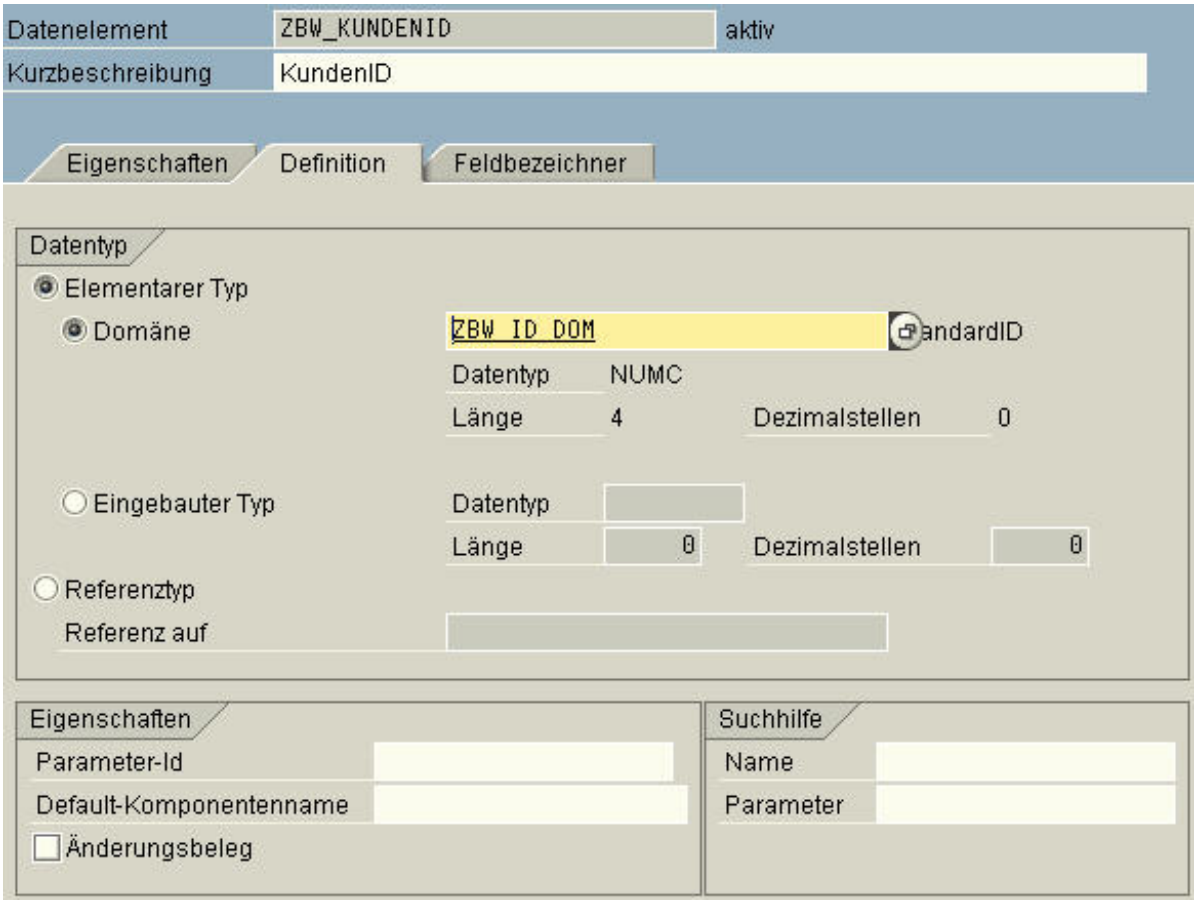


Abbildung 29.20.: Datenbank: Datenelemente pflegen

Wir befinden uns im „Datenelement pflegen“. Nun kann entschieden werden ob eine Domäne oder ein eingebauter Typ verwendet werden soll (Abbildung 29.20). Ist die Domäne noch nicht vorhanden kann durch Doppelklick auf den Domänennamen eine neue Domäne angelegt werden (Abbildung 29.21).

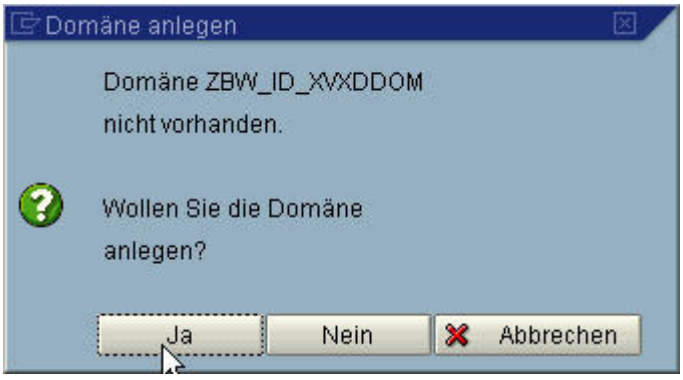


Abbildung 29.21.: Datenbank: Domänen anlegen

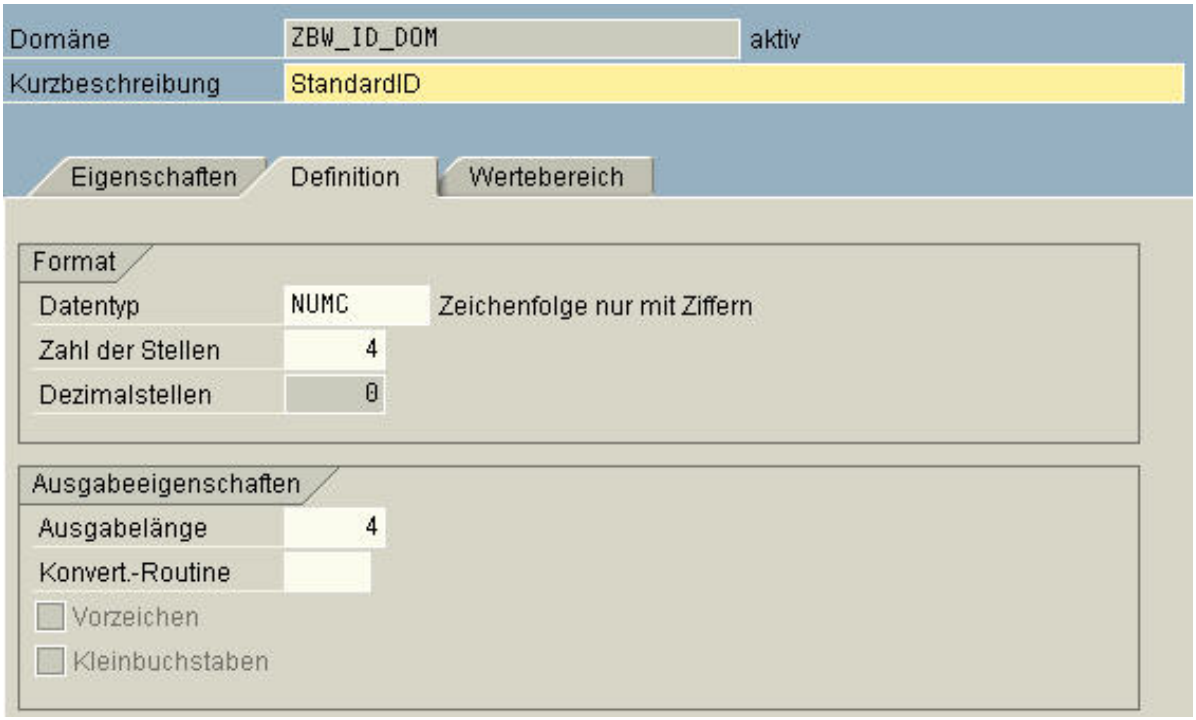


Abbildung 29.22.: Datenbank: Domänen pflegen

Wir befinden uns in „Domäne pflegen“. Hier können wir eine Domäne anlegen. Es müssen definiert werden „Kurzbeschreibung“, „Datentyp“, „Zahl der Stellen“, „Dezimalstellen“ und die „Ausgabelänge“ (Abbildung 29.22).

Kurzbeschreibung: „Kurzbeschreibung des Datenelements/Domäne“
Datentyp: „NUMC“ (Zeichenfolge nur mit Ziffern) (Beispielsitzung)
Zahl der Stellen: „4“ (Anzahl der Zahlen, zB. maximal Wert: 9999)
Dezimalstellen: „0“ (Anzahl der Nachkommastellen)
Ausgabelänge: „4“ (Ausgabelänge Grösse des Feldes für die Werteingabe)

Übersicht der Datentypen im ABAP Dictionary:

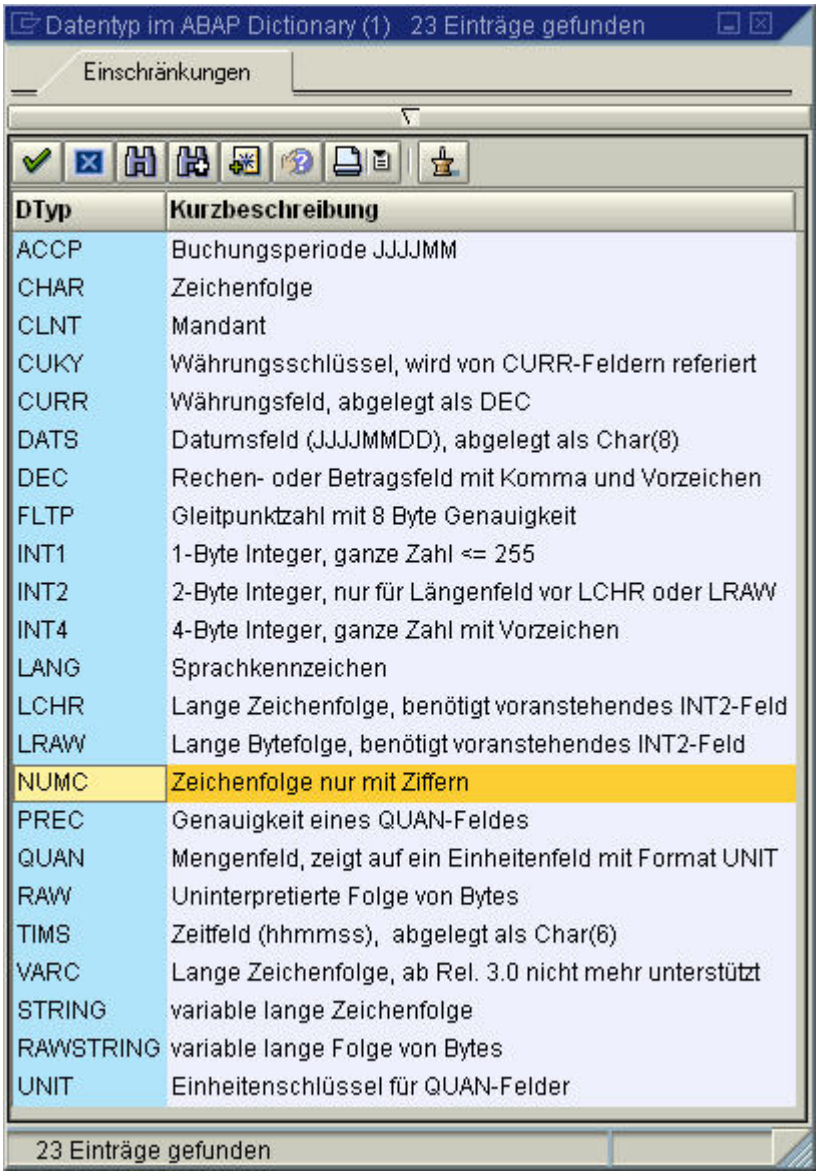


Abbildung 29.23.: Datenbank: Datentypen im ABAP Dictionary

Nach Abspeicherung und „zurück“ befinden wir uns wieder im Reiter „Definition“ (Abbildung 29.22). Folgend müssen die Feldbezeichner eingegeben werden (Reiter „Feldbezeichner“). Nun müssen die Bezeichnungen für die „Kurz“, „Mittel“, „Lang“ und „Überschrift“ mit der „Länge“ festgelegt werden (Abbildung 29.24).

Abbildung 29.24.: Datenbank: Datenelemente und Domänen

Nach der Eingabe und Aktivierung der Domänen/Datenelementen müssen evtl. „Foreign Key“ (Fremdschlüssel) festgelegt werden, um Beziehungen zu anderen Tabellen herzustellen. Dies erfolgt wenn man die Zeile auswählt und dann auf das „Schlüsselsymbol“ klickt (Abbildung 29.25).

29.5.5. Erstellung des Fremdschlüssels

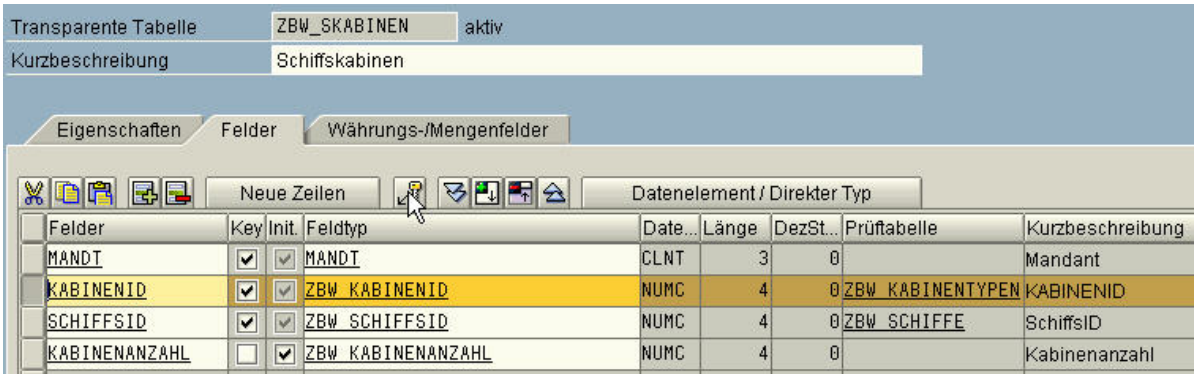


Abbildung 29.25.: Datenbank: Fremdschlüssel anlegen

Folgend erscheint das Fenster „Fremdschlüssel anlegen“ (Abbildung 29.26). Vorgehensweise:

1. Eingabe der „Kurzbeschreibung“
2. Die Prüftabelle kann aus der Auflistung aller Tabellen entnommen werden
3. Auf „Vorschlag erzeugen“ klicken und übernehmen

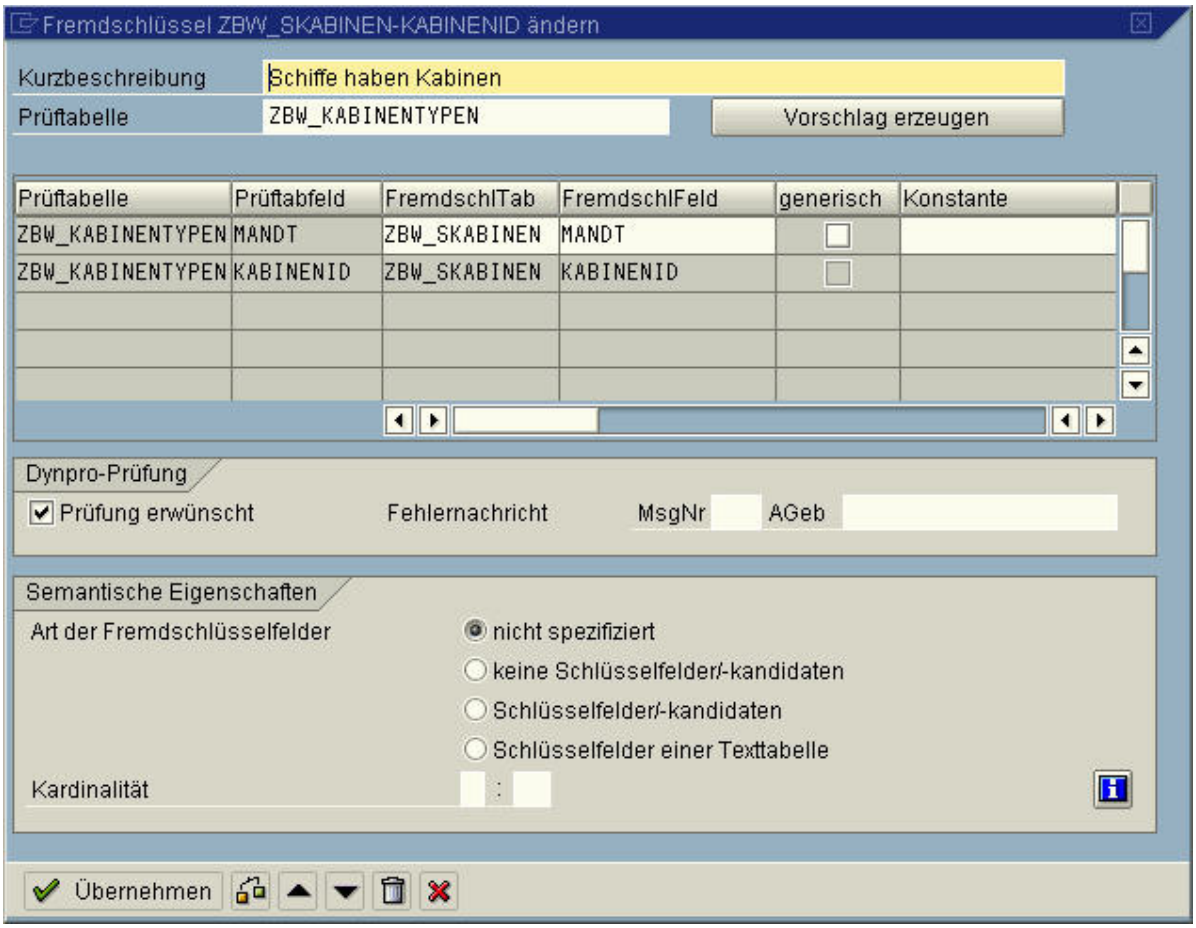


Abbildung 29.26.: Datenbank: Eigenschaften der Fremdschlüssel

29.5.6. Sonderfall Währungs- und Mengenfelder

In SAP R/3 gibt es einen Sonderfall für Währungs- und Mengenfelder, wobei wir nur den Fall Währungsfelder behandeln. Für Preise müssen Datenelemente vom Typ „CURR“ (Currency) die wiederum müssen einem Währungsschlüssel „CUKY“ (Currenc Key) zugeordnet werden. Außerdem muss noch das Referenzfeld und die Referenztabelle angegeben werden. Die Wertetabelle für alle Währungen ist Systemweit „TCURC“. Zusätzlich muss eine Spalte „WAEHRUNG“ mit dem Datenelement „MWAER“ vom Datentyp „CUKY“ erstellt werden.

Wir befinden uns in der SE80 Oject Navigator. Dem Attribut „Kabinenpreis“ soll ein Währungsdatentyp zugeordnet werden (Abbildung 29.27).

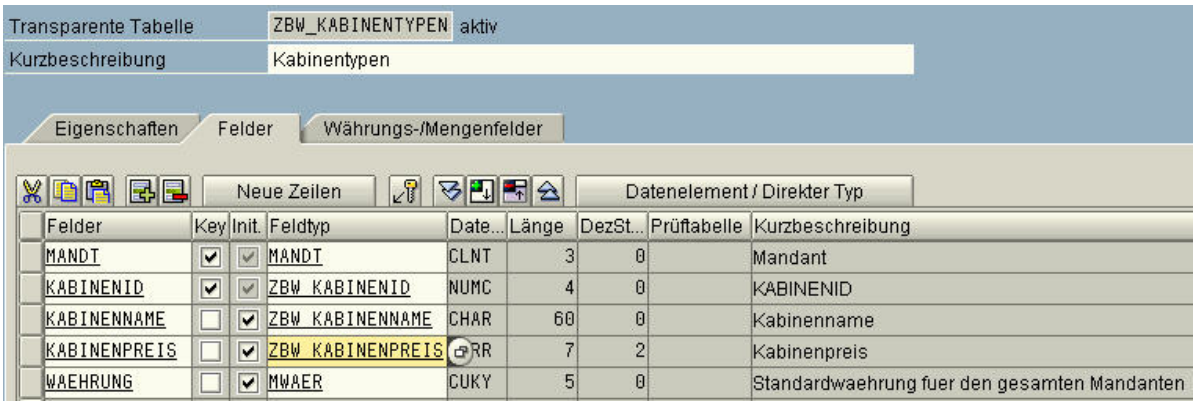


Abbildung 29.27.: Datenbank: Sonderfall Währungs- und Mengenfelder

Vorgehensweise:

1. Doppelklick auf dem Datenelement „ZBW_KABINENPREIS“ (Abbildung 29.27)

2. Vergabe der Domäne, Beispiel: „ZBW_PREIS_DOM“ (Abbildung 29.28)

3. Eingabe der Kurzbeschreibung: „Preis“ (Abbildung 31)
   Datentyp: „CURR“ (Währungsfeld)
   Zahl der Stellen: „7“ (Anzahl der Zahlen, z.B. maximal Wert: 9999999)
   Dezimalstellen: „2“ (Anzahl der Nachkommastellen)
   Ausgabelänge: „9“ (Ausgabelänge Grösse des Feldes für die Werteingabe)

4. Nach Abspeicherung und „zurück“ befinden wir uns wieder im Reiter „Definition“ (Abbildung 29.29). Folgend müssen die Feldbezeichner eingegeben werden (Reiter „Feldbezeichner“). Nun müssen die Bezeichnungen für die „Kurz“, „Mittel“, „Lang“ und „Überschrift“ mit der „Länge“ festgelegt werden (Abbildung 29.30).

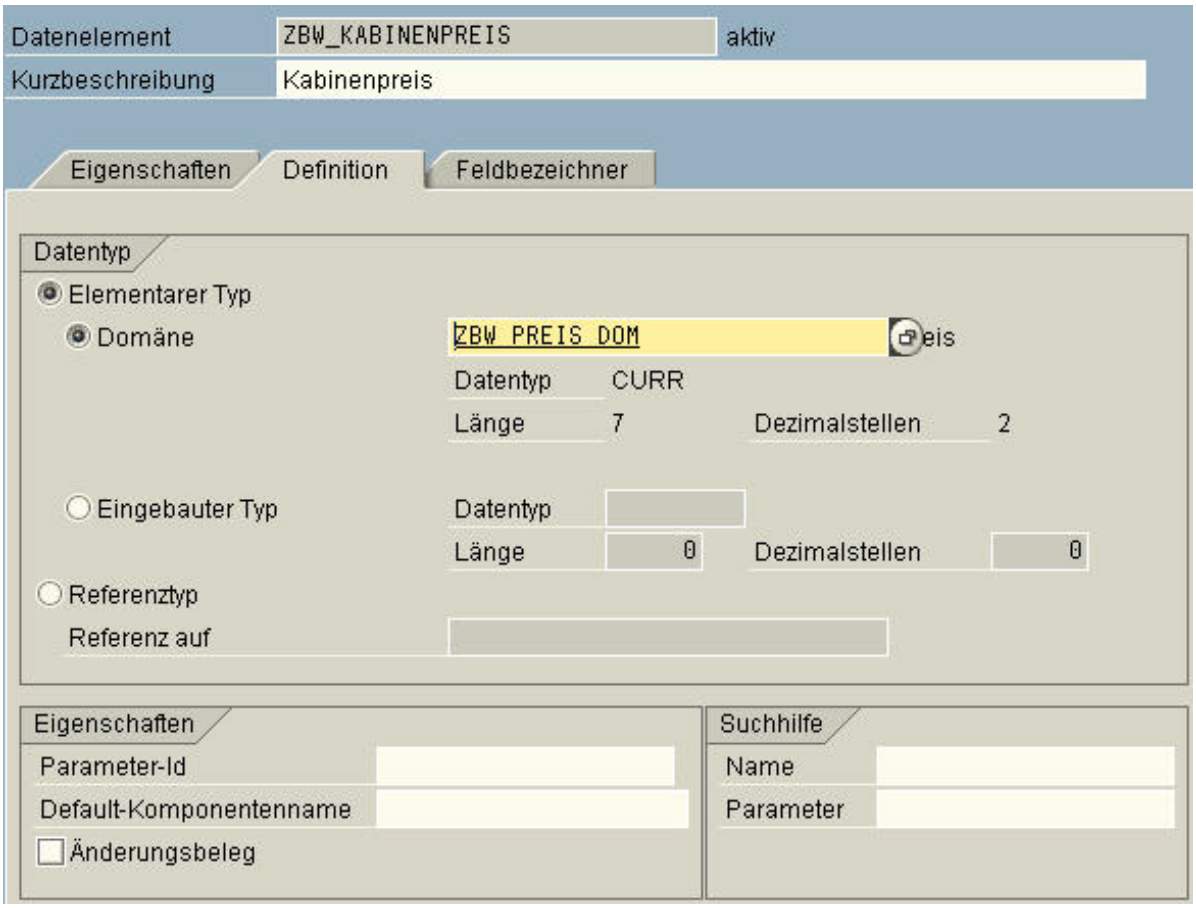


Abbildung 29.28.: Datenbank: Erstellung des Währungsdatenelements (1)

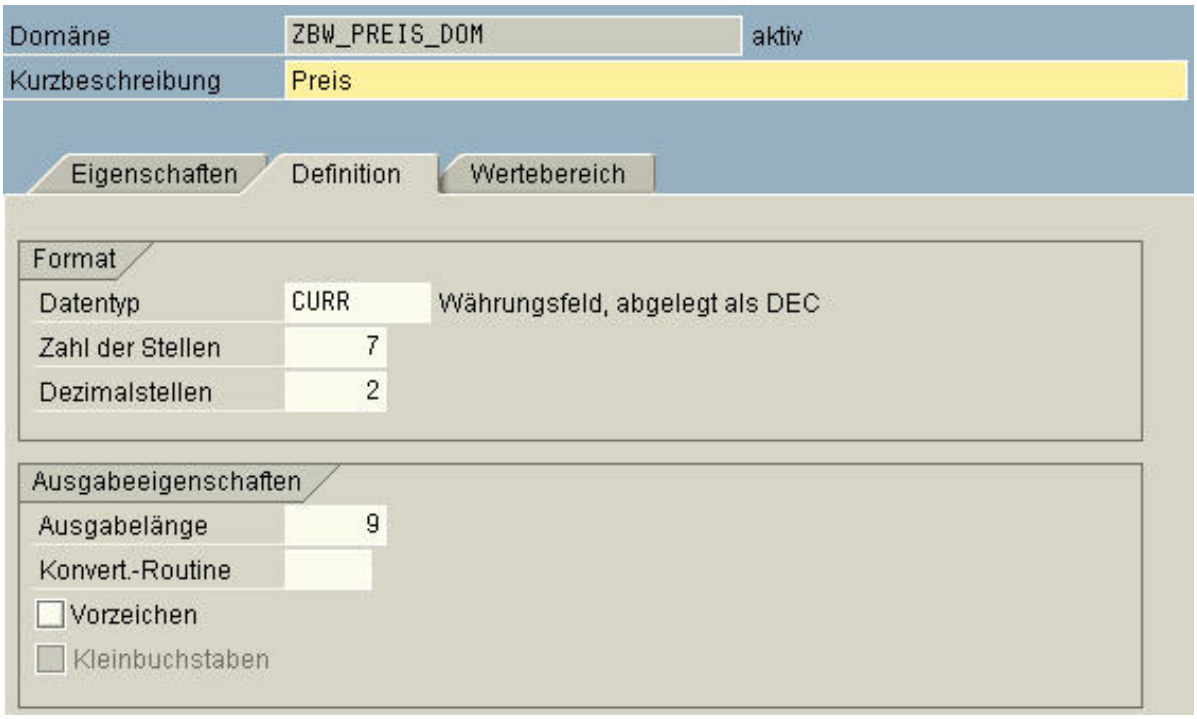


Abbildung 29.29.: Datenbank: Erstellung der Währungsdomäne

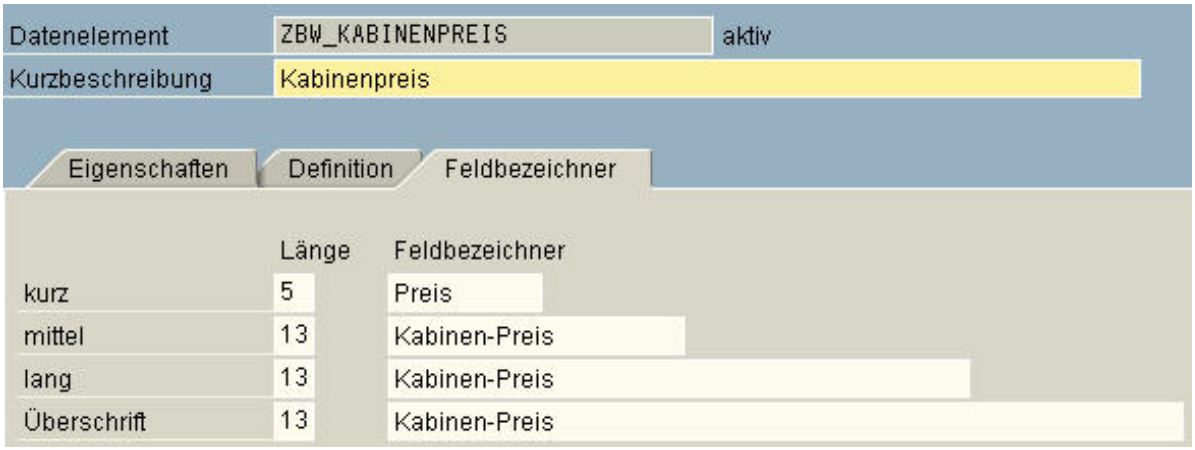


Abbildung 29.30.: Datenbank: Erstellung des Währungsdatenelements (2)

Nachdem das Datenelement und der Datentyp gespeichert und aktiviert wurden, muss dem Attribut „Kabinenpreis“ die Refernztabelle „ZBW_KABINENTYPEN“ und das Referenzfeld „WAEHRUNG“ zugewiesen werden. Dadurch wird im System eine Internationalisierung der Währungen ermöglicht (Abbildung 29.31).

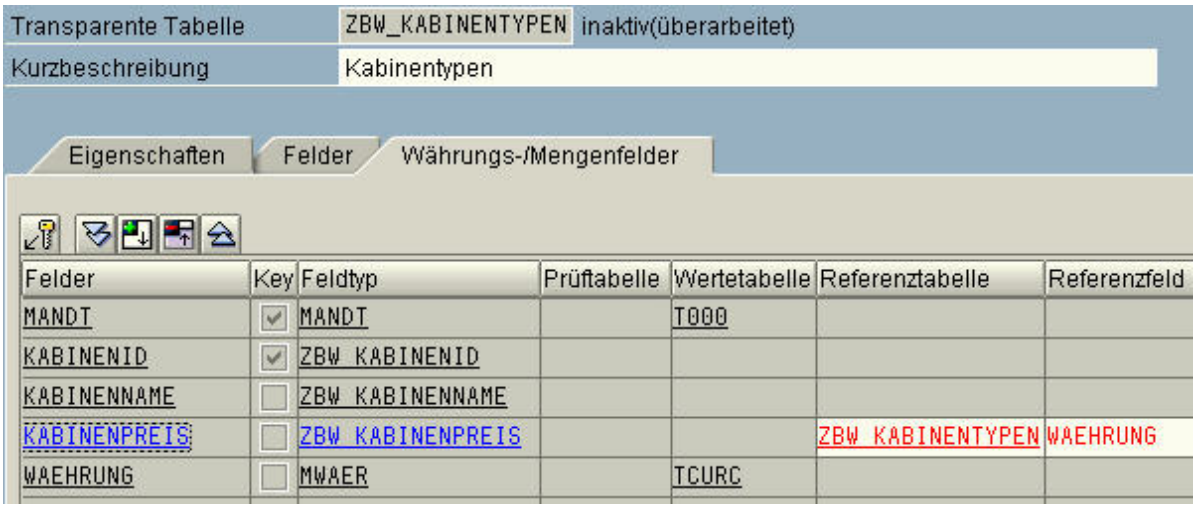


Abbildung 29.31.: Datenbank: Zuweisung der Referenztabelle/Referenzfeld

29.5.7. Erstellung der Views

Ein View ist eine gedachte Tabelle und enthält keine Daten. Es ist eine Sicht auf eine oder mehrere Tabellen. Dies wird mit sogenannten „JOIN“-Bedingungen realisiert. In SAP R/3 wird kein Quellcode für einen JOIN benötigt, es wird hierfür ein „Datenbank-View“ angelegt.

Wir befinden uns im Modul SE80 Object Navigator. Unsere Entwicklungsklasse ist bereits angelegt und ausgewählt. Zum Anlegen der Views wählen Sie mit Rechtsklick den Namen der Entwicklingsklasse an und wählen Sie folgenden Menüpfad: „Anlegen“ → „DDIC-Object“ → „View“ (Abbildung 29.32).

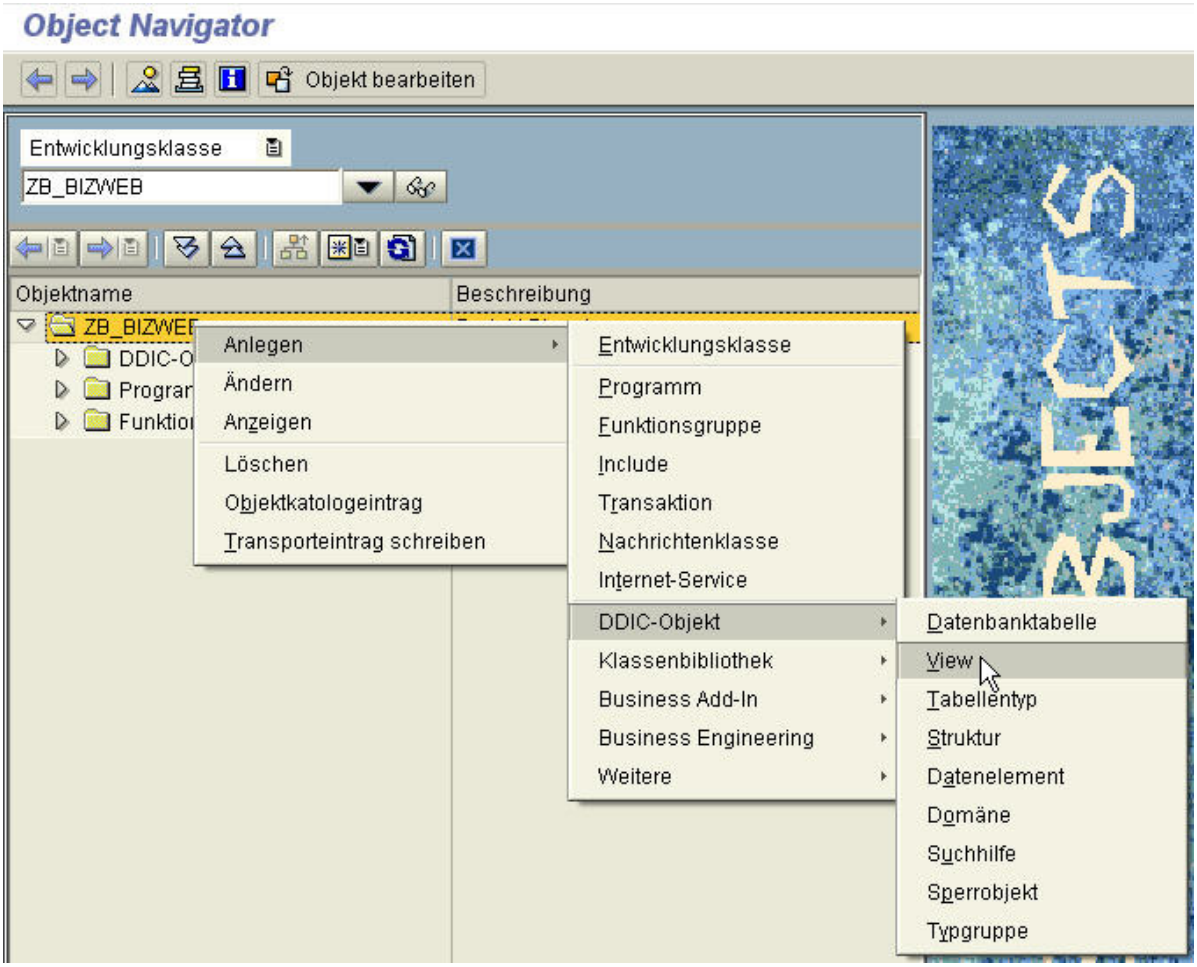


Abbildung 29.32.: Datenbank: Erstellung der Views – Menüpfad

Es erscheint eine Auswahl an möglichen Views. Geben Sie einen Viewnamen und wählen Sie „Datenbankview“ aus (Abbildung 29.33).

Abbildung 29.33.: Datenbank: Eingabe des Viewnamen

Als Beispiel bearbeiten wir den View „ZBW_Buchungen“. Vorgehensweise:

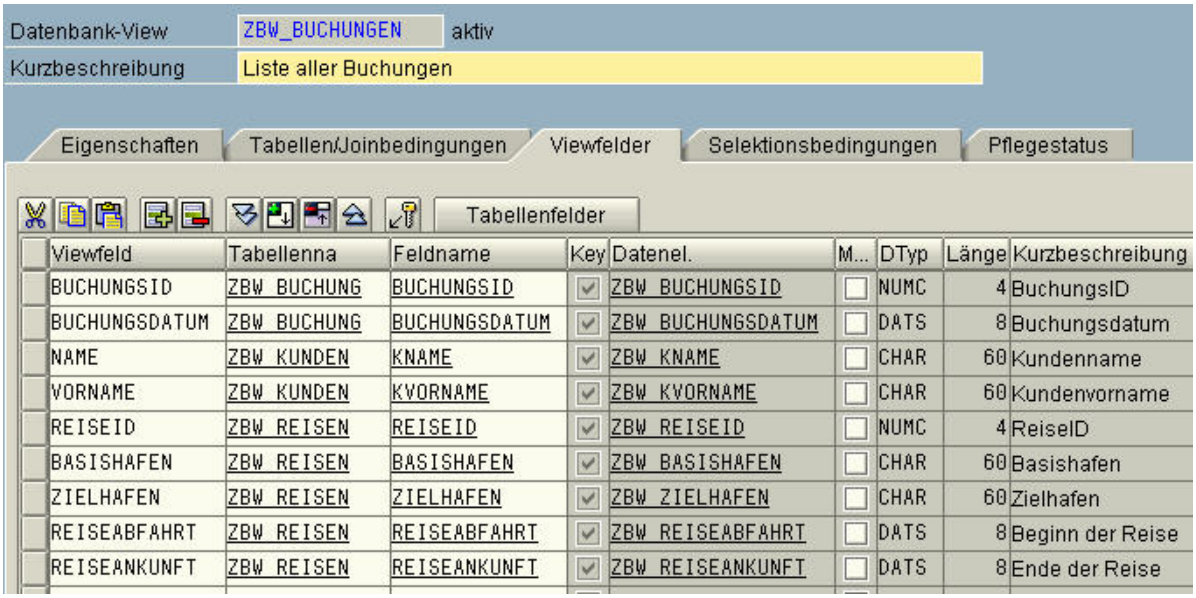
1. Unter dem Reiter „Tabellen/Joinbedingungen“ kann die Kurzbeschreibung eingegeben werden.

2. Eintragung der Tabellen die miteinander Verknüpft werden sollen (links unter Spalte „Tabellen“).

3. Nun müssen die betreffenden Attributnamen eingetragen werden, über die die Beziehung hergestellt werden soll (Fremdschlüssel). „Mandt“ ist vom System vorgegeben und muss immer eingetragen werden (rechts unter „Joinbedingungen“) (Abbildung 29.34).

4. Unter dem Reiter „Viewfelder“ können nun die gewünschten Attribute aus den einzelnen Tabellen, die später zur Ansicht kommen sollen, aufgelistet werden (Abbildung 37).

Abbildung 29.34.: Datenbank: Verknüpfung der Tabellen – Joinbedingungen



Viewfeld	Tabellenna	Feldname	Key	Datenel.	M...	DTyp	Länge	Kurzbeschreibung
BUCHUNGSID	ZBW_BUCHUNG	BUCHUNGSID	☑	ZBW_BUCHUNGSID	☐	NUMC	4	BuchungsID
BUCHUNGSDATUM	ZBW_BUCHUNG	BUCHUNGSDATUM	☑	ZBW_BUCHUNGSDATUM	☐	DATS	8	Buchungsdatum
NAME	ZBW_KUNDEN	KNAME	☑	ZBW_KNAME	☐	CHAR	60	Kundenname
VORNAME	ZBW_KUNDEN	KVORNAME	☑	ZBW_KVORNAME	☐	CHAR	60	Kundenvorname
REISEID	ZBW_REISEN	REISEID	☑	ZBW_REISEID	☐	NUMC	4	ReiseID
BASISHAFEN	ZBW_REISEN	BASISHAFEN	☑	ZBW_BASISHAFEN	☐	CHAR	60	Basishafen
ZIELHAFEN	ZBW_REISEN	ZIELHAFEN	☑	ZBW_ZIELHAFEN	☐	CHAR	60	Zielhafen
REISEABFAHRT	ZBW_REISEN	REISEABFAHRT	☑	ZBW_REISEABFAHRT	☐	DATS	8	Beginn der Reise
REISEANKUNFT	ZBW_REISEN	REISEANKUNFT	☑	ZBW_REISEANKUNFT	☐	DATS	8	Ende der Reise

Abbildung 29.35.: Datenbank: Erstellung der Viewfelder

29.5.8. Erstellung eines Programms

Wir befinden uns im Modul SE80 Object Navigator. Unsere Entwicklungsklasse ist bereits angelegt und ausgewählt. Zum Anlegen der Programme wählen Sie mit Rechtsklick den Namen der Entwicklingsklasse an und wählen Sie folgenden Menüpfad: „Anlegen“ → „Programm“ (Abbildung 29.36).

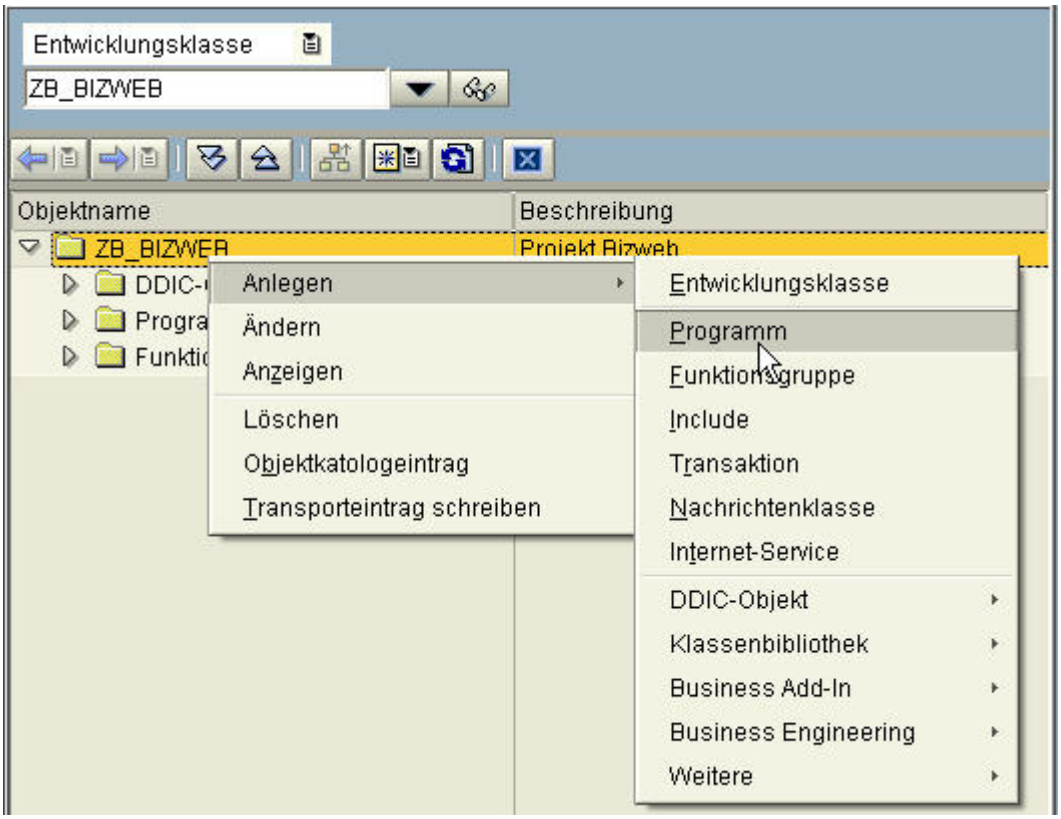


Abbildung 29.36.: Datenbank: Erstellung eines Programms – Menüpfad

Es erscheint eine Aufforderung zur Eingabe des Programmnamens. TOP-Include wird nicht berücksichtigt (Abbildung 29.37).

Abbildung 29.37.: Datenbank: Programm – Namensvergabe

Vorgehensweise: Die Grundeinstellung sind meist ausreichend.

1. Vergabe eines Titels
2. Klicken Sie auf „Sichern“ (Abbildung 29.38)

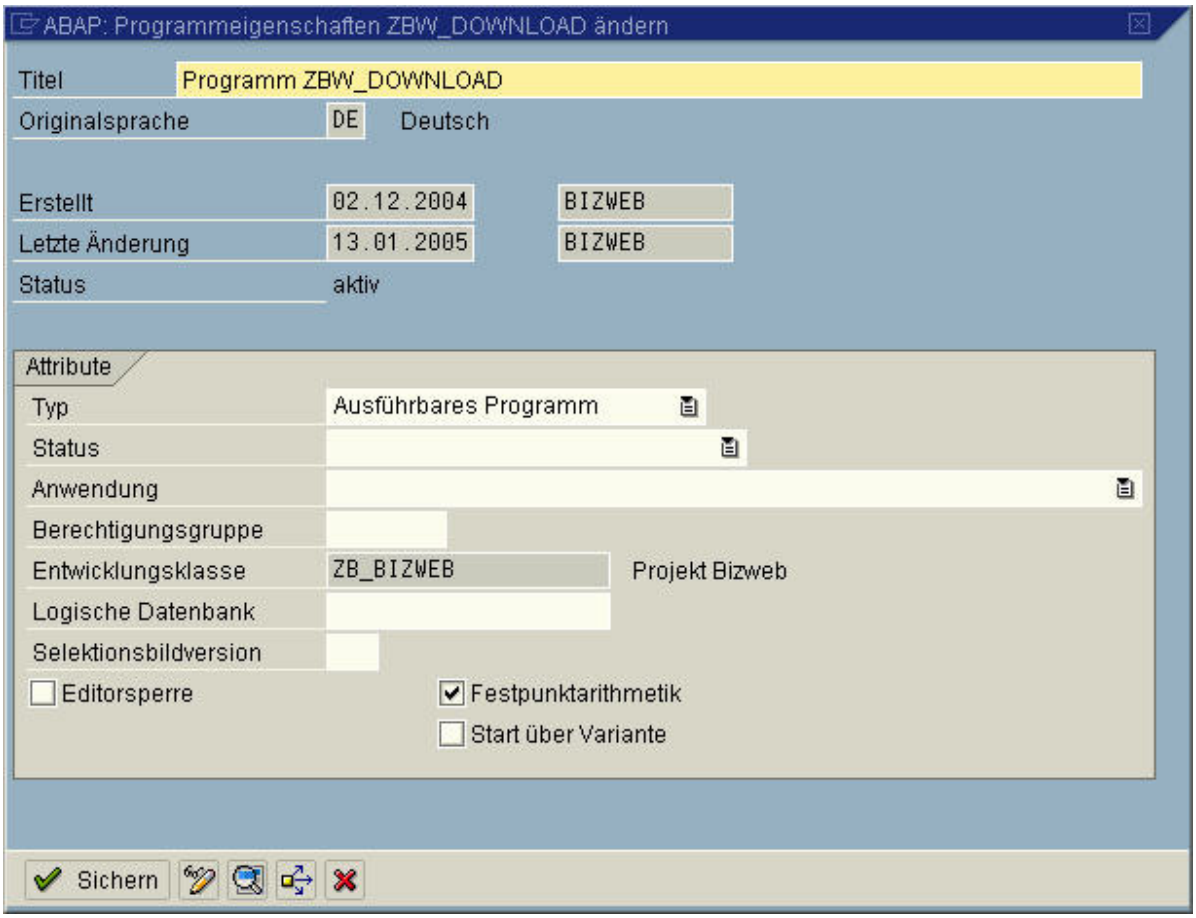


Abbildung 29.38.: Datenbank: Programmeigenschaften

29.6. Datensicherung

29.6.1. Programm: ZBW_DOWNLOAD

Mit diesem Programm wird einen Datensicherung von dem Inhalt aller für das BizWeb-Projekt benötigten Stammdaten vorgenommen. Der Datendownload erfolgt auf die lokale Festplatte in das Verzeichnis „C:\temp\“. Es ist zwingend notwendig dieses Verzeichnis vorher mit allen Schreibrechten anzulegen.

```
1  *&---------------------------------------------------------------------*
2  *& Report  ZBW_DOWNLOAD                                               *
3  *&                                                                     *
4  *&---------------------------------------------------------------------*
5  *&                                                                     *
6  *&                                                                     *
7  *&---------------------------------------------------------------------*
8
9  REPORT  zbw_download.
10
11 *& anlegen der working areas und der internen Tabellen*
```

```
12 DATA: "TABELLE ZBW_ANBIETER
13 wa_zbw_anbieter TYPE zbw_anbieter,
14 it_zbw_anbieter LIKE TABLE OF wa_zbw_anbieter,
15
16 "TABELLE ZBW_KREDITKARTEN
17 wa_zbw_kreditkarten TYPE zbw_kreditkarten,
18 it_zbw_kreditkarten LIKE TABLE OF wa_zbw_kreditkarten,
19
20 "TABELLE ZBW_KUNDEN
21 wa_zbw_kunden TYPE zbw_kunden,
22 it_zbw_kunden LIKE TABLE OF wa_zbw_kunden,
23
24 "TABELLE ZBW_SCHIFFE
25 wa_zbw_schiffe TYPE zbw_schiffe,
26 it_zbw_schiffe LIKE TABLE OF wa_zbw_schiffe,
27
28 "TABELLE ZBW_KABINENTYPEN
29 wa_zbw_kabinentypen TYPE zbw_kabinentypen,
30 it_zbw_kabinentypen LIKE TABLE OF wa_zbw_kabinentypen,
31
32 "TABELLE ZBW_SKABINEN
33 wa_zbw_skabinen TYPE zbw_skabinen,
34 it_zbw_skabinen LIKE TABLE OF wa_zbw_skabinen,
35
36 "TABELLE ZBW_REGIONEN
37 wa_zbw_regionen TYPE zbw_regionen,
38 it_zbw_regionen LIKE TABLE OF wa_zbw_regionen,
39
40 "TABELLE ZBW_REISEN
41 wa_zbw_reisen TYPE zbw_reisen,
42 it_zbw_reisen LIKE TABLE OF wa_zbw_reisen,
43
44 "TABELLE ZBW_BUCHUNG
45 wa_zbw_buchung TYPE zbw_buchung,
46 it_zbw_buchung LIKE TABLE OF wa_zbw_buchung.
47
48 *& Kopieren des Inhalts der transparenten Tabellen in die internen Tabellen*
49 SELECT * FROM zbw_anbieter INTO TABLE it_zbw_anbieter.
50 SELECT * FROM zbw_schiffe INTO TABLE it_zbw_schiffe.
51 SELECT * FROM zbw_kreditkarten INTO TABLE it_zbw_kreditkarten.
52 SELECT * FROM zbw_skabinen INTO TABLE it_zbw_skabinen.
53 SELECT * FROM zbw_buchung INTO TABLE it_zbw_buchung.
54 SELECT * FROM zbw_kunden INTO TABLE it_zbw_kunden.
55 SELECT * FROM zbw_kabinentypen INTO TABLE it_zbw_kabinentypen.
56 SELECT * FROM zbw_regionen INTO TABLE it_zbw_regionen.
57 SELECT * FROM zbw_reisen INTO TABLE it_zbw_reisen.
58
59 *& Aufruf der Export Funktion, die internen Tabellen werden auf die Festplatte
60 gespeichert*
61 CALL FUNCTION 'WS_DOWNLOAD'
62 EXPORTING
63 filename = 'c:\temp\zbw_anbieter.dat'
64 TABLES
65 data_tab = it_zbw_anbieter.
66
67 CALL FUNCTION 'WS_DOWNLOAD'
68 EXPORTING
69 filename = 'c:\temp\zbw_kreditkarten.dat'
70 TABLES
71 data_tab = it_zbw_kreditkarten.
72
73 CALL FUNCTION 'WS_DOWNLOAD'
74 EXPORTING
75 filename = 'c:\temp\zbw_kunden.dat'
76 TABLES
77 data_tab = it_zbw_kunden.
```

```
78
79 CALL FUNCTION 'WS_DOWNLOAD'
80 EXPORTING
81 filename = 'c:\temp\zbw_schiffe.dat'
82 TABLES
83 data_tab = it_zbw_schiffe.
84
85 CALL FUNCTION 'WS_DOWNLOAD'
86 EXPORTING
87 filename = 'c:\temp\zbw_kabinentypen.dat'
88 TABLES
89 data_tab = it_zbw_kabinentypen.
90
91 CALL FUNCTION 'WS_DOWNLOAD'
92 EXPORTING
93 filename = 'c:\temp\zbw_skabinen.dat'
94 TABLES
95 data_tab = it_zbw_skabinen.
96
97 CALL FUNCTION 'WS_DOWNLOAD'
98 EXPORTING
99 filename = 'c:\temp\zbw_buchung.dat'
100 TABLES
101 data_tab = it_zbw_buchung.
102
103 CALL FUNCTION 'WS_DOWNLOAD'
104 EXPORTING
105 filename = 'c:\temp\zbw_reisen.dat'
106 TABLES
107 data_tab = it_zbw_reisen.
108
109 CALL FUNCTION 'WS_DOWNLOAD'
110 EXPORTING
111 filename = 'c:\temp\zbw_regionen.dat'
112 TABLES
113 data_tab = it_zbw_regionen.
```

Quelltext 29.1: Datenbank: ZBW_DOWNLOAD

29.6.2. Programm: ZBW_UPLOAD

Mit diesem Programm wird eine Datenwiederherstellung von dem Inhalt aller für das BizWeb-Projekt benötigten Stammdaten vorgenommen. Die Wiederherstellung erfolgt von der lokalen Festplatte aus dem Verzeichnis „C:\temp\“. Es ist zwingend notwendig dieses Verzeichnis vorher mit allen Leserechten und den Daten anzulegen. Folgende Dateien mit dem entsprechend passendem Inhalt werden benötigt:

Dateiname	Inhalt
zbw_anbieter.dat	Auflistung der Anbieter
zbw_kreditkarten.dat	Kreditkartennummern
zbw_kunden.dat	Auflistung der Kunden
zbw_schiffe.dat	Auflistung der Schiffsnamen
zbw_kabinentypen.dat	Kabninentypen
zbw_skabinen.dat	Beziehung zwischen Kaninen und Schiffen
zbw_buchung.dat	Alle vorgenommenen Buchungen
zbw_reisen.dat	Die angebotenen Reisen
zbw_regionen.dat	Auflistung aller Regionen

Tabelle 29.3.: Datenbank: Benötigte Dateien zum Import der Daten

```
1  *&---------------------------------------------------------------------*
2  *& Report  ZBW_UPLOAD                                                 *
3  *&                                                                     *
4  *&---------------------------------------------------------------------*
5  *&                                                                     *
6  *&                                                                     *
7  *&---------------------------------------------------------------------*
8
9  REPORT  zbw_upload. "NO STANDARD PAGE HEADING.
10
11 *& anlegen der working areas und der internen Tabellen*
12 DATA: "TABELLE ZBW_ANBIETER
13       wa_zbw_anbieter TYPE zbw_anbieter,
14       it_zbw_anbieter LIKE TABLE OF wa_zbw_anbieter,
15
16       "TABELLE ZBW_KREDITKARTEN
17       wa_zbw_kreditkarten TYPE zbw_kreditkarten,
18       it_zbw_kreditkarten LIKE TABLE OF wa_zbw_kreditkarten,
19
20       "TABELLE ZBW_KUNDEN
21       wa_zbw_kunden TYPE zbw_kunden,
22       it_zbw_kunden LIKE TABLE OF wa_zbw_kunden,
23
24       "TABELLE ZBW_SCHIFFE
25       wa_zbw_schiffe TYPE zbw_schiffe,
26       it_zbw_schiffe LIKE TABLE OF wa_zbw_schiffe,
27
28       "TABELLE ZBW_KABINENTYPEN
29       wa_zbw_kabinentypen TYPE zbw_kabinentypen,
30       it_zbw_kabinentypen LIKE TABLE OF wa_zbw_kabinentypen,
31
32       "TABELLE ZBW_SKABINEN
33       wa_zbw_skabinen TYPE zbw_skabinen,
34       it_zbw_skabinen LIKE TABLE OF wa_zbw_skabinen,
35
36       "TABELLE ZBW_REGIONEN
37       wa_zbw_regionen TYPE zbw_regionen,
38       it_zbw_regionen LIKE TABLE OF wa_zbw_regionen,
39
40       "TABELLE ZBW_REISEN
41       wa_zbw_reisen TYPE zbw_reisen,
42       it_zbw_reisen LIKE TABLE OF wa_zbw_reisen,
43
44       "TABELLE ZBW_BUCHUNG
45       wa_zbw_buchung TYPE zbw_buchung,
46       it_zbw_buchung LIKE TABLE OF wa_zbw_buchung.
47
48 *& Aufruf der "WS_UPLOAD" Funktion, die die internen Tabellen mit den Inhalten der
```

```
Festplatte füllt*
CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_anbieter.dat'
     TABLES
          data_tab = it_zbw_anbieter.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_kreditkarten.dat'
     TABLES
          data_tab = it_zbw_kreditkarten.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_kunden.dat'
     TABLES
          data_tab = it_zbw_kunden.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_schiffe.dat'
     TABLES
          data_tab = it_zbw_schiffe.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_kabinentypen.dat'
     TABLES
          data_tab = it_zbw_kabinentypen.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_skabinen.dat'
     TABLES
          data_tab = it_zbw_skabinen.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_buchung.dat'
     TABLES
          data_tab = it_zbw_buchung.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_reisen.dat'
     TABLES
          data_tab = it_zbw_reisen.

CALL FUNCTION 'WS_UPLOAD'
     EXPORTING
          filename = 'c:\temp\zbw_regionen.dat'
     TABLES
          data_tab = it_zbw_regionen.

*& Die Interne Tabellle wird nun Zeile für Zeile ins Working Area übertragen und danach
in die transparenten Tabellen eingefügt*
LOOP AT it_zbw_kreditkarten INTO wa_zbw_kreditkarten.
  INSERT INTO zbw_kreditkarten VALUES wa_zbw_kreditkarten.
  COMMIT WORK.
ENDLOOP.

LOOP AT it_zbw_kunden INTO wa_zbw_kunden.
  INSERT INTO zbw_kunden VALUES wa_zbw_kunden.
  COMMIT WORK.
ENDLOOP.
```

```
115
116 LOOP AT it_zbw_schiffe INTO wa_zbw_schiffe.
117   INSERT INTO zbw_schiffe VALUES wa_zbw_schiffe.
118   COMMIT WORK.
119 ENDLOOP.
120
121 LOOP AT it_zbw_kabinentypen INTO wa_zbw_kabinentypen.
122   INSERT INTO zbw_kabinentypen VALUES wa_zbw_kabinentypen.
123   COMMIT WORK.
124 ENDLOOP.
125
126 LOOP AT it_zbw_skabinen INTO wa_zbw_skabinen.
127   INSERT INTO zbw_skabinen VALUES wa_zbw_skabinen.
128   COMMIT WORK.
129 ENDLOOP.
130
131 LOOP AT it_zbw_regionen INTO wa_zbw_regionen.
132   INSERT INTO zbw_regionen VALUES wa_zbw_regionen.
133   COMMIT WORK.
134 ENDLOOP.
135
136 LOOP AT it_zbw_reisen INTO wa_zbw_reisen.
137   INSERT INTO zbw_reisen VALUES wa_zbw_reisen.
138   COMMIT WORK.
139 ENDLOOP.
140
141 LOOP AT it_zbw_buchung INTO wa_zbw_buchung.
142   INSERT INTO zbw_buchung VALUES wa_zbw_buchung.
143   COMMIT WORK.
144 ENDLOOP.
145
146 LOOP AT it_zbw_anbieter INTO wa_zbw_anbieter.
147   INSERT INTO zbw_anbieter VALUES wa_zbw_anbieter.
148   COMMIT WORK.
149 ENDLOOP.
```

Quelltext 29.2: Datenbank: ZBW_UPLOAD

29.6.3. Auflistung der Programme in SE80

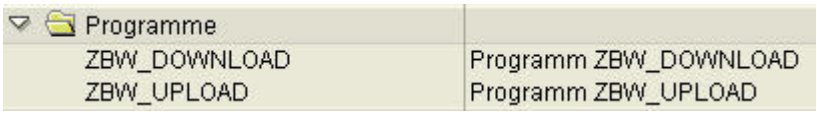


Abbildung 29.39.: Datenbank: Auflistung der Programme in SE80

30. JCo

30.1. Einleitung

Der Java–Connector ist im BizWeb–Projekt das entscheidende Bindeglied zwischen dem proprietären SAP–System mit seinen Daten und dem Java–Web Service. Ohne ihn ist es zwar nicht unmöglich auf die SAP–Daten zuzugreifen oder diese zu manipulieren, doch soll im Projektzusammenhang besonderes Augenmerk auf die Geschäftsprozesslogik gelegt werden, die nur mit dem Java–Connector umzusetzen ist. Eine ODBC–Verbindung würde beispielsweise den Ansprüchen nicht genügen und ein Aufrufen der Funktionsbausteine in SAP nur schwer oder gar nicht ermöglichen.

In diesem Kapitel wird erläutert, was es mit dem SAP–Java–Connector (JCo) auf sich hat, wofür er verwendet wird, wie man ihn installiert, wie er aufgebaut ist und welche Elemente von ihm bei in der Projekt–Lösung zur Anwendung kommen. Darauf folgt die konkrete Beschreibung der Integration des JCos in das Gesamtkonzept des Web Services, wie Daten aus der SAP–Datenhaltung gelesen und neue Daten eingefügt werden. Das dabei auftretende Problem der Datenkonvertierung wird darauf folgend erläutert, bis es schließlich zum Ende zur Übersicht der Klassen kommt und ein Fazit zusammen mit den Quellenangaben das JCo–Kapitel komplettiert.

30.2. Einordnung in den Projekt–Kontext

Eine Zielvorgabe des Projektes „BizWeb“ war es, den Geschäftsprozess über ein SAP–System abzuwickeln. Alle Daten für die Kreuzfahrten werden dort vorgehalten. Der Web Service muss also Daten aus SAP R/3 lesen und auch welche hinein schreiben. Um den Web Service in die Lage zu versetzen diese Daten in einer geordneten Form zu erhalten und auch Buchungsdaten an das SAP–System zu übergeben, wurde ein Schnittstellen–Programm entwickelt, welches sich der SAP–Java–Connector Technologie bedient. Wenn in diesem Kapitel davon gesprochen wird, dass „etwas“ an den Web Services „geschickt“ wird, geschieht dies zur Vereinfachung, denn die nötigen Klassen wurden in einem separaten Package in die Web Service–Anwendung integriert und sind somit Bestandteil des Web Services. Im Folgenden wird die Einordnung in den Projekt–Kontext grafisch dargestellt:

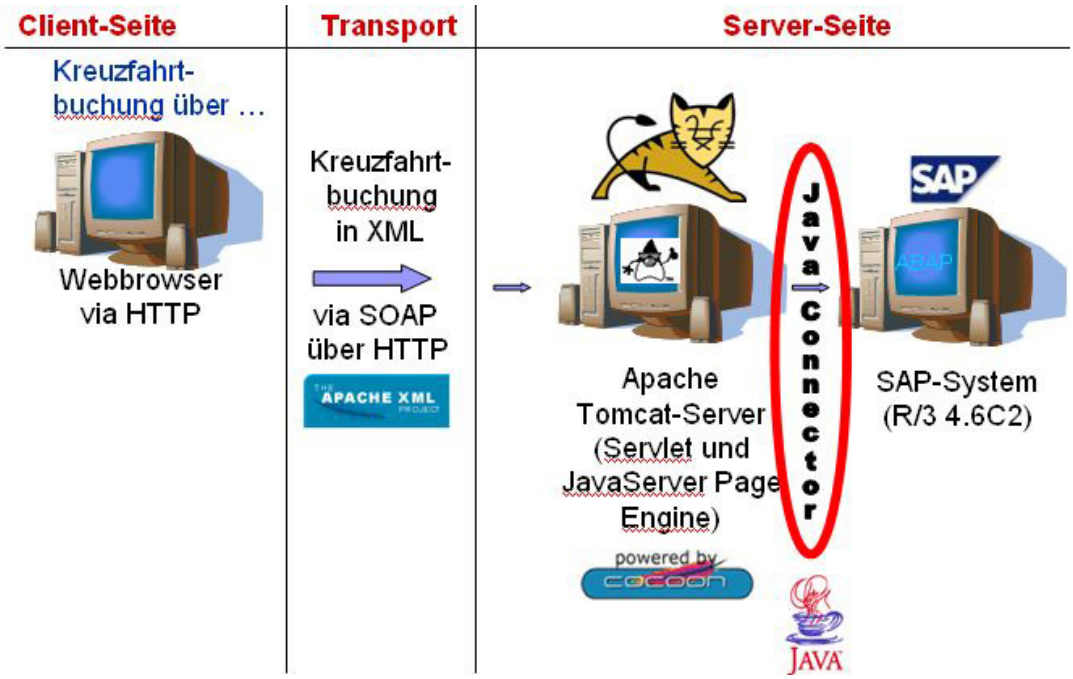


Abbildung 30.1.: JCo: Einordnung in den Projekt–Kontext

30.2.1. Wofür wird der Java–Connector benötigt?

Die SAP–Java–Connector Technologie wird benötigt um dem Web Service für die Kreuzfahrtbuchung die Daten aus dem SAP–System zur Verfügung zu stellen bzw. die Buchungsdaten in das SAP–System zu übertragen. Der Web Service nutzt die nötigen Klassen der erstellten Schnittstellen–Lösung um mit dem SAP–System zu kommunizieren. Über den SAP–Java–Connector können im SAP–System vorhandene Programme (sog. remotefähige Funktionsbausteine) aufgerufen werden, welche dann über Übergabeparameter Daten empfangen und senden können. Die Funktionsbausteine stellen die korrekte Arbeitsweise innerhalb des SAP–Systems sicher und lösen bei Fehlern Exceptions aus.

30.3. Java–Connector Grundlagen

Der SAP Java Connector (JCo) ermöglicht grundsätzlich die Kommunikation zwischen SAP–Systemen und Java–Programmen. Der JCo versetzt den Programmierer in die Lage RFC–Programme auf einem Nicht–SAP–System in Java zu schreiben, ohne direkt auf das RFC API (Remote Function Call Application Programming Interface) zugreifen zu müssen.

Unter einem Remote Function Call (RFC) wird ein entfernter Funktionsaufruf verstanden. Es handelt sich dabei um den Aufruf eines remotefähigen Funktionsbausteins, wel-

cher in einem anderen System vorgehalten wird als das aufrufende Programm. Das RFC API ist eine C–Bibliothek, auf welcher der JCo aufsetzt und der SAP–Java–Connector kann daher als objektorientierte Schicht über dem RFC API angesehen werden.

Trotz seines Namens ist der Java–Connector kein Connector bzw. Ressourcenadapter im Sinne der J2EE Connector–Spezifikation. Aufrufe auf SAP–Systeme durch Java (inbound) und Aufrufe von Java durch SAP–Systeme (outbound) sind möglich[1]. Der JCo wird von SAP zur Verfügung gestellt und wird auf deren Internetpräsenz berechtigten Nutzern als Download angeboten.

30.3.1. Installation

Um die Kommunikation zwischen einem Java–Programm und einem SAP–System zu ermöglichen muss zunächst eine Version des Java–Connectors von der Internetpräsenz der SAP herunter geladen werden. Im Projekt wurde die Version 2.1.3 eingesetzt. Der Download umfasst ein Archiv, welches folgende Bestandteile beinhaltet:

- Die Java–Implementierung des Java–Connectors; zu finden in der Datei `sapjco.jar`
- Die DLL–Bibliotheken `sapjcorfc.dll` und `librfc32.dll`, welche den RFC–Stack implementieren
- Eine Dokumentation im HTML–Format
- Beispiel–Programme

Zur Installation müssen zunächst die DLL–Bibliotheken in die Betriebssystemumgebung kopiert werden. Unter Windows ist dies bspw. das Verzeichnis `C:\Windows\system32`. Nachdem die Dateien dorthin kopiert wurden, muss die Projektumgebung unter Eclipse konfiguriert werden. Dazu wird nach einem Rechtsklick auf das betreffende Projekt der Punkt „Properties“ bzw. „Einstellungen“ gewählt. Danach wird unter „Java Build Path“ ⟶ „Libraries“ mit dem Button „Add External JARs...“ die `sapjco.jar` ausgewählt und so dem Projekt zur Verfügung gestellt. Der Java–Connector kann nun genutzt werden.

30.3.2. Die Architektur des Java–Connectors

Das herunter geladene Paket stellt einen Java–Wrapper um die C–basierende RFC–Implementierung bereit. Aufrufe aus einem Java–Programm werden durch den Wrapper über die JNI (Java Native Interface) an die oben beschriebenen C–DLLs weitergeleitet. Diese DLLs kommunizieren ihrerseits mit der RFC–Schicht im SAP–System. Nach der

[1] vgl. [BIS] – SAP Java Connector (JCo), S. 51

Abarbeitung des Aufrufs durch einen remotefähigen Funktionsbaustein wird das Ergebnis über denselben Weg wieder zurück an das aufrufende Java–Programm zurückgegeben. Falls bei der Verarbeitung im SAP–System Fehler auftreten, werden Fehlermeldungen über den gleichen Mechanismus an das externe Programm geliefert. Das Java–Programm wirft diese Fehler dann als Exception aus, und somit ist die Implementierung einer Fehlerbehandlung möglich.

Folgende Abbildung beschreibt die Abarbeitung eines Methodenaufrufs aus einer externen Java–Anwendung in den unterschiedlichen Schichten des Java–Connectors im Zusammenspiel mit dem SAP–System.

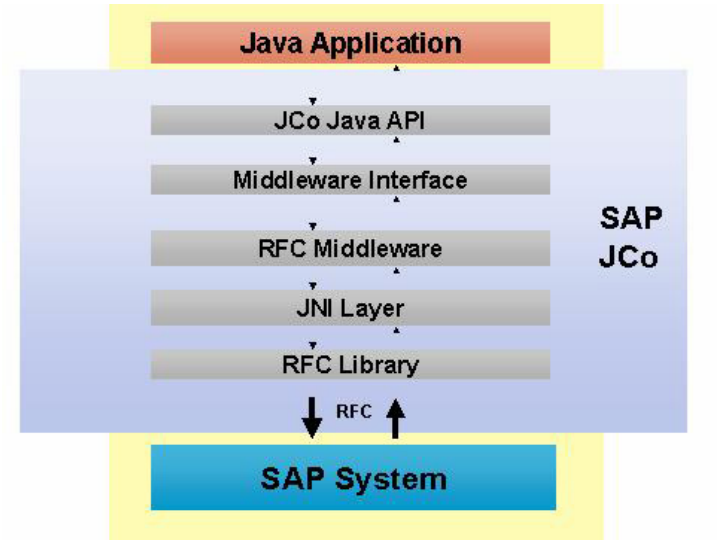


Abbildung 30.2.: JCo: Aufruflogik des Java–Connectors, Quelle: [SAP] – SAP JCo Architecture

Die Verarbeitung des Methodenaufrufs geht von der externen Java–Anwendung aus. Von dort aus delegiert Java den Methodenaufruf zunächst über die JCo–Java–API und dem Middleware Interface zur RFC Middleware Implementierung. Von diesem Punkt aus wird der Java–Aufruf mit Hilfe des Java Native Interface (JNI) zu einem RFC–Aufruf konvertiert und zum SAP–System gesendet. Wie oben bereits erwähnt, wird derselbe Mechanismus auch für die Antwort nach der Verarbeitung durch das SAP–System verwendet. Es ist möglich weitere Middleware–Technologien in das Framework des Java–Connectors mit einzubeziehen. Es ist zum Beispiel möglich, die Kommunikation über SOAP zu realisieren.

Dadurch, dass der Java–Connector alle Aufrufe, welche über die RFC–Schnittstelle an das SAP–System gehen, an die DLLs weiter delegiert, entsteht ein Vorteil für den Nutzer des Java–Connectors. Dieses Vorgehen bringt einen sehr hoher Abstraktionsgrad mit sich, was zur Folge hat, dass es nicht nötig ist, sich mit RFC im Detail auseinanderzuset-

zen. Man muss kein SAP–Fachmann bzw. ABAP–Programmierer sein, um remotefähige Funktionsbausteine aufzurufen und diese in externen Java–Anwendungen zu nutzen[2].

30.3.3. Einsatz und Verwendung des Java–Connectors

Unter diesem Punkt soll der allgemeine Ablauf der Verwendung des Java–Connectors aufgezeigt werden. Zuerst wird grundlegend auf die nötigen Informationen der remotefähigen Funktionsbausteine auf der SAP–Seite eingegangen. In den darauf folgenden Unterpunkten wird die Verwendung der JCo–Klassen, welche für die Projektlösung eingesetzt wurden, erläutert.

RFC–Bausteine

Um remotefähige Funktionsbausteine eines SAP–Systems über eine externe Anwendung nutzen zu können, müssen zunächst bestimmte Informationen über diese bekannt sein. Die remotefähigen Funktionsbausteine werden auch als RFC–Bausteine bezeichnet. Dieser Begriff schließt sowohl die Programmierung und den Aufruf von Remote Function Methods (RFM) als auch die Verwendung von BAPIs (Business Application Programming Interface) ein.

Grundlegende Informationen über RFC–Bausteine sind folgende:

- **Importparameterliste:** Die Liste der Parameter, welcher der Baustein empfängt muss vorliegen
- **Exportparameterliste:** Es muss bekannt sein, welche Parameter mit welchen Typen zurückgeliefert werden
- **Tabellenparameterliste:** Diese Parameter können in beide Richtungen übertragen werden und müssen ebenfalls bekannt sein

Import– und Exportparameter sind im Normalfall einfache Felder oder Strukturen, während Tabellenparameter mehrere Zeilen und Spalten beinhalten können.

All diese Informationen können im SAP–System unter der Transaktion `SE37` eingesehen werden. An dieser Stelle kann auch die Art, die Anzahl und die Datentypen der Parameter ermittelt werden[3][4].

Bevor mit der Programmierung mit Hilfe des Java–Connectors begonnen werden kann, muss festgelegt werden in welcher Form mit den RFC–Bausteinen kommuniziert werden

[2] vgl. [EAI] – Architektur des Java-Connectors, S. 196-197
[3] vgl. [BIS] – BAPIs und andere RFMs, S. 55
[4] vgl. [EAI] – RFC- und BAPI-Clients, S. 191

soll bzw. von welcher „Art“ der Aufruf eines RFC–Bausteins sein soll. Für die Anbindung des SAP–Systems an den Web Service wurde ein synchroner Java–Client programmiert. Dies bedeutet, die Java–Anwendung sendet den RFC–Aufruf und warte solange bis dieser vom SAP–System abgearbeitet und das Ergebnis zurückgeliefert wurde[5]. Dieses wird auch „Synchroner RFC“ genannt und stellt den üblichen Fall dar. Es sind auch andere Formen des Aufrufs möglich, allerdings wird in dieser Ausarbeitung nur das Vorgehen bei einer synchronen Verbindung erläutert.

Verbindungsaufbau

Um überhaupt die Funktionalitäten eines SAP–Systems nutzen zu können, muss zuerst eine Verbindung zu diesem aufgebaut werden. Hierzu gibt es zwei Möglichkeiten:

- **Direkte Verbindungen:** Die Java–Anwendung öffnet aktiv eine Verbindung zum SAP–System und schließt diese wieder, sobald kein Zugriff mehr benötigt wird
- **Gepoolte Verbindungen:** Bei dieser Art der Verbindung nutzt das aufrufende Java–Objekt bereits bestehende Verbindungen, welche aus einem sog. Client–Pool bezogen werden. Wird eine Verbindung nicht mehr benötigt, gibt es diese wieder frei. Die Verbindung wird wieder in den Pool zurückgegeben und kann somit von anderen Nutzern verwendet werden.

Es stellt sich nun die Frage, wann welche Art der Verbindung gewählt werden soll. Dies kann kaum allgemein beantwortet werden, denn die Antwort hängt sehr stark von der Frequentierung des SAP–Systems durch externe Java–Anwendungen ab. Ein Einsatz von gepoolten Verbindungen ist allerdings bei webbasierten Anwendungen sehr sinnvoll, denn dort müssen mehrere unterschiedliche Nutzer verwaltet werden und die Zugriffsdauer und –häufigkeit lässt sich nur sehr schwer vorhersagen. Aus diesen Gründen wurde auch bei der Projekt–Lösung ein Pool von Verbindungen verwendet. Folgende Klassen werden in Zusammenhang mit Verbindungspools benötigt[6]:

- `JCO.Pool` repräsentiert einen Verbindungspool
- `JCO.PoolManager` verwaltet alle Verbindungspools innerhalb der Java Virtual Machine (JVM)

Die Erzeugung eines Verbindungspools kann wie folgt aussehen:

```
1 JCO.addClientPool(
2   "Beispielpool", // Name des Verbindungspools
3   10,             // max. Anzahl Verbindungen
4   "107",          // SAP Client
5   "user",         // Benutzer
```

[5] vgl. [EAI] – Synchroner RFC, S. 197
[6] vgl. [EAI] – Synchroner RFC, S. 197-198

```
6   "password",      // Passwort
7   "DE",            // Sprache
8   "servername",    // Servername
9   "00" );          // Systemnummer
```

Wenn eine neue Verbindung aus dem Pool benötigt wird, kann eine solche Anfrage über `client = JCO.getClient("Beispielpool");` gestellt werden. Die `getClient`–Methode liefert entweder eine bestehende offene Verbindung zurück oder öffnet eine neue Verbindung, falls die Anzahl maximaler Verbindungen noch nicht erreicht wurde. Mit der Methode `JCO.releaseClient` wird die Verbindung wieder in den Verbindungspool „zurückgegeben“, wenn sie nicht mehr benötigt wird[7].

Aufruf von RFC–Bausteinen

Die Metainformationen aller RFC–Bausteine, die über den JCo aufgerufen werden sollen, müssen – wie oben beschrieben – bekannt sein. Durch das `JCO.Repository`–Objekt können diese Informationen ermittelt werden. Die Metainformationen der RFC–Bausteine können im `JCO.Repository`–Objekt dynamisch vom SAP–System zur Laufzeit abgerufen werden oder fest kodiert sein. Folgende JCO–Klassen und Interfaces werden in Verbindung mit dem JCo Repository eingesetzt:

- **JCO.Repository:** enthält die Metainformationen der RFC–Bausteine
- **IFunctionTemplate:** enthält die Metainformationen für einen RFC–Baustein
- **JCO.Function:** repräsentiert einen RFC–Baustein mit Parametern.
- **JCO.ParameterList:** enthält die Import–, Export– und Tabellen–Parameter einer `JCO.Function`
- **JCO.Structure:** enthält eine Datenstruktur
- **JCO.Table:** enthält eine Tabelle

JCO.Repository Die Erzeugung einer `JCO.Repository`–Instanz erfolgt über zwei Parameter. Der erste gibt einen frei wählbaren Namen des Repository an, während der zweite Parameter den Namen eines Verbindungspools oder ein `JCO.Client`–Objekt enthält, je nachdem welche Verbindungsart genutzt werden soll. Wie folgt könnte die Erzeugung eines `JCO.Repository` aussehen:

```
1 JCO.Repository repository = new JCO.Repository( "Beispielrepository", pool);
```

[7] vgl. [BIS] – SAP Java Connector (JCo), S. 55

JCO.Function Um einzelne RFC–Bausteine zu nutzen, muss ein `JCO.Function`–Objekt erzeugt werden. Dies geschieht über zwei Schritte. Als erstes wird ein `IFunctionTemplate`–Objekt erstellt, welches alle Metainformationen für einen RFC–Baustein enthält. Diese Informationen bekommt der Java–Connector nur einmal und cacht sie aus Geschwindigkeitsgründen. Mit der Methode `getFunctionTemplate` des Interfaces `IFunctionTemplate` wird eine „Funktionsvorlage“ erstellt, aus welcher über die `getFunction`–Methode ein `JCO.Function`–Objekt erzeugt werden kann. Dieses Objekt enthält zusätzlich zu den Metainformationen auch die benötigten Parameter für den Aufruf des RFC–Bausteins.

JCO.ParameterList Mit den Methoden `getImportParameterList`, `getExportParameterList` und `getTableParameterList` kann auf alle Parameter des `JCO.Function`–Objekts zugegriffen werden. Die einzelnen Parameter können innerhalb der zurückgegebenen Parameterlisten über ihren Typ und Namen erreicht werden[8]. Mit der Methode `setValue` können Importparameter gesetzt werden (siehe folgendes Beispiel). Das Funktions–Objekt wird über die `execute`–Methode des `JCO.Client`–Objekts aufgerufen und somit auch der entsprechende RFC–Baustein im SAP–System.

Beispiel 1 Die Verwendung der JCO–Klassen und Objekte soll nun anhand eines Code–Auszugs aus dem implementierten Schnittstellenprogramm verdeutlicht werden:

```
1 JCO.Function function =
2     repository.getFunctionTemplate("ZBW_READ_TABLE").getFunction();
3
4 function.getImportParameterList().setValue(table.toUpperCase(),"QUERY_TABLE");
5 ...
6 connection.execute(function);
```

Im obigen Auszug wird zum Auslesen der Kreuzfahrt–Daten aus dem SAP–System der RFC–Baustein `ZBW_READ_TABLE` benutzt, welcher später noch etwas genauer erläutert wird. Das explizite Erstellen eines `IFunctionTemplate`–Objekts, wie oben beschrieben, wurde an dieser Stelle übersprungen. Als Beispiel für das Setzen eines Parameters soll der Tabellenname, welcher in der String–Variable `table` festgelegt ist, dienen. Der dafür zuständige Übergabeparameter des RFC–Bausteins trägt den Namen `QUERY_TABLE`. Der RFC–Baustein soll die Tabelle, deren Namen übergeben wurde, auslesen und die gewünschten Daten zurückliefern. Damit dies funktioniert müssen noch weitere Parameter gesetzt werden, was hier aber nicht dargestellt werden soll. Die Parameterübergabe sollte generell in Grossbuchstaben erfolgen.

JCO.Structure Um Zugriff auf alle Struktur–Parameter in einer Import– oder Export–Parameterliste zu bekommen, wird die Methode `getStructure` eingesetzt. Eine Struktur besteht aus Feldern, welche jeweils einen bestimmten Datentyp innehaben. Dadurch das ABAP andere Datentypen als Java verwendet, wird ein Mapping der Datentypen

[8] vgl.[BIS] – JCo Repository, S. 55-57

notwendig. `JCO.Structure` enthält typspezifische `get`–Methoden, wie z.B. `getString` für Strings oder `getInt` für Integer–Werte[9].

Beispiel 2 Nachdem der Aufruf des RFC–Bausteins `ZBW_READ_TABLE` erfolgreich verlaufen ist, kann das Ergebnis weiterverarbeitet werden:

```
1 JCO.Table rows = function.getTableParameterList().getTable("DATA");
2 SAPReadData[] tableRows = new SAPReadData[rows.getNumRows()];
3
4 for(int i = 0; i < rows.getNumRows(); i++){
5   rows.setRow(i);
6   tableRows[i] = new SAPReadData(rows.getString("WA"));
7 }
8 ...
```

Als erstes wird die Liste der Tabellenparameter über die `getTableParameterList`–Methode abgerufen, in der über die `getTable`–Methode auf den konkreten Tabelleparameter zugegriffen wird. Der Parameter `DATA` enthält alle Daten, welche vom RFC–Baustein zurückgeliefert werden. Diese Daten werden in einer `JCO.Table` gespeichert. Als nächstes wird ein Array geschaffen, welches Objekte der selbst definierten Klasse `SAPReadData` aufnimmt. Die Größe dieses Arrays wird auf die Anzahl der Zeilen in der JCO.Table `rows` festgelegt. Die for–Schleife läuft nun über diese JCO–Tabelle, liest den Inhalt Zeile für Zeile aus und speichert diese im Array `tableRows`. Da der RFC–Baustein das Abfrageergebnis in Form von durch Semikola getrennten Strings zurückliefert, wird die typspezifische Methode `getString` verwendet um eine Zeile auszulesen.

30.4. Lösung zur Anbindung des Web–Service an das SAP–System

30.4.1. Verwendete RFC–Bausteine

Unter diesem Punkt werden die RFC–Bausteine, welche zum Auslesen der Kreuzfahrtdaten und zum Schreiben der Buchungsdaten verwendet wurden, aus der Sicht des Java–Connectors beschrieben.

ZBW_READ_DATA

Der RFC–Baustein `ZBW_READ_DATA` ist für das Auslesen einer beliebigen Tabelle oder einer Ansicht zuständig. Er stellt dabei eine sichere und korrekte Abarbeitung innerhalb des SAP–Systems sicher. Der Baustein besitzt fünf Importparameter, drei Tabellenparameter und kann sechs verschiedene Exceptions auslösen. Der folgenden Auflistung, welche aus dem Kommentar des Bausteins stammt, sind diese Angaben zu entnehmen.

[9] vgl.[BIS] – JCo Repository, S. 57

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(QUERY_TABLE) LIKE  DD02L-TABNAME
*"     VALUE(DELIMITER) LIKE  SONV-FLAG DEFAULT SPACE
*"     VALUE(NO_DATA) LIKE  SONV-FLAG DEFAULT SPACE
*"     VALUE(ROWSKIPS) LIKE  SOID-ACCNT DEFAULT 0
*"     VALUE(ROWCOUNT) LIKE  SOID-ACCNT DEFAULT 0
*"  TABLES
*"      OPTIONS STRUCTURE  RFC_DB_OPT
*"      FIELDS STRUCTURE  RFC_DB_FLD
*"      DATA STRUCTURE  ZTAB1024
*"  EXCEPTIONS
*"      TABLE_NOT_AVAILABLE
*"      TABLE_WITHOUT_DATA
*"      OPTION_NOT_VALID
*"      FIELD_NOT_VALID
*"      NOT_AUTHORIZED
*"      DATA_BUFFER_EXCEEDED
*"----------------------------------------------------------------------
```

Importparameter Im Parameter `QUERY_TABLE` erwartet der RFC–Baustein den Namen der auszulesenden Tabelle oder Ansicht. Der zweite Parameter (`DELIMITER`) erwartet ein Trennzeichen, welches zwischen den ausgelesen Attribut–Werten der Tabelle gesetzt werden soll, d.h. es stellt die Form der Spaltentrennung dar. Wenn `NO_DATA` auf `X` gesetzt wird, liefert der Baustein keine Inhaltsdaten aus der Tabelle zurück, sondern nur deren Strukturdaten. Der Parameter `ROWSKIPS` wird mit einem Zahlenwert belegt, der bestimmt ab welcher Zeile Daten aus der Tabelle zurückgeliefert werden sollen und `ROWCOUNT` gibt an wie viele Zahlen insgesamt zurückgegeben werden sollen.

Tabellenparameter Dem `OPTIONS`-Parameter werden die Abfragebedingungen in Form einer ABAP–kompatiblen SQL–Where–Klausel innerhalb eines String–Arrays übergeben. Also z.B.:

```
...
public ArrayList getSAPList(String schiffsname) {
  String[] opts = {"SCHIFFSNAME = '"+schiffsname.trim().toUpperCase()+"'"};
...
}
...
```

Dieses String–Array muss danach noch über eine `JCO.Table` mit der `setValue`–Methode gesetzt werden. Ähnliches gilt für den Tabellenparameter `FIELDS`. Hier werden die auszulesenden Spalten der Tabelle in einem String–Array festgelegt. Bleibt das Array leer, so werden alle Spalten ausgelesen. In den folgenden Unterpunkten wird die Verwendung von `FIELDS` und `OPTIONS` noch einmal verdeutlicht. Der Parameter `DATA` enthält, wie oben schon einmal erwähnt alle Daten der Tabellen, welche vom RFC–Baustein ausgelesen wurden.

Exceptions Unter diesem Punkt sind alle Exceptions aufgelistet, welche der RFC–Baustein an die Java–Anwendung zurückliefern kann, um dort wiederum eine Exception auszulösen. Dies geschieht zum Beispiel wenn eine Tabelle mit dem übergebenen Namen nicht im SAP–System vorhanden ist (`TABLE_NOT_AVAILABLE`) oder die Syntax der übergebenen Where–Klausel nicht in Ordnung ist (`OPTION_NOT_VALID`).

ZBW_BUCHUNG

Mit dem RFC–Baustein `ZBW_BUCHUNG` wurde das Einfügen, der für eine Kreuzfahrtbuchung relevanten Daten, in das SAP–System realisiert. Der Baustein erwartet zwei Importparameter, liefert einen Exportparameter zurück und kann eine Exception auslösen. Dieses ist im Folgenden analog zum RFC–Baustein `ZBW_READ_TABLE` dargestellt:

```
1  *"----------------------------------------------------------------------
2  *"*"Lokale Schnittstelle:
3  *"  IMPORTING
4  *"     VALUE(KID) LIKE  ZBW_BUCHUNG-KUNDENID
5  *"     VALUE(RID) LIKE  ZBW_BUCHUNG-REISEID
6  *"  EXPORTING
7  *"     VALUE(BID) LIKE  ZBW_BUCHUNG-BUCHUNGSID
8  *"  EXCEPTIONS
9  *"      VORHANDEN
10 *"----------------------------------------------------------------------
```

Der Parameter `KID` erwartet die Identifikationsnummer des Kunden, welcher die Kreuzfahrt bucht. Die Reise, die angetreten werden soll, wird mit Hilfe des `RID`–Parameters identifiziert. Falls alles in Ordnung ist, werden die übergeben Daten in die entsprechenden Tabellen eingefügt und der Baustein liefert in seinem Exportparamter `BID` die Buchungsnummer dieses Vorgangs zurück. Falls die Reise aber schon einmal in dieser Form gebucht wurde, wird die Exception `VORHANDEN` ausgelöst und dem aufrufenden Java–Programm signalisiert. Ein Schreibvorgang im SAP–System wird dann nicht durchgeführt.

30.4.2. Der allgemeine Ablauf – Auslesen

Der Java Web Service soll später die vom Benutzer gewünschten Daten aus der SAP–Datenbank im Web–Interface angezeigt bekommen. Bis dahin ist es ein relativ langer Weg. Die wichtigste Klasse, die später die Daten an den Webservice weitergibt, ist die Klasse `RFCReadTable`. Sie organisiert mit Hilfe weiterer Klassen im Package `de.bizweb.sap.jco` den Verbindungsaufbau zu SAP, holt sich die Daten, konvertiert diese und gibt sie an den Webservice weiter.

Alles, was der Web Service tun muss, um Daten vom JCo zu erhalten, ist das Aufrufen einer `getSAPList()`–Methode (in der Klasse `RFCReadTable`), die ihm eine ArrayList mit allen benötigten und konvertierten Daten zurückliefert. Das Konvertieren der Daten findet in der Klasse `convertData` statt. Es steht nicht nur eine `getSAPList()`–Methode

zur Verfügung, sondern insgesamt fünf überladene Methoden, so dass die Möglichkeit besteht, die Abfrage der Daten mit Where–Klauseln, abhängig von den Auswahlkriterien des Benutzers, einzuschränken. Beispielsweise werden nicht nur die Kreuzfahrten eines Anbieters herausgesucht, sondern näher spezifiziert nach Basishafen, Reiseabfahrt o.ä. Dazu kann der Kunde in verschiedenen Kombinationen aus den, auf der Web–Oberfläche zur Verfügung stehenden, Auswahllisten seine Wunschdaten auswählen und sich dazu die verfügbaren Kreuzfahrten anzeigen lassen.

Der Verbindungsaufbau zum SAP–System

Zuerst wird im Konstruktor von `RFCReadTable` mit der Klasse `SAPLogon` eine Verbindung zu SAP aufgebaut. Dem Konstruktor der Klasse werden alle notwendigen Daten dafür übergeben.

```
1 public SAPLogon(String clnt,String user,String pass,String lang,String host,String sysn)
2 { ... }
```

Dazu gehören wie oben beschrieben:

- Nummer des Klienten
- Benutzername
- Passwort
- Sprache
- Host
- Systemnummer

Weiterhin bedient sich die `SAPLogon`–Klasse des PoolManagers, der im JCo implementiert ist. Dieser Manager ist notwendig, um für die Verbindung zum SAP–System einen Verbindungspool zu nutzen (s.o.). In diesem Falle sind es maximal 10 Verbindungen, die zur Verfügung stehen. Der PoolManager verhindert außerdem per time–out, dass eine oder mehrere dieser Verbindungen nach längerer Inaktivität bestehen bleiben und beendet diese.

```
1 public void connect() {
2   if(JCO.getClientPoolManager().getPool(SID) == null)
3   JCO.addClientPool(SID, // Anmeldungspool
4       10, // max. Anzahl der Verbindungen
5       clnt,user,pass,lang,host,sysn);
6 }
```

Steht die Verbindung, kann mit dem Auslesen der Daten und deren Struktur begonnen werden.

Lesen der Tabellenstruktur

Um später mit den Daten arbeiten zu können, muss der Web Service übermittelt bekommen, um welche Art von Daten es sich handelt. Dazu gibt es die Methode `getFields`.

```
1  private SAPReadStructure[] getFields(String[] fields) {
2    try {
3      tableFields = SAPReadStructure.getList(
4          logon.getConnection(),
5          logon.getRepository(),
6          CRUISETABLE,     // table
7          ";",             // delimiter
8          "0",             // rowskips
9          "0",             // rowcount
10         "",              // options
11         fields);
12   } catch (Exception e) {
13       e.printStackTrace();
14   }
15 return tableFields;
16 }
```

Die Methode `getFields` liefert ein Array von dem selbst definierten Typ `SAPReadStructure` zurück, welches die Strukturdaten der ausgelesenen Tabelle beinhaltet. Die Übergabeparameter `OPTIONS` wird nicht weiter beachtet. Da nur die Struktur ausgelesen werden soll, wird auf diesen Parameter verzichtet. Der Parameter `fields` ist im Regelfall ein leeres String–Array und dadurch werden alle Spalten der SAP–Tabelle zurückgegeben.

Weiter wird die Methode `getList` aus der Klasse `SAPReadStructure` aufgerufen, welche die eigentliche Arbeit macht. Sie erhält als Übergabewerte:

- aus welcher SAP–Tabelle die Daten selektiert werden sollen (hier ist der Name in der String–Variablen `CRUISETABLE` gespeichert)
- Trennzeichen zwischen den Attributen (hier: Semikolon)
- ab welcher Zeile mit dem Auslesen begonnen werden soll
- wie viele Zeilen insgesamt ausgegeben werden sollen
- `OPTIONS` und `FIELDS` werden „leer“ gelassen (s.o.)

Nun wird durch die Methode `getList` der RFC–Baustein `ZBW_READ_TABLE` in SAP aufgerufen, der die oben aufgelisteten Parameter für die Abfrage übergeben bekommt. Dieser ruft von der übergebenen SAP–Tabelle nur die Struktur und nicht die Daten ab (durch `.setValue('X', "NO_DATA")`):

```
1  public static SAPReadStructure[] getList(...) throws Exception {
2    JCO.Function function =
3        repository.getFunctionTemplate("ZBW_READ_TABLE").getFunction();
4
5    function.getImportParameterList().setValue(table.toUpperCase(),
```

```
6         "QUERY_TABLE");
7
8     function.getImportParameterList().setValue('X', "NO_DATA");
9     ...
10    connection.execute(function);
11    ...
12  }
```

Als letztes werden die einzelnen Attribute im Array `tableRows` gespeichert, dass an seinen Aufrufer `convertData.getSAPTable(..)` in der Methode `RFCReadTable.getSAPList(...)` zurückgeliefert wird:

```
1  JCO.Table rows = function.getTableParameterList().getTable("FIELDS");
2
3  SAPReadStructure[] tableRows = new SAPReadStructure[rows.getNumRows()];
4
5  for (int i = 0; i < rows.getNumRows(); i++) {
6    rows.setRow(i);
7    tableRows[i] = new SAPReadStructure(rows.getString("FIELDNAME"),
8        rows.getString("OFFSET"), rows.getString("LENGTH"),
9        rows.getString("TYPE"), rows.getString("FIELDTEXT"));
10 }
11 return tableRows;
```

Die Struktur der Ergebnistabelle ist nun eingelesen. Dies ist nötig um später den ABAP–Datentypen einer jeden Spalte ermitteln zu können. Zuvor werden aber noch die Inhalte benötigt.

Lesen der Daten

Nun sollen die Inhalte der Tabelle ausgelesen werden. Dazu bekommt die Methode `RFCReadTable.getRows` Optionen (eine SQL–Where–Klausel) und Felder (die Spalten der Tabelle) übergeben. In ihr wird die `SAPReadData.getList()` mit den gleichen Parametern wie bei der Methode `SAPReadStructure.getList()` weiter oben aufgerufen. Allerdings wird nun, falls vom Web Service angefordert, ein String–Array mit einer Abfrageeinschränkung zusätzlich übergeben:

```
1  private SAPReadData[] getRows(String[] options, String[] fields) {
2    try {
3      tableRows = SAPReadData.getList(
4          logon.getConnection(),
5          logon.getRepository(),
6          CRUISETABLE,
7          ";",              // delimiter
8          "0",              // rowskips
9          "0",              // rowcount
10         options,
11         fields);
12   } catch (Exception e) {
13     e.printStackTrace();
14   }
15   return tableRows;
16 }
```

Die `getList()`–Methode ruft wieder den remotefähigen SAP–Funktionsbaustein `ZBW_READ_TABLE` auf, der nun nicht die Struktur, sondern die Daten zurückliefern soll. Deshalb sind die Informationen über das Trennzeichen, `rowskips` und `rowcount` jetzt wichtig, ebenso wie das `options`–String–Array:

```
public static SAPReadData[] getList(...) throws Exception{
  JCO.Function function =
      repository.getFunctionTemplate("ZBW_READ_TABLE").getFunction();

  function.getImportParameterList().setValue(table.toUpperCase(),
      "QUERY_TABLE");
  ...
  function.getImportParameterList().setValue(delimiter,"DELIMITER");
  function.getImportParameterList().setValue(rowskips,"ROWSKIPS");
  function.getImportParameterList().setValue(rowcount,"ROWCOUNT");

  JCO.Table options_table = function.getTableParameterList().getTable("OPTIONS");

  for(int i=0; i < options.length; i++) {
    options_table.appendRow();
    options_table.setValue(options[i].toUpperCase(),"TEXT");
  }
  ...
  connection.execute(function);
  ...
}
```

Nun werden die Daten, wie im Abschnitt „Aufruf von RFC–Bausteinen“ beschrieben, ausgelesen und in das Array `tableRows` eingefügt, welches zurückgegeben wird. Jeder Array–Eintrag entspricht einer Tabellenzeile, allerdings sind die einzelnen Attribute zu einem großen String verschmolzen, nur durch Semikola getrennt:

```
JCO.Table rows = function.getTableParameterList().getTable("DATA");
SAPReadData[] tableRows = new SAPReadData[rows.getNumRows()];

for(int i = 0; i < rows.getNumRows(); i++) {
  rows.setRow(i);
  tableRows[i] = new SAPReadData(rows.getString("WA"));
}

return tableRows;
```

Der bisherige Ablauf bewirkte, dass die Strukturdaten und die Tabelleninhalte aus SAP in zwei separaten Arrays gespeichert wurden.

30.4.3. Konvertierung der Datentypen

Die Tabelleninhalte aus SAP können nicht ohne weiteres weiterverwendet werden, weil beim Export aus SAP die Datentypen verloren gehen und nur Strings zurückgeliefert werden. Deshalb wird mittels der ausgelesenen Struktur der Tabelle jedes Attribut in seinen ursprünglichen Datentyp umkonvertiert.

Die Vorgehensweise:

Jede Tabellenzeile, welche aus dem SAP–System erhalten wird, ist ein langer String. Deshalb werden mittels for–Schleife in der Methode `getSAPTable` der Klasse `convertData` alle Einträge des `SAPReadData`–Arrays einzeln durchlaufen und die Daten Attribut für Attribut herausgefiltert. Ist das letzte Attribut (letzter „Eintrag“ im String) erreicht, wird die nächste Zeile aus dem Data–Array auseinander genommen usw. Zu den Daten gehörten natürlich noch die passenden Datentypen. Sie werden der Reihe nach aus dem `SAPReadStructure`–Array entnommen. Anhand der aus der Struktur ausgelesenen ABAP–Datentypen werden die nun einzelnen Strings in Ihren jeweiligen Java–Datentypen umgewandelt. Zum einfacheren Konvertieren der Werte wurden Methoden geschrieben, welche das Konvertieren für je einen Java–Datentypen übernehmen, z.B. `string2date`. Welcher ABAP–Datentyp dabei dem passenden Java–Datentyp und der passenden JCO–Konstante entspricht, zeigt Tabelle 30.1.

ABAP		Java	JCO
`b`	1-Byte Integer	`int`	`JCO.TYPE_INT1`
`s`	2-Byte Integer	`int`	`JCO.TYPE_INT2`
`l`	4-Byte Integer	`int`	`JCO.TYPE_INT`
`C`	Zeichen	`String`	`JCO.TYPE_CHAR`
`N`	Numerisches Zeichen	`String`	`JCO.TYPE_NUM`
`P`	BCD (Binary Coded Decimal)	`BigDecimal`	`JCO.TYPE_BCD`
`D`	Datum	`Date`	`JCO.TYPE_DATE`
`T`	Zeit	`Date`	`JCO.TYPE_TIME`
`F`	Fließkommazahl	`double`	`JCO.TYPE_FLOAT`
`X`	Raw Data	`byte[]`	`JCO.TYPE_BYTE`
`g`	String (variable Länge)	`String`	`JCO.TYPE_STRING`
`y`	Raw Data (variable Länge)	`byte[]`	`JCO.TYPE_XSTRING`

Tabelle 30.1.: JCo: ABAP-Datentypen und zugehörige Java-Datentypen und JCO-Konstanten

30.4.4. Der allgemeine Ablauf – Buchung

Der Buchungsvorgang ist weitaus weniger umfangreich als das Auslesen der Daten, da nur die beiden vom Web Service gelieferten Werte `kid` (Kunden–ID) und `rid` (Reise–ID) in die SAP–Datenbank eingefügt werden müssen. Das übernimmt die RFC–Baustein `ZBW_BOOKING`, samt der Fehlerbehandlung. Mit Hilfe eines `JCO.Field` wird der vom Baustein zurück gelieferte Exportparameter mit der Buchungs–ID zwischengespeichert und mit `getInt` ausgelesen.

```
1 public class SAPCreateBooking extends Object{
2
3   public static int book(JCO.Client connection,
4       IRepository repository,
5       String kid,
6       String rid) throws Exception
```

```
7   {
8     JCO.Function function =
9         repository.getFunctionTemplate("ZBW_BOOKING").getFunction();
10
11    function.getImportParameterList().setValue(kid, "KID");
12    function.getImportParameterList().setValue(rid, "RID");
13
14    JCO.Field bid = function.getExportParameterList().getField("BID");
15
16    connection.execute(function);
17
18    return bid.getInt();
19  }
20 }
```

30.4.5. Zusammenfassung

Kurz vor Ende dieses Kapitels soll noch einmal anhand der UML–Darstellung ein Überblick der für das Schnittstellen–Programm notwendigen Klassen gegeben werden. Es soll dabei aufgezeigt werden, wie die Klassen untereinander in Zusammenhang stehen und kommunizieren.

Die UML–Ansicht in der unten stehenden Abbildung macht deutlich, dass die Klasse `RFCReadTable` eine zentrale Rolle spielt. Sie enthält die `getSAPList()`–Methoden, welche vom Web Service aufgerufen werden. Deshalb muss in der Klasse zuerst eine Verbindung mit SAP aufgebaut werden, was durch die `SAPLogon`–Klasse geschieht. Jetzt gibt es zwei Möglichkeiten, einmal das Auslesen und einmal das Einfügen von Daten aus bzw. in SAP.

Beim Auslesen und „Weiterreichen“ der Daten an den Web Service werden erst mit Hilfe der `SAPReadStructure`– und `SAPReadData`–Klasse die Daten und die dazugehörigen Datentypen aus SAP ausgelesen. Diese werden in der `convertData`–Klasse sinnvoll miteinander kombiniert und als Objekte in einer ArrayList gespeichert. Diese ArrayList wird danach der `RFCReadTable`–Klasse übergeben und intern von den `getSAPList()`–Methoden an den Web Service als Rückgabewert zurückgeliefert.

Die zweite Möglichkeit, das Buchen, benötigt zusätzlich zur `RFCReadTable`–Klasse die Klasse `SAPCreateBooking`, welche die Kunden– und Reise–ID an das SAP–System sendet und die Buchungs–ID empfängt.

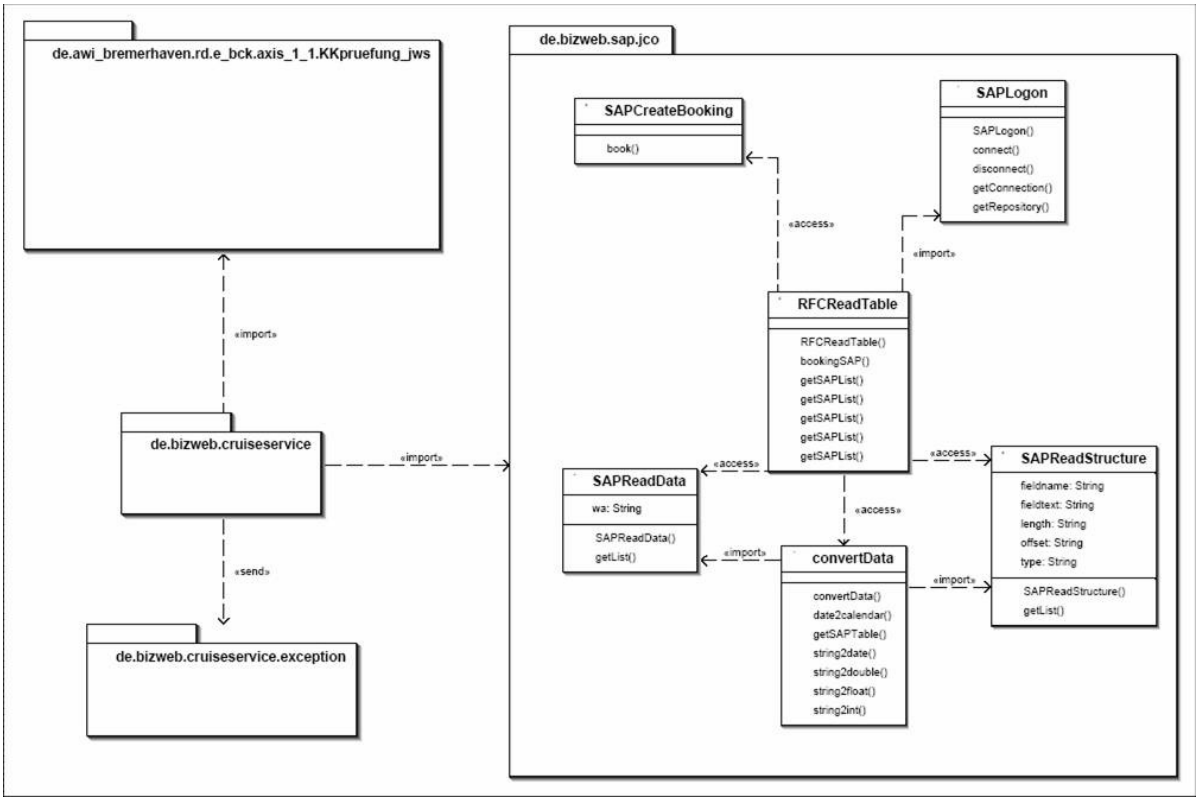


Abbildung 30.3.: JCo: Übersicht über die Klassen des Schnittstellen–Packages

Der gesamte Quellcode der Schnittstellen–Lösung mit Hilfe des SAP–Java–Connectors befindet sich im Anhang der Gesamt–Dokumentation.

30.5. Quellenangaben

[BIS]
Bischofs, Ludger (2002): Anbindung von SAP R/3 über die J2EE Connector-Architektur. Diplomarbeit, Universität Oldenburg

[EAI]
Englbrecht, Michael; Wegelin, Michael: EAI-Programmierung mit mySAP. Schnittstellenentwicklung mit ABAP, C, Java und C#. Heidelberg und München: Spektrum Akademischer Verlag, 2005

[SAP]
http://help.sap.com/

31. Web Service

31.1. Einleitung

Nach intensiven und erfolgreichen Schulungen in der ersten Phase des Projektes BizWeb wurde im Rahmen einer Hausarbeit von jedem Projektmitglied ein kleiner Web Service implementiert, welcher inklusive einer Dokumentation in den Semesterferien fertig zustellen war.

In der zweiten Phase des Projektes sollte nun das vorher definierte Ziel, die beispielhaften Geschäftsprozesse einer Kreuzfahrtbuchung mit Web Services, umgesetzt werden. Für die Realisierung wurden fünf Teilprojekte mit je zwei bis drei Mitgliedern zur Lösung der Gesamtaufgabe gebildet.

Die Aufgabe dieses Teilprojektes war es, folgende Dienstleistungen mittels Web Services zu erstellen:

- Suche einer Kreuzfahrt nach bestimmten Kriterien
- Buchung einer Kreuzfahrt

Hierbei ist zu berücksichtigen, dass der Zugriff sowie das Eintragen der Daten über die Standardsoftware SAP R/3 erfolgt. Die Schnittstellenprogrammierung mittels eines Java Connectors gehörte nicht zu den Aufgaben dieses Teilprojektes.

Die grafische Darstellung der Informationen und Buchungsdaten sollen über einen Client in Form eines Servlets in einem Portal erfolgen.

Die folgende Abbildung zeigt die Einordnung des Teilprojektes in den Gesamtkontext des Projektes:

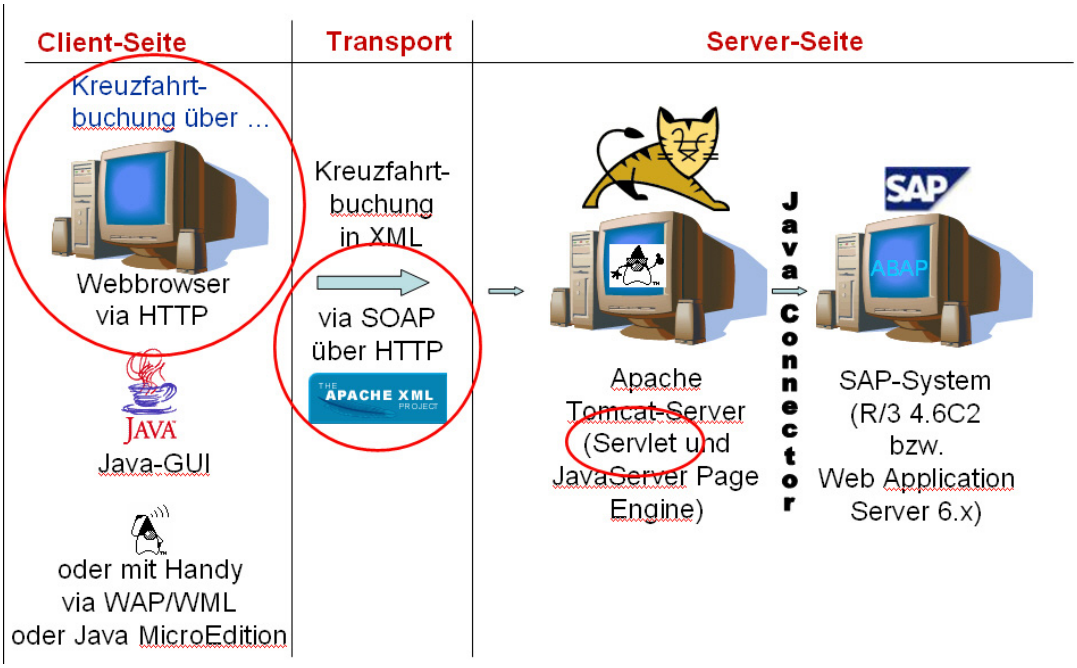


Abbildung 31.1.: Web Service: Einordnung in den Gesamtkontext

31.2. Vorüberlegungen

In diesem Abschnitt soll zum einen der aufgestellte Zeitplan und konzeptionellen Vorüberlegungen, die die Basis zur späteren Realisierung bilden, erläutert werden. Hierzu gehören auch die Thematik der Erweiterbarkeit, sowie die Strukturierung des Web Services.

31.2.1. Zeitplanung

Um das Projekt termingerecht fertig zustellen, ist eine detaillierte Zeitplanung notwendig. Hierbei ist insbesondere eine Abstimmung mit den anderen Teilprojekten erforderlich gewesen. Vor allem die Abnahme und Implementierung des SAP Datenbankmodells stellte eine bedeutende Restriktion dar.

Da das Projekt spätestens bis zum Ende des Semesters abgeschlossen sein musste, galt es folgende Zeitplanung einzuhalten:

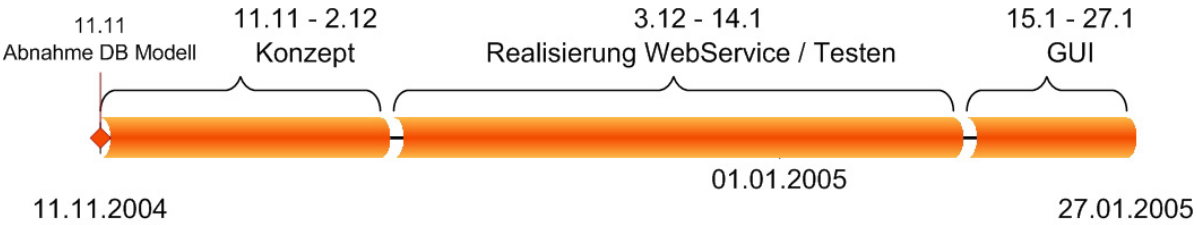


Abbildung 31.2.: Web Service: Zeitplanung

31.2.2. Anforderung an den Web Service

Trotz der komplexen Thematik, Geschäftsprozesse mit Web Services abzubilden, sind folgende Kriterien für die Realisierung unabdingbar:

- Skalierbarkeit
- Erweiterbarkeit
- Einfache Strukturierung der Klassen
- Einfache Wartung

Um erweiterbar und skalierbar zu sein und dadurch mit immer neuen Zielsetzungen und Anforderungen der Benutzer Schritt zu halten, muss dieser Web Service eine möglichst

offene Architektur haben und kompatibel mit den wichtigsten Standards und zukunftsorientierten Technologien sein. Durch die einfache und übersichtliche Strukturierung der Klassen soll der Wartungsaufwand so gering wie möglich gehalten werden.

Die genannten Anforderungen sollen ein fester Bestandteil im Rahmen der Realisierung des Web Services sein.

31.2.3. Weitere Überlegungen

Im Folgenden sollen nun weitere Überlegungen zur Realisierung des Web Services dargestellt werden. Die Umsetzung der Suche nach bestimmten Kreuzfahrten sowie die verwendeten Datentypen zum Austausch der Daten zwischen dem Web Service und dem SAP System stehen hierbei im Mittelpunkt der Überlegungen.

Die Schnittstelle zwischen dem Web Service und dem SAP System stellt der JAVA Connector her. Um den Datenaustausch der beiden Komponenten zu gewährleisten, muss eine einheitliche Datenstruktur geschaffen werden. Das SAP System bekommt vom Web Service entsprechende Parameter, nach denen gesucht wird, übergeben und liefert darauf hin eine Ergebnismenge an den Web Service zurück. Diese Ergebnismenge stellt eine Menge von Objekten dar. Die zu verwendende Datenstruktur muss daher in der Lage sein, eine Liste von Objekten zu speichern. Eine möglichst hohe Performance bei der Ausgabe der Ergebnisse und die Möglichkeit, diese (dynamisch) erweitern zu können, stellen weitere Anforderungen an die Datenstruktur dar.

Aus den folgenden Gründen soll eine ArrayList zum Datenaustausch eingesetzt werden:

- beliebige Erweiterbarkeit
- Speicherung von Objekten
- Garantie der Reihenfolge der Objekte

Verschiedene Abfragen zur Selektion möglicher Kreuzfahrten sollen durch die Überladung einer Methode mit entsprechenden Parametern erfolgen, wie beispielsweise:

Suche nach Schiffsnamen ⇒ `getList(String schiffsname)`

Suche nach Schiffsnamen und Region ⇒ `getList(String schiffsname,String region)`

31.2.4. Textuelle Beschreibung von Nutzungsszenarien

Die Geschäftsprozesse:

- Suche einer Kreuzfahrt nach bestimmten Kriterien
- Buchung einer Kreuzfahrt

sollen nun in Form von Use Case –, Sequenz– und Kollaborationsdiagrammen näher erläutert werden und der grundsätzliche Ablauf des Web Services in Verbindung mit dem SAP System aufgezeigt werden.

Use Case Diagramm

Der Web Service soll für den Kunden zwei Dienstleistungen bereitstellen, zum einen kann dieser nach vorgegebenen Parametern Kreuzfahrten suchen und zum anderen eine Buchung einer gewünschten Kreuzfahrt vornehmen. Die Anbieter von Kreuzfahrten pflegen ihre Datenbanken. Die Daten von allen Anbietern werden für den Veranstalter konsolidiert und sind in einer SAP Datenbank hinterlegt.

Der Veranstalter kann – wie der Kunde – Kreuzfahrten suchen und eine Buchung vornehmen. Des Weiteren ist er für die Verwaltung der SAP Datenbank verantwortlich.

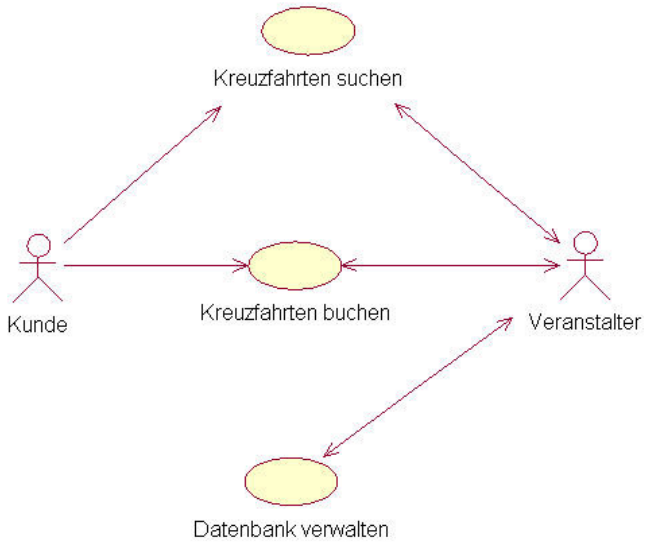


Abbildung 31.3.: Web Service: Use Case

Sequenzdiagramme

Wie aus dem Use Case Diagramm zu entnehmen müssen zwei Geschäftsvorfälle unterschieden werden. Der Ablauf der Suche und der Buchung von Kreuzfahrten sollen mit Hilfe von Sequenzdiagrammen im Folgenden dargestellt werden.

Sequenzdiagramm zum Suchen von Kreuzfahrten Wie aus der unteren Abbildung ersichtlich ist, besteht das Diagramm aus zwei Akteuren, die in dem System agieren können. Der Veranstalter ist für die Datenpflege im SAP System verantwortlich, das heißt die Überarbeitung, Änderung und das Anlegen von Datensätzen. Dabei greift der Veranstalter nur auf die Ebene des SAP Systems zu.

Der Kunde kann sich mittels eines Web Services über mögliche Kreuzfahrten informieren, das heißt er hat keinen direkten Zugriff auf das SAP Systems, sondern kann die Daten über die Ebene des Web Services anfordern.

Die gewünschten Informationen zu Kreuzfahrten werden vom Kunden an den Web Service geschickt. Der Web Service stellt mit den übermittelten Suchkriterien eine Anfrage an das SAP System. Sofern Datensätze zu den angefragten Merkmalen in der SAP Datenbank existieren, werden diese an den Web Service geschickt. Bleibt die Suchanfrage erfolglos wird eine Fehlermeldung ausgegeben. Die vom SAP System erhaltenen Daten werden mittels des Web Services dem Kunden zur Verfügung gestellt.

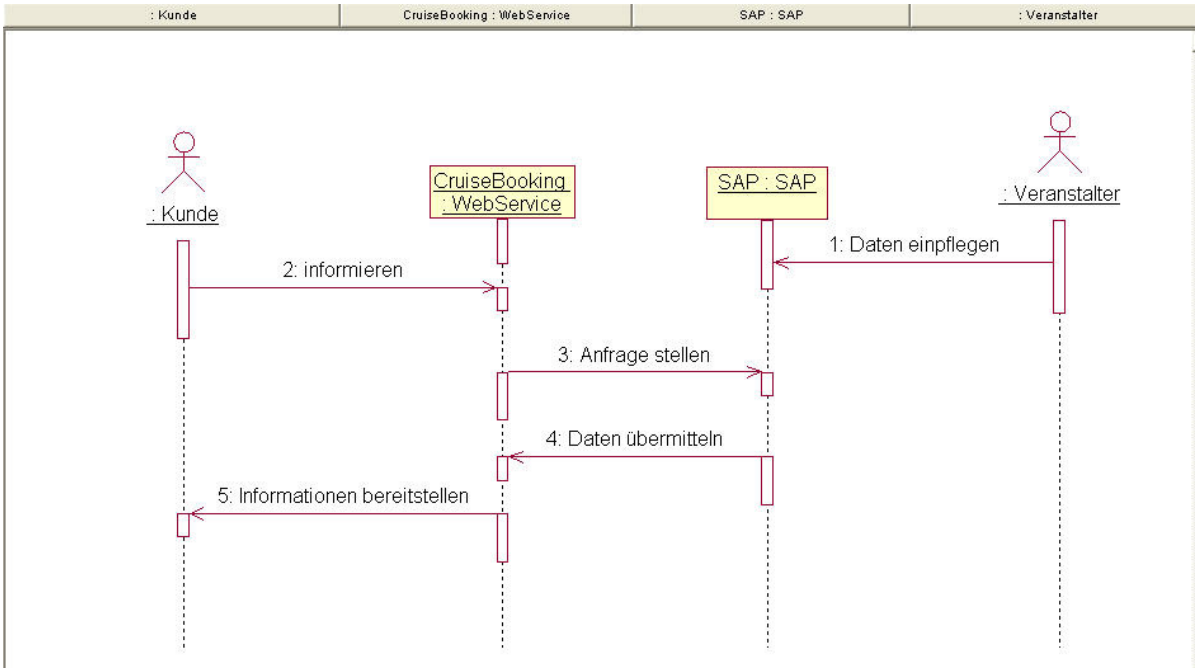


Abbildung 31.4.: Web Service: Sequenzdiagramm Search

Sequenzdiagramm zum Buchen von Kreuzfahrten Wie auch das bereits oben erwähnte Diagramm, besteht auch das Sequenzdiagramm zum Buchen von Kreuzfahrten aus zwei Akteuren, wobei hier Buchung über drei Ebenen stattfindet:

- Web Service CruiseBooking
- Web Service Kreditkartenprüfung
- SAP System

Der Veranstalter agiert wie bei der Suche von Kreuzfahrten nur im SAP System, um die Daten zu pflegen.
Der Kunde stellt mit Hilfe seiner Kunden–, Buchungs– und Kreditkartennummer eine Anfrage an den CruiseBooking Web Service. Die Kreditkartennummer wird an den Web Service Kreditkartenprüfung weitergeleitet, der diese Nummer auf Gültigkeit prüft. Bei erfolgreicher Prüfung wird ein Rückgabewert an den Service CruiseBooking übermittelt, der dann die Buchungsanfrage an das SAP System weiterreicht. Ansonsten bekommt der Kunde eine Fehlermeldung von diesem Web Service. Ist die Kunden–, und Buchungsnummer gültig (d.h. die Kreuzfahrtreise existiert, ist noch nicht ausgebucht und der Kunde ist beim Veranstalter registriert) wird die Buchung im SAP System vorgenommen und eine Buchungsbestätigung an den Web Service CruiseBooking übermittelt. Der Web Service stellt dem Kunden, die vom SAP System vergebene Buchungsnummer zur Verfügung.

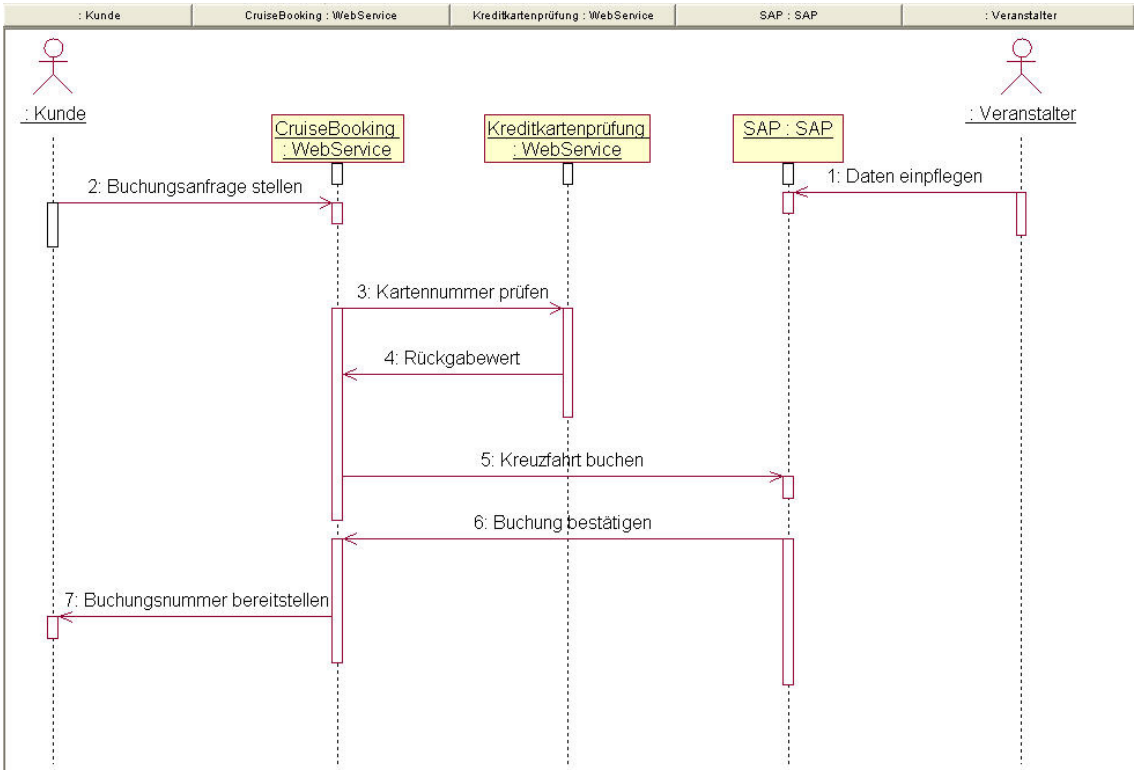


Abbildung 31.5.: Web Service: Sequenzdiagramm CruiseBooking

Kollaborationsdiagramme

Aus den erstellten Sequenzdiagramm können mit Hilfe von Rational Rose die dazugehörigen Kollaborationsdiagramme erstellt werden. Die folgenden Diagramme sollen die Objekte (Kunde, Web Service, SAP System und Veranstalter) und ihre Zusammenarbeit untereinander visualisieren. Im Vergleich zu den obigen Sequenzdiagrammen spielt der zeitliche Ablauf bei dieser Darstellung keine Rolle, sondern die ereignisbezogene Darstellung zwischen dem Kunden, Web Service, SAP System und Veranstalter.

Kollaborationsdiagramm zum Suchen von Kreuzfahrten Wie aus der Abbildung ersichtlich greift der Veranstalter auf das SAP System zur Datenpflege zu. Hierbei handelt es sich um eine einseitige Interaktion. Bei allen anderen Beziehungen (Kunde – Web Service, Web Service – SAP) liegen bilaterale Verbindungen vor.

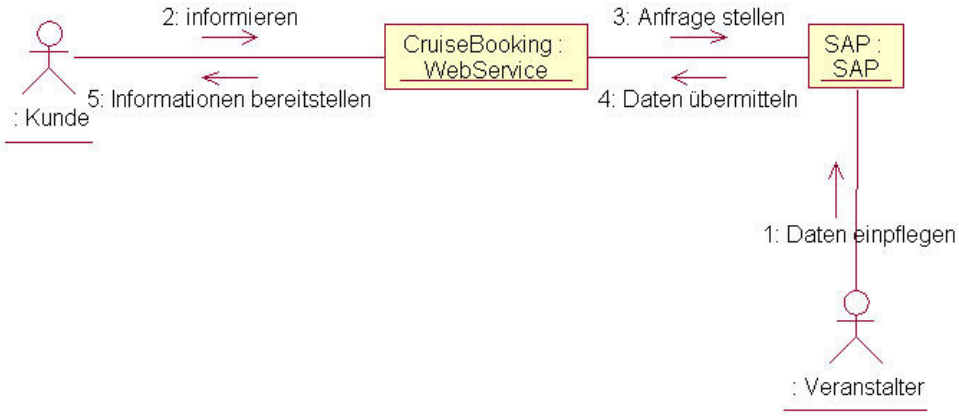


Abbildung 31.6.: Web Service: KollSearch

Kollaborationsdiagramm zum Buchen von Kreuzfahrten Die Beziehung Veranstalter – SAP System ist auch hier eine einseitige Beziehung, da – wie bereits oben erwähnt – der Veranstalter lediglich Daten an das SAP System einträgt.

Folgende Objekte fungieren untereinander sowohl als Sender und Empfänger:

- Kunde – Web Service CruiseBooking
- Web Service CruiseBooking – Web Service Kreditkartenprüfung
- Web Service CruiseBooking – SAP

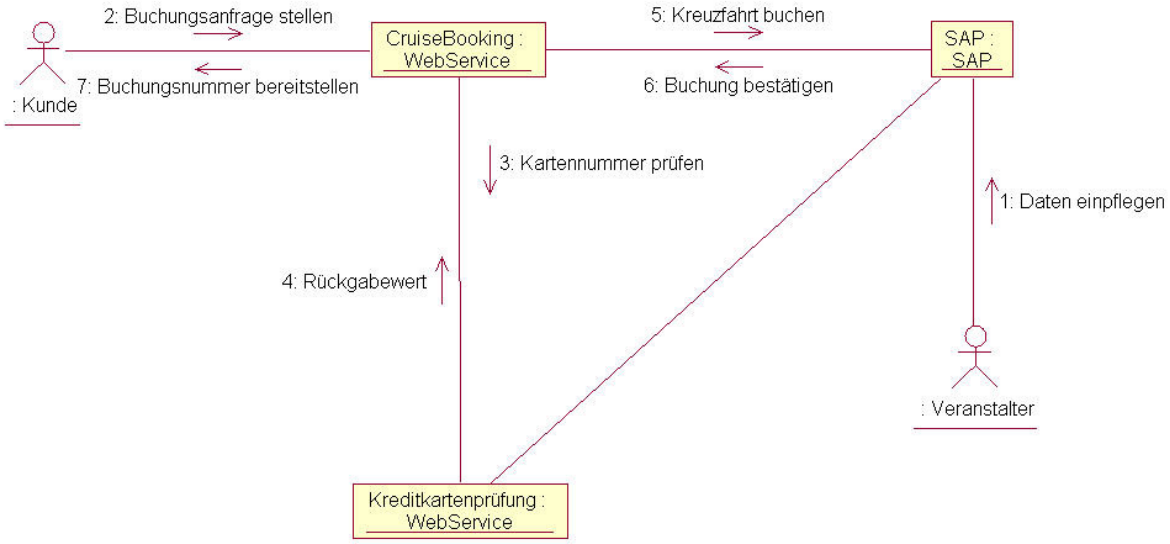


Abbildung 31.7.: Web Service: KollBooking

31.3. Die Realisierung des Web Services

Im Folgenden soll die Realisierung des Web Services und die damit implementierte Klassenstruktur aufgezeigt werden. Dabei wird zunächst auf die Beanklasse CruiseBean eingegangen, bevor danach die Providerklasse des Web Services mit ihren Methoden erläutert wird. Die dazugehörigen Fehlerklassen werden im darauf folgenden Abschnitt erklärt. Abschließend wird das Deployment des entwickelten Web Services mit einem Web Service Deployment Discriptor (WSDD) vorgenommen.

31.3.1. Die Beanklasse CruiseBean

Die Beanklasse repräsentiert einen zusammengesetzten, komplexen Datentypen. Sie dient der Abbildung des Abfrageergebnisses aus dem SAP R/3–System. Im Detail umfasst ein `CruiseBean` folgende Objekte:

```
...
private java.lang.Integer mp_reiseID;
  private String mp_anbieter;
  private String mp_schiff;
  private String mp_region;
  private String mp_basishafen;
  private String mp_zielhafen;
  private Date mp_reiseabfahrt;
  private Date mp_reiseankunft;
  private java.lang.Float mp_preis;
...
```

Die Instantiierung von CruiseBeans erfolgt mit dem Konstruktor `CruiseBean()`:

```
...
public CruiseBean(java.lang.Integer reiseID,
                  String anbieter,
                  String schiff,
                  String region,
                  String basishafen,
                  String zielhafen,
                  Date reiseabfahrt,
                  Date reiseankunft,
                  java.lang.Float preis) {
           this.mp_reiseID = reiseID;
           this.mp_anbieter = anbieter;
           this.mp_schiff = schiff;
           this.mp_region = region;
           this.mp_basishafen = basishafen;
           this.mp_zielhafen = zielhafen;
           this.mp_reiseabfahrt = reiseabfahrt;
           this.mp_reiseankunft = reiseankunft;
           this.mp_preis = preis;
}
...
```

Zum Auslesen und Speichern der `private`–Variablen wurden `getter`– bzw. `setter`–Methoden implementiert:

Variable	getter–Methode	setter–Methode
`mp_reiseID`	`getReiseID()`	`setReiseID(java.lang.Integer reiseID)`
`mp_anbieter`	`getAnbieter()`	`setAnbieter(String anbieter)`
`mp_schiff`	`getSchiff()`	`setSchiff(String schiff)`
`mp_region`	`getRegion()`	`setRegion(String region)`
`mp_basishafen`	`getBasishafen()`	`setBasishafen(String basishafen)`
`mp_zielhafen`	`getZielhafen()`	`setZielhafen(String zielhafen)`
`mp_reiseabfahrt`	`getReiseabfahrt()`	`setReiseabfahrt(Date reiseabfahrt)`
`this.mp_reiseankunft`	`getReiseankunft()`	`setReiseankunft(Date reiseankunft)`
`this.mp_preis`	`getPreis()`	`setPreis(java.lang.Float preis)`

Tabelle 31.1.: Web Service: Getter– und Setter–Methoden

31.3.2. Die Providerklasse CruiseBooking

Die Java–Klasse CruiseBooking ist Bestandteil des Packages `de.bizweb.cruiseservice` und implementiert die Providerklasse des Web Service. Mit ihr wird die (Web-)Schnittstelle zwischen dem Applikationsserver und dem Client zur Verfügung gestellt. Im Rahmen des BizWeb–Projektes wurde der Applikationsserver SAP R/3 und ein Servlet im Cocoon–Portal als Client–Anwendung eingesetzt. Die Client–Anfragen basieren auf SOAP–Requests, die über die Apache–Axis–Technologie an die Providerklasse herangetragen werden. Die Antworten, resp. SOAP–Resonses, werden ebenfalls über die „Vermittlungsstelle“ Apache–Axis an den Client zurückgegeben.

Im Wesentlichen sind die Aufgaben der Providerklasse,

- eine Liste von Kreuzfahrten zu bestimmten Selektionskriterien zurückzuliefern (`getList()`–Methoden) und
- Kreuzfahrten zu buchen (`booking()`–Methode).

Für beide Aufgaben ist jeweils eine Verbindung zum R/3–System erforderlich. Eine entsprechende Instanz des SAP JConnectors wird mit der Variable `mp_jcoInstance` vom Typ `RFCReadTable` des Packages `de.bizweb.sap.jco` deklariert. Der Verbindungsaufbau erfolgt schließlich mit der Methode `getJcoInstance()`:

```
1 ...
2 private RFCReadTable mp_jcoInstance = null;
3
4 public void getJcoInstance(){
5   if (mp_jcoInstance == null){
6     mp_jcoInstance = new RFCReadTable();
7   }
8 }
9 ...
```

Die getList()–Methoden

Diese Methoden liefern Kreuzfahrtlisten in Form von `CruiseBean`–Arrays zurück. Insgesamt wurden fünf Methoden implementiert, die jeweils mit folgenden Parametern überladen werden können:

Methode	Funktionalität
`getList ()`	Liefert die Gesamtliste aller vorhandenen Kreuzfahrten zurück.
`getList (String schiffsname)`	Liefert zu einem Schiffnamen eine Liste aller Kreuzfahrten zurück.
`getList (String schiffsname, String regionname)`	Liefert zu einem Schiffnamen und einer Region eine Liste aller Kreuzfahrten zurück.
`getList (String schiffsname, String basishafen, Date reiseabfahrt)`	Liefert zu einem Schiffnamen, einem Abfahrtshafen und einem Abfahrtsdatum eine Liste aller Kreuzfahrten zurück.
`getList (String anbietername, String basishafen, String zielhafen, Date reiseabfahrt)`	Liefert zu einem Anbieter, einem Abfahrts– und Zielhafen und einem Abfahrtsdatum eine Liste aller Kreuzfahrten zurück.

Tabelle 31.2.: Web Service: getList–Methoden und deren Parameter

Alle Methoden haben ein entsprechendes Pendant in der JConnector–Klasse `RFCReadTable`. Mit `getJcoInstance()` wird eine Verbindungsinstanz zum R/3–System hergestellt. Alle öffentlichen Methoden des `RFCReadTable`–Objektes, `mp_jcoInstance`, können somit nach der Instantiierung genutzt werden.

Die `getSAPList()`–Methoden liefern die (ggf. durch Parameter eingeschränkte) Kreuzfahrtliste aus dem SAP–System zurück.

Methode	für CruiseBooking	Methode für	RFCReadTable
`getList`	`()`	`getSAPList`	`()`
`getList`	`(String schiffsname)`	`getSAPList`	`(String schiffsname)`
`getList`	`(String schiffsname, String regionname)`	`getSAPList`	`(String schiffsname, String regionname)`
`getList`	`(String schiffsname, String basishafen, Date reiseabfahrt)`	`getSAPList`	`(String schiffsname, String basishafen, Date reiseabfahrt)`
`getList`	`(String anbietername, String basishafen, String zielhafen, Date reiseabfahrt)`	`getSAPList`	`(String anbietername, String basishafen, String zielhafen, Date reiseabfahrt)`

Tabelle 31.3.: Web Service: getSAPList– und RFCReadTable–Methoden

Die zurückgelieferte Datenstruktur ist eine ArrayList. Sie muss wiederum für den SOAP–Transport in ein `CruiseBean`–Array umgewandelt werden. Hierfür wurde die Konvertie-

rungsfunktion `arrayList2CruiseBeanArray()` implementiert, die eine ArrayList übergeben bekommt und diese in ein `CruiseBean`–Array aufbereitet. Ist das Abfrageergebnis die leere Menge (`null`), so wird eine `NoSuchCruiseException` mit dem entsprechendem Hinweis geworfen.

```
1  ...
2  private CruiseBean[] arrayList2CruiseBeanArray(ArrayList arrayList)
3        throws NoSuchCruiseException{
4
5    if (arrayList == null){
6      throw new NoSuchCruiseException("Leider konnten keine Kreuzfahrten "
7                                      + "zu diesen Selektionskriterien "
8                                      + "ermittelt werden.");
9    }
10
11   Iterator it = arrayList.listIterator();
12
13   //je 9 Elemente der ArrayList ergeben 1 CruiseBean.
14   CruiseBean[] resConvert = new CruiseBean[arrayList.size()/9];
15
16   int counter = 0;
17   while (it.hasNext()){
18     CruiseBean elem = new CruiseBean((java.lang.Integer)it.next(),
19                                      (String)it.next(),
20                                      (String)it.next(),
21                                      (String)it.next(),
22                                      (String)it.next(),
23                                      (String)it.next(),
24                                      (Date)it.next(),
25                                      (Date)it.next(),
26                                      (java.lang.Float)it.next());
27     resConvert[counter] = elem;
28     counter++;
29   }
30
31   return resConvert;
32 }
33 ...
```

Zunächst wird die erforderliche Array–Größe ermittelt und ein Array vom Typ `CruiseBean` wird instantiiert. Die Größe ist abhängig von der Anzahl der Datentypen, die das `CruiseBean` repräsentiert. Änderungen, die in diesem Zusammenhang an der `CruiseBean`–Klasse vorgenommen werden (insbesondere das Hinzufügen oder Löschen eines Elementes) müssen zwangsläufig an dieser Stelle ebenfalls berücksichtigt werden.
Danach wird die ArrayList wird mit einem Iterator durchlaufen und die Objekte der ArrayList in ein `CruiseBean`–Element kopiert. Die neu angelegten `CruiseBean`–Elemente werden dem `CruiseBean`–Array hinzugefügt und an die anfragende `getList()`–Methode zurückgegeben.

Abschließend übergibt die `getList()`–Methode das `CruiseBean`–Array an Axis, wo es dann für die SOAP–Response–Nachricht aufbereitet wird.

Die booking()–Methode

Mit dieser Methode werden Kreuzfahrten gebucht. Als Parameter erwartet `booking()` die Kunden– und Reisenummer, sowie die Kreditkarteninformationen Kreditkartennummer und Gültigkeit (in Form der Integer–Werte Jahr und Monat).

```
1  public int booking (int kundenID,
2                      int reiseID,
3                      String kknr,
4                      int year,
5                      int month) throws Exception{
6
7    //Einbindung des WebServices zur Kreditkartenprüfung
8    KKpruefungService kkPruefungService = new KKpruefungServiceLocator();
9    KKpruefung kkPruefung = kkPruefungService.getKKpruefung();
10
11   //Prüfung der KK-Informationen
12   if (kkPruefung.pruefung(kknr, year, month)){
13     getJcoInstance();
14     int buchungsID = mp_jcoInstance.bookingSAP(kundenID,
15                                                reiseID);
16     // Keine Buchung erfolgt, wenn die buchungsID = -1
17     if (buchungsID == -1){
18       throw new BookingException("Bei der Buchung ist ein Fehler "
19                                  + "aufgetreten. "
20                                  + "Es konnte keine gültige Buchungsnummer "
21                                  + "erstellt werden.");
22     }
23     return buchungsID;
24   } else {
25   //fehlerhafte Kreditkarteninformationen
26   throw new BookingException("Ihre eingegebenen Daten der Kreditkarte "
27                              + "sind fehlerhaft. Bitte überprüfen Sie "
28                              + "Ihre Eingaben.");
29   }
30 }
```

Die Kreditkartenprüfung ist ein eigener (ausgelagerter) Web Service. Nur, wenn die boolsche Methode `pruefung(String kknr, int year, int month)` den Wert `TRUE` zurückliefert, darf eine Buchung im R/3–System erfolgen. Bei fehlerhaften Kreditkarteninformationen wird eine `BookingException` geworfen. Auf die Verschlüsselung dieser sensiblen Daten soll im Rahmen dieses Projektes nicht weiter eingegangen werden.

Erfolgreiche Buchungen werden mit einer zurückgelieferten Buchungsnummer (Integer–Wert) bestätigt. Fehlerhafte Buchungen werden mit der fiktiven Buchungsnummer „`-1`“ abgefangen und eine entsprechende `BookingException` mit Hinweis geworfen.

31.3.3. Die Fehlerklassen (abgeleitet von RMI – Exception)

Unsere implementierten Klassen sind von JAVA Remote Method Invokation (JAVA–RMI) abgeleitet, diese sind für den Einsatz von verteilten Systemen zu verwenden. Damit kann ein Programm X auf Rechner A eine Methode X1 auf einem anderen Rechner B aufrufen und dabei evtl. auch Parameter übergeben. Im Folgenden werden die zwei Fehlerklassen erläutert.

NoSuchCruiseExeption.java

Diese Fehlerklasse wirft eine Exception, wenn für die ausgewählten Selektionskriterien keine Ergebnisse ermittelt werden können (leere Menge). Die abgeleitete Fehlerklasse übergibt die entsprechende Fehlermeldung in Form eines Strings an die Klasse `CruiseBooking` und dort an die entsprechende `getList()`–Methode.

```
1 package de.bizweb.cruiseservice.exception;
2 import java.rmi.*;
3
4 public class NoSuchCruiseException extends RemoteException {
5   public NoSuchCruiseException (String message) {
6     super (message);
7   }
8 }
```

BookingException.java

Konnte keine Buchung im SAP System vorgenommen werden, so wirft diese Fehlerklasse eine Exception. Analog zur `NoSuchCruiseException` ist diese Klasse auch von JAVA–RMI abgeleitet und übergibt die Fehlermeldung an die `booking()`–Methode, die sich ebenfalls in der Klasse `CruiseBooking` befindet.

```
1 package de.bizweb.cruiseservice.exception;
2 import java.rmi.RemoteException;
3
4 public class BookingException extends RemoteException{
5   public BookingException(String message){
6     super(message);
7   }
8 }
```

31.3.4. Das Deployment des Web Services

Bevor das Deployment vorgenommen werden kann, sind zunächst einige Vorarbeiten erforderlich. Die Packages

- `de.bizweb.cruiseservice`,
- `de.bizweb.cruiseservice.exception` und
- `de.bizweb.sap.jco`

sind auf dem BizWeb–Server in folgendes Verzeichnis zu kopieren und danach zu kompilieren: `$TOMCAT_HOME/webapps/axis/WEB-INF/classes`.

Ein Autodeployment ist hier aufgrund der Komplexität der Anwendung nicht möglich. Mit einer Web Service–Deployment–Deskriptor (kurz WSDD) Datei werden alle erforderlichen Informationen über den WebService erfasst. Für den CruiseBooking–Service wurde die Datei `deploy.wsdd` erstellt:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="CruiseBooking" provider="java:RPC">
    <parameter name="className"
               value="de.bizweb.cruiseservice.CruiseBooking"/>
    <parameter name="allowedMethods" value="*"/>

    <typeMapping qname="myNS:CruiseBean"
       xmlns:myNS="http://cruiseservice.bizweb.de"
       languageSpecificType="java:de.bizweb.cruiseservice.CruiseBean"
       serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
       deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </service>
</deployment>
```

Hier sind Informationen über den Service–Namen, Providertyp und Package– und Klassennamen enthalten. Die Angabe `allowedMethods` kann zusätzlich genutzt werden, um die Verwendung öffentlicher Methoden noch einmal einzuschränken – der Wert „*“ ermöglicht die Nutzung aller öffentlichen Methoden. Ein weiteres Element ist das `typeMapping`, das zur Serialisierung und Deserialisierung der Bean–Klasse `CruiseBean` dient. Diese Angaben nutzt Axis, um das Java–Bean in das XML–konforme SOAP zu konvertieren.

Das Deployment der wsdd–Datei kann jetzt mit der Konsolenanweisung
`java org.apache.axis.client.AdminClient -p80 deploy.wsdd`
auf dem Port 80 (`-p80`) erfolgen.

Unter der Adresse http://195.37.183.100/axis/servlet/AxisServlet kann man sich eine Liste der deployten Services und deren öffentliche (und freigegebenen) Methoden ansehen:

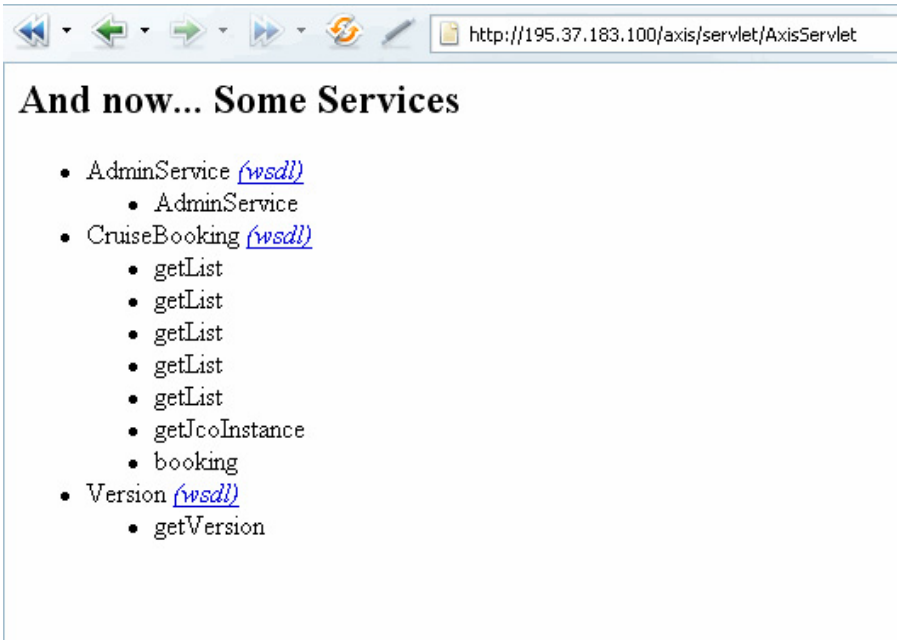


Abbildung 31.8.: Web Service: Deployments

31.4. Beispielsitzungen mit einem Servlet

31.4.1. Beispiel zur Suche von Kreuzfahrten

Nach Aufruf des Servlets wird Benutzer gebeten entsprechende Suchkriterien einzugeben. In der folgenden Abbildung wurde keine Auswahl getätigt, worauf der Benutzer nochmals gebeten wird, eine Eingabe zu tätigen.

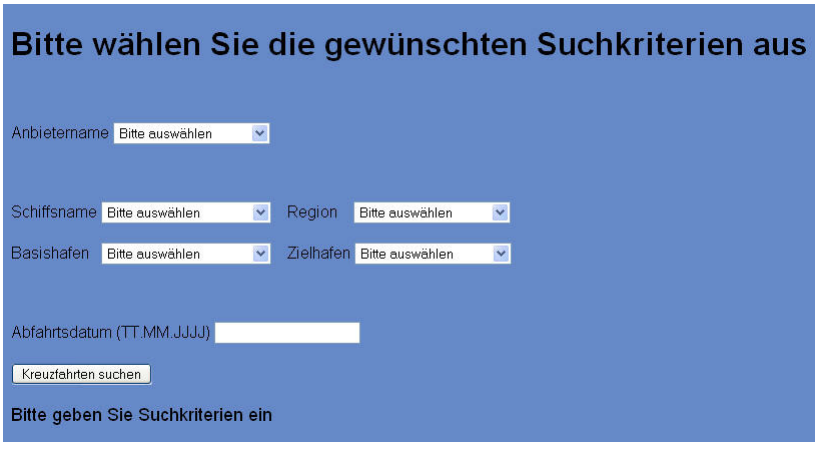


Abbildung 31.9.: Web Service: Suche ohne Parameter

In diesem Beispiel sucht der Benutzer eine Kreuzfahrt auf der AIDA Cara und der Region Mittelamerika. Es werden zwei gültige Ergebnisse in Tabellenform ausgegeben.

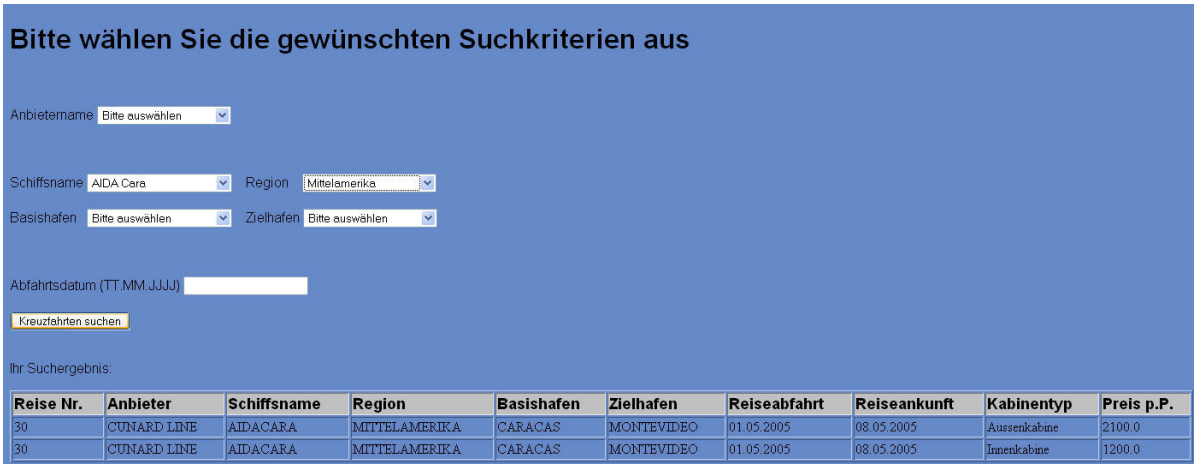


Abbildung 31.10.: Web Service: Suche Schiffsname & Parameter

Wird vom Benutzer eine ungültige Kombination von Suchkriterien ausgewählt, wird vom System darauf hingewiesen, dass diese Kombination nicht möglich, mit der Option sich alle Kreuzfahrten anzeigen zulassen.

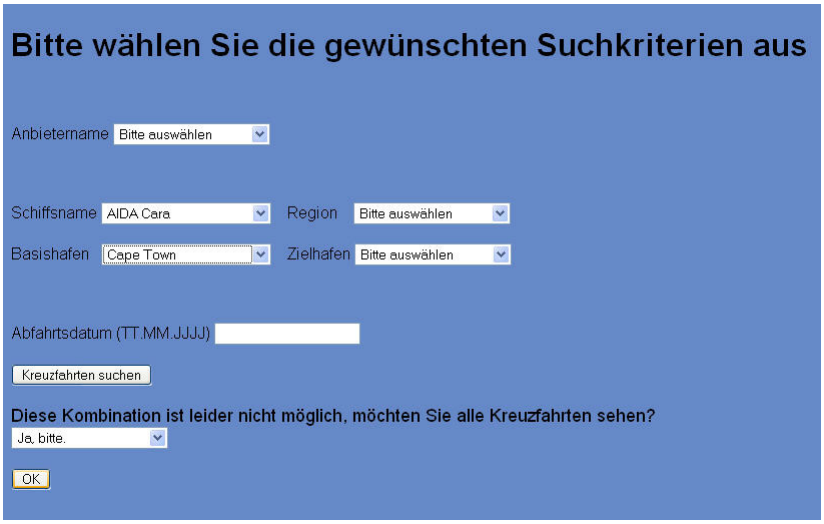


Abbildung 31.11.: Web Service: Suche ungültige Kombi

Wurde von dieser oben genannten Option Gebrauch gemacht bekommt der Benutzer die Liste alle Kreuzfahrten ausgegeben.

Bitte wählen Sie die gewünschten Suchkriterien aus

Anbietername: Bitte auswählen

Schiffsname: Bitte auswählen Region: Bitte auswählen

Basishafen: Bitte auswählen Zielhafen: Bitte auswählen

Abfahrtsdatum (TT.MM.JJJJ):

Kreuzfahrten suchen

Liste aller möglichen Kreuzfahrten:

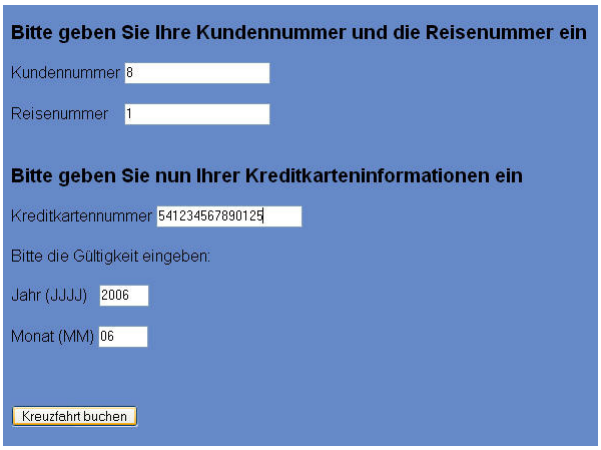
Ihr Suchergebnis:

Reise Nr.	Anbieter	Schiffsname	Region	Basishafen	Zielhafen	Reiseabfahrt	Reiseankunft	Kabinentyp	Preis p.P.
31	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	CELEBRITY CRUISES GALAXY	MITTELAMERIKA	CAYENNE	LA PLATA	02.01.2005	16.01.2005	Aussenkabine	2100.0
32	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	CELEBRITY CRUISES GALAXY	MITTELAMERIKA	CAYENNE	LA PLATA	12.03.2005	26.03.2005	Aussenkabine	2100.0
33	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	COSTA EUROPA	MITTELAMERIKA	LIMA	VALPARAISO	12.03.2005	26.03.2005	Aussenkabine	2100.0
34	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	COSTA EUROPA	MITTELAMERIKA	LIMA	VALPARAISO	31.03.2005	08.04.2005	Aussenkabine	2100.0
35	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	COSTA ROMANTICA	MITTELAMERIKA	LIMA	BUENOS AIRES	31.03.2005	08.04.2005	Aussenkabine	2100.0
36	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	COSTA ROMANTICA	MITTELAMERIKA	LIMA	BUENOS AIRES	04.07.2005	24.04.2005	Aussenkabine	2100.0
14	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	MS BERLIN	HAWAII	LOS ANGELES	HONULULU	17.09.2005	24.09.2005	Aussenkabine	2100.0

Abbildung 31.12.: Web Service: Suche alle Kreuzfahrten

31.4.2. Beispiel zur Buchung von Kreuzfahrten

Der Kunde wird gebeten seine Kunden–, Reise– und eine gültige Kreditkartennummer einzugeben.



Bitte geben Sie Ihre Kundennummer und die Reisenummer ein

Kundennummer: 8

Reisenummer: 1

Bitte geben Sie nun Ihrer Kreditkarteninformationen ein

Kreditkartennummer: 541234567890125

Bitte die Gültigkeit eingeben:

Jahr (JJJJ): 2006

Monat (MM): 06

Kreuzfahrt buchen

Abbildung 31.13.: Web Service: Buchung (1)

Sind alle Angaben vollständig und gültig eingegeben worden, so erhält der Kunde eine Buchungsbestätigung in Form einer Buchungsnummer.

Abbildung 31.14.: Web Service: Buchung (2)

In dieser Abbildung ist zwar eine gültige Kreditkartennummer eingegeben worden, jedoch ist die vorhandene Kundennummer nicht beim Veranstalter registriert und damit ungültig. Der Kunde wird in Form einer Fehlermeldung darauf hingewiesen.

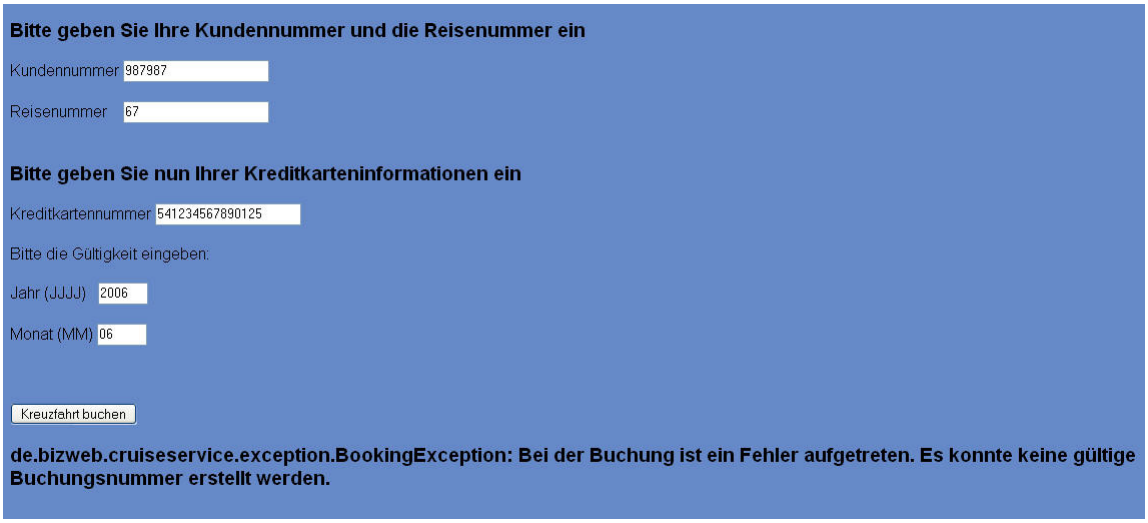


Abbildung 31.15.: Web Service: Buchung (3)

Sind alle Eingaben bis auf die Kreditkartennummer korrekt, so gibt der Web Service Kreditkartenprüfung eine Fehlermeldung zurück.

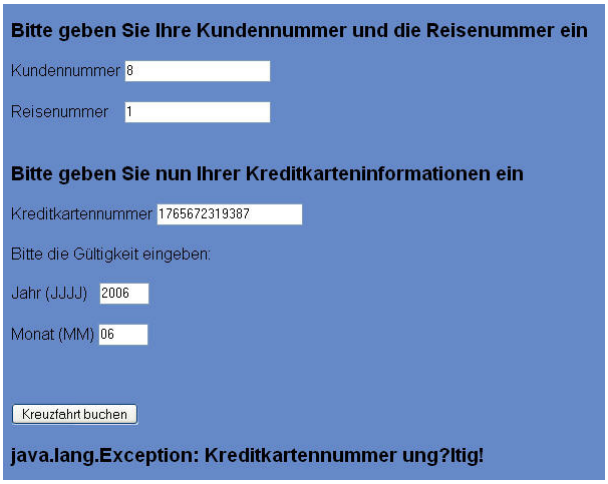


Abbildung 31.16.: Web Service: Buchung falsche KkNr

32. Intermediary

32.1. Einleitung

Zu den eigentlichen Prinzipien der Web Service-Technologie gehören die Intermediaries, sprich die Zwischenstationen, über welche der ursprüngliche Web Service mit weiteren kommunizieren kann.

Eine Nachricht, die zwischen zwei Webkomponenten (Sender und Empfänger) ausgetauscht wird, kann auf ihrem Weg von SOAP-Knoten weitergeleitet werden. Solche als SOAP-Intermediaries bezeichneten SOAP-Knoten haben sowohl die Rolle des SOAP-Senders als auch des SOAP-Empfängers. Ein SOAP-Intermediary kann selbst Teile der Nachricht verarbeiten oder auch eine Nachricht umkodieren. Der Weg der SOAP-Nachricht vom ursprünglichen SOAP-Sender über potentiell mehrere SOAP-Intermediaries zum endgültigen SOAP-Empfänger wird als SOAP-Message-Path bezeichnet.

Abbildung 32.1 zeigt die einfachste Form der Web-Service-Kommunikation, basierend auf dem Sender-/Empfänger- bzw. Anfrage-/Antwortprinzip.

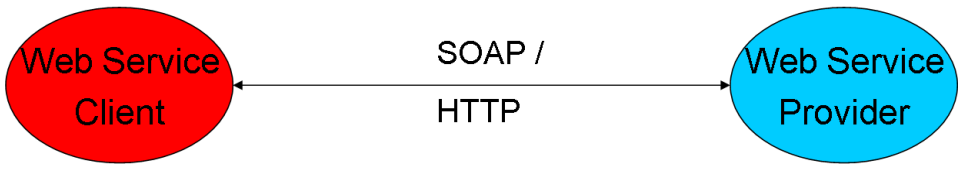


Abbildung 32.1.: Intermediary: Beispiel 1

Dieser Ablauf wird nun um eine weitere Station ergänzt:

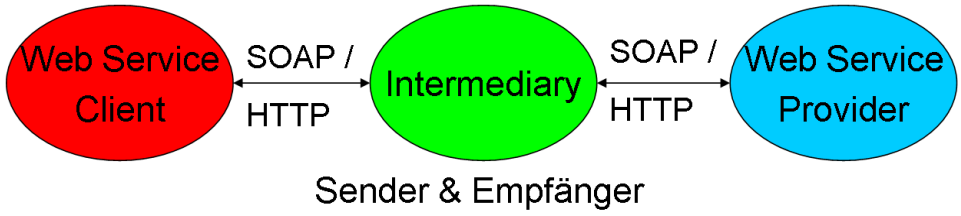


Abbildung 32.2.: Intermediary: Beispiel 2

Dieses Prinzip kann je nach Bedarf des Service und gewünschten Entwicklungsaufwand beliebig erweitert werden, aufgezeigt in Abbildung 32.3.

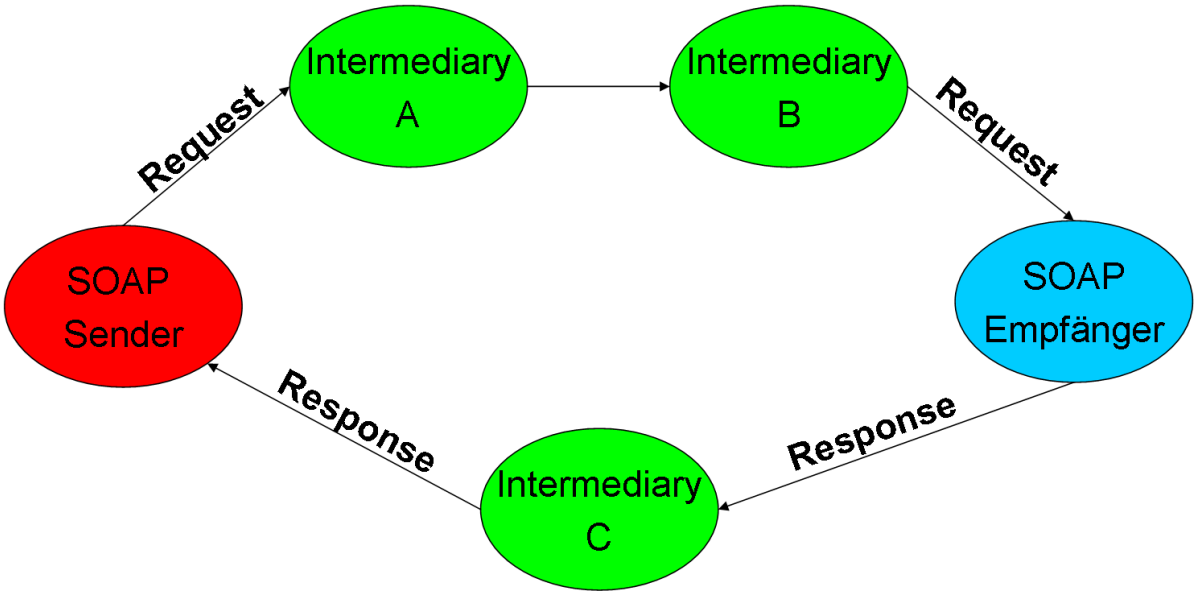


Abbildung 32.3.: Intermediary: Beispiel 2

Ein Ziel des BizWeb-Web Service ist es , um das Intermediary-Prinzip zu erläutern, eine Zwischenstation zu implementieren. Zur Verdeutlichung der Verteilung der Web Services wurde eine Kooperation mit dem Alfred-Wegener-Institut für Polar und Meeresforschung in Bremerhaven eingegangen. Auf einer deren Servern läuft ein weiterer Apache Tomcat-Server (Version 5.0.28), ausgerüstet mit Axis (Version 1.1).

Auf diesem Server befindet sich der bereits erwähnte Web Service zur Kreditkartenprüfung während der Buchung der Kreuzfahrt.

32.2. Prinzipieller Ablauf

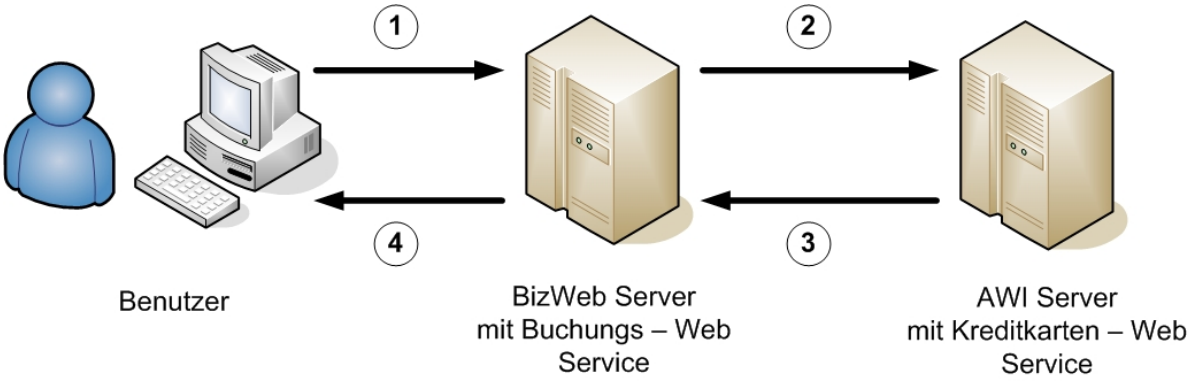


Abbildung 32.4.: Intermediary: Ablauf

1. Der Benutzer wählt eine Kreuzfahrt auf und wünscht diese zu buchen. Die Anfrage wird an den BizWeb-Server weitergeleitet und zum größtenteil von dem dort befindlichen Web Service abgearbeitet. Dieser Web Service enthält eine Methode, welche die Kreditkartenprüfung aufruft

2. Nun werden die Kreditkartendaten an den prüfenden Web Service gesendet. Wie später noch erläutert wird, besteht die Prüfung aus zwei Teilen, welche zum Einen die Kreditkartennummer algorithmisch berechnet und zum Anderen das Datum mit der Systemzeit des Servers abgleicht.

3. Gleich ob das Ergebnis positiv oder negativ ausfiel, es wird eine Antwort an den BizWeb-Web Service gesendet.

4. Dieser gibt nun entweder eine Buchungs-ID zurück (was einer Bestätigung gleichzusetzen ist) oder gibt eine Fehlermeldung aus bzw. liefert diese dem Benutzer zurück, daß die Buchung auf Grund eines Fehlers bei der Eingabe oder der Ungültigkeit der Kreditkartendaten nicht durchgeführt wurde.

32.3. Ereignisgesteuerte Prozesskette

Eine weitere Möglichkeit zur theoretischen Darstellung des Ablaufs ist die ereignisgesteuerte Prozesskette:

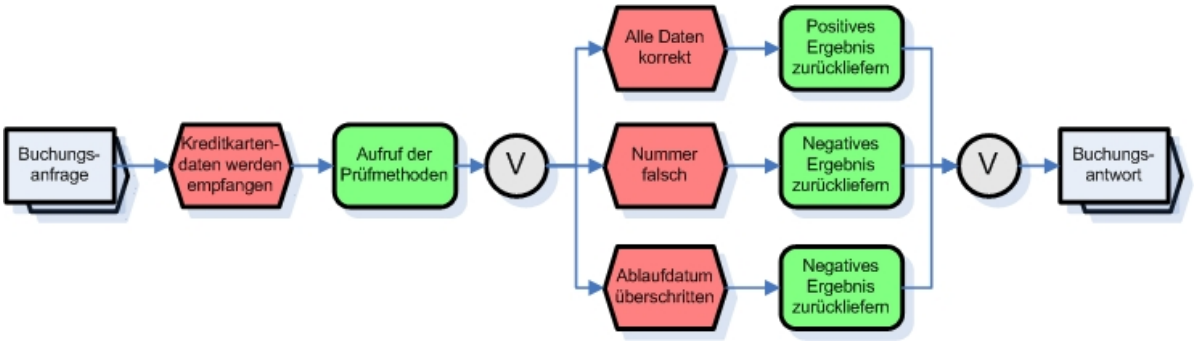


Abbildung 32.5.: Intermediary: EPK

Nach der Übermittlung der zu prüfenden Daten gibt es drei Möglichkeiten, welche eintreten können:

1. Alle Daten sind korrekt – das Prüfergebnis ist positiv.
2. Die Kreditkartennummer ist falsch – das Prüfergebnis ist negativ.
3. Das Gültigkeitsdatum ist abgelaufen – das Prüfergebnis ist negativ.

Der vierte Fall, der theoretisch eintreten könnte, nämlich das Nummer *und* Datum falsch sind, wird nicht abgebildet, da bereits einer der anderen Fälle (2. bzw. 3.) eingetreten ist und das Ergebnis so schon negativ ausfällt.

32.4. Erläuterungen zum Quelltext

Der Web Service zur Überprüfung der Kreditkartendaten besteht aus zwei Methoden, welche nun näher erläutert werden.

```
1  private static boolean mod10validation(String numberToCheck) {
2    int nulOffset = '0';
3    int sum = 0;
4    for (int i = 1; i <= numberToCheck.length(); i++) {
5        int currentDigit = numberToCheck.charAt(numberToCheck.length() - i)
6                - nulOffset;
7        if ((i % 2) == 0) {
8            currentDigit *= 2;
9            currentDigit = currentDigit > 9
10                   ? currentDigit - 9
11                   : currentDigit;
12           sum += currentDigit;
13       } else {
14           sum += currentDigit;
```

```
15          }
16      }
17      if (!((sum % 10) == 0))
18          return false;
19      else
20          return true;
21  }
```

Quelltext 32.1: Intermediary: Validierung der Kreditkartenummer

Die Methode *mod10validation()* ist der erste Teil des Web Service und prüft die Nummer. Der Algorithmus wurde mit Hilfe einer Exceltabelle und den mathematischen Angaben der Seite http://www.beachnet.com/~hstiles/cardtype.html entwickelt. Je nachdem, ob die Berechnungen innerhalb der Methode richtig oder falsch sind, wird *true* oder *false* zurückgegeben.

```
1  private static boolean datumPruefung(int year, int month)
2          throws Exception {
3      int day;
4      if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8
5              || month == 10 || month == 12) {
6          day = 31;
7      } else if (month == 2) {
8          day = 28;
9      } else {
10         day = 30;
11     }
12     GregorianCalendar dateExpire = new GregorianCalendar(year, month, day);
13     GregorianCalendar dateToday = new GregorianCalendar();
14     long difference = dateExpire.getTimeInMillis()
15             - dateToday.getTimeInMillis();
16     dateToday.setTimeInMillis(difference);
17     if (difference < 0) {
18         return false;
19     } else {
20         return true;
21     }
22 }
```

Quelltext 32.2: Intermediary: Abgleich des Datums

Die Methode zur Überprüfung des Gültigkeitdatums bekommt das Jahr und den Monat übergeben, der Tag wird daraufhin gesetzt. Dann folgt ein Abgleich mit der Systemzeit. Ist diese Differenz kleiner Null, ist die Karte abgelaufen und es wird *false* zurückgegeben; ist sie größer Null, liefert die Methode *true* zurück.

33. Portal

33.1. Einleitung

33.1.1. Projektzusammenhang

Das Portal des Projektes BizWeb dient zum einen als Dokumentations- und Informationsseite, zum anderen als Präsentationsmedium der Web Services. Es verwendet die in der Schulungsphase des Projektes vermittelten Technologien und Werkzeuge wie XML, XSL, CSS, Apache Tomcat, sowie die Programmiersprachen Java und Java Script. Mit dieser Ausarbeitung soll dem außenstehenden Leser die sehr komplexe Thematik einer Portalentwicklung im Rahmen eines kleinen Projektberichtes vermittelt werden. Gesagt sei allerdings, dass es sich hierbei nur um ausgewählte Beispiele handelt, da eine gesamte Niederlegung aller Quelltexte und deren Dokumentation ein ganzes Buch füllen würde.

33.1.2. Entwicklungsumgebung

Als Entwicklungsumgebung wurde sich für das Cocoon Framework von Apache entschieden. Dieses Framework bietet eine Menge von Werkzeugen, die für die Entwicklung von Web-Applikationen wichtig sind. Im folgenden Kapitel dieser Ausarbeitung wird auf das Grundkonzept und die Arbeitsweise von Cocoon eingegangen, bevor das eigentliche Portal und die hierfür von Apache bereitgestellte und hier verwendete Portal-Engine erläutert wird.

33.1.3. Zeitplanung

Das Teilprojekt „Portal“ wurde am 4. November 2004 gestartet (Siehe Abb. 33.1). Grundlegender Aufbau und Funktionalitäten des Portals wurden zu Beginn festgelegt. Während einer einmonatigen Einarbeitungs- und Schulungsphase wurde ein erster lauffähiger Prototyp des Portalgerüstes mit den vorher definierten Farben und Schriften entwickelt.

Das Portal mit den einzelnen Rubriken (siehe Kapitel 5) wurde anschließend in einer einmonatigen Realisierungsphase fertig gestellt. Zum Ende hin wurde der Kreuzfahrt-Webservice unter der Rubrik „Kreuzfahrt“ integriert. Der Versuch, mittels Cocoon Flow

eine in Cocoon direkt integrierte Anbindung an den Web Service zu realisieren, ist in Kapitel 33.7 näher beschrieben.

Der Abschluss des Teilprojektes erfolgt mit der Integration der Gesamtdokumentation in das Portal.

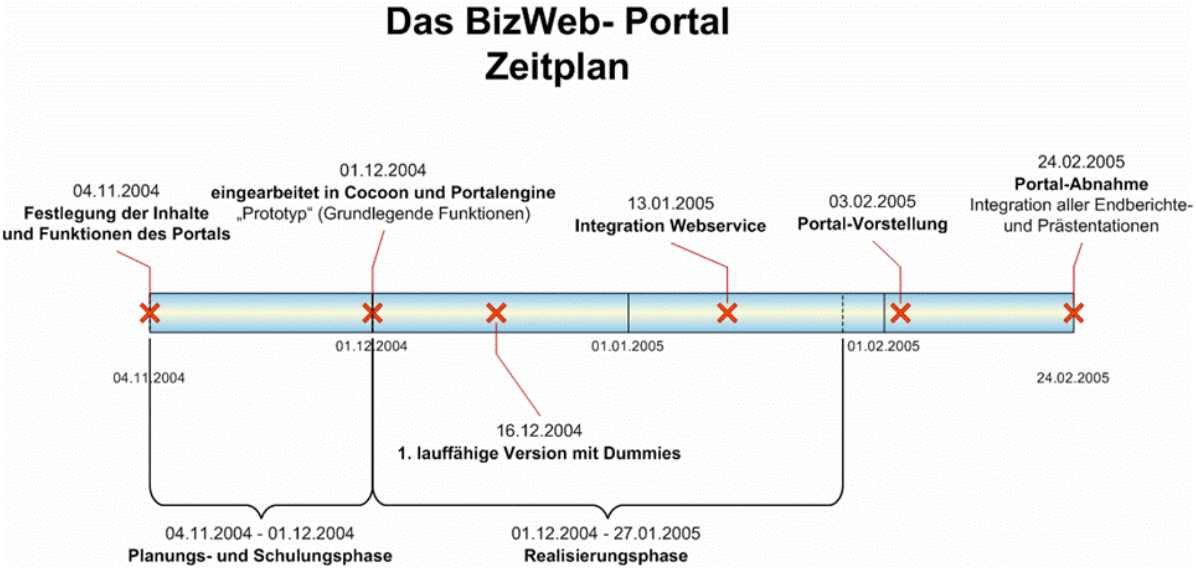


Abbildung 33.1.: Portal: Zeitplan

33.2. Das Apache Cocoon Framework

33.2.1. Cocoon kurzgefasst

Cocoon ist in erster Linie ein XML-Publishing System. Es ist in der Lage, aus einem oder mehreren XML-Dokument(en) und dem zugehörigen XSL-Dokument(en) ein beliebiges Ausgabeformat zu produzieren und an beliebige Endgeräte zu versenden. Da XML/XSL basierte Lösungen immer mehr Zuspruch finden, jedoch die meisten Endgeräte noch nicht in der Lage sind, diese Formate zu lesen, dient Cocoon als „Übersetzer“ in die jeweilige „Sprache“ des Empfängers. Cocoon ist z.B. sogar in der Lage, das Gerät, das Daten anfordert, zu erkennen und ihm das passende Dokument zu generieren und anschließend zuzusenden. Als Ausgabeformate seien an dieser Stelle z.B. HTML, WAP, PDF oder RTF aufgelistet, um nur einige zu nennen. Außerdem definiert Cocoon explizit den Workflow, den ein XML-Dokument zu durchlaufen hat, bevor es auf die Reise zum Endgerät gehen darf. Dieser Workflow ist standardisiert.

33.2.2. Funktionalitäten

Im Laufe der letzten Jahre sind die Anforderungen an Web-Applikationen sehr stark gewachsen. Im Folgenden werden diese Anforderungen kurz erläutert, da sie in ihrer Gesamtheit die einzelnen Funktionalitäten und Vorteile des Apache Cocoon Frameworks darstellen.

Trennung von Layout, Inhalt und Logik
Cocoon trennt Daten, Layout und Logik (Abbildung 33.2). Dieses Konzept wird in der modernen Softwarenentwicklung schon seit einiger Zeit erfolgreich eingesetzt und ist allgemein als MVC (Model View Controller) Architektur bekannt. Dadurch ist es möglich, dass die verschiedenen Entwickler getrennt an ein und demselben Projekt arbeiten können.

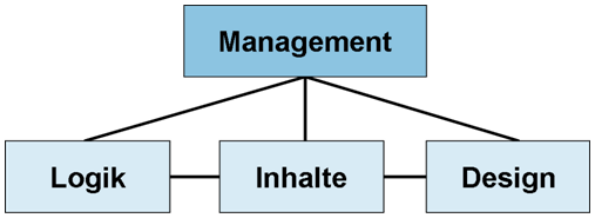


Abbildung 33.2.: Portal: Trennung von Logik, Inhalt und Design

Plattformunabhängigkeit
Darunter versteht man, dass eine Anwendung nicht an ein bestimmtes Betriebssystem gebunden ist. Da Cocoon vollständig in Java geschrieben ist, ist eine Lauffähigkeit auf jedem Betriebssystem mit einer aktuellen Java-Laufzeitumgebung gegeben. Der Focus ist dabei natürlich besonders auf Unix, Linux und Windows gerichtet.

Personalisierung
Gerade auf kommerziellen Seiten im Internet spielt das Thema Personalisierung eine wichtige Rolle. Dem Kunden soll dabei eine - an seine Bedürfnisse angepasste - Webseite angeboten werden können. Zusätzliche Vorteile ergeben sich außerdem durch die mögliche Auswertung von „Surf-Gewohnheiten“ der Nutzer.Cocoon stellt hierfür die nötigen Werkzeuge bzw. Funktionalitäten bereit, um eine solche Realisierung zu bewerkstelligen.

Modularität und einfache Erweiterung
Eine wichtige Eigenschaft von modernen Web-Applikationen ist die Verwendung von unabhängigen Bausteinen, die sich unter bestimmten Bedingungen beliebig und vor allem einfach kombinieren lassen.

Das Cocoon Framework stellt eine Vielzahl von Komponenten zur Verfügung, die durch Deklaration eingebunden und ausgetauscht werden können. Eine Erweiterung dieser Komponenten ist durch eine übersichtliche Struktur der Interfaces und Basisklassen gegeben.

Internationalisierung
Im Zuge der Globalisierung ist es auf vielen Webseiten unausweichlich geworden, den Inhalt in verschieden Landessprachen oder Währungen darzustellen.

Die so genannte I18nTransformer-Komponente bietet in Cocoon die Möglichkeit, sowohl Text als auch Datums, Zahlen- und Währungsformate in die jeweilige Zielsprache zu übersetzen.

Skalierbarkeit
In Verbindung mit Web Applikationen hat der Begriff „Skalierbarkeit“ gleich zwei Bedeutungen. Zum einen ist damit gemeint, dass entwickelte Applikationen eine größere Anfrageflut standhalten müssen und somit die Leistungsfähigkeit entsprechend erweiterbar sein sollte, zum anderen die Fähigkeit des „Mitwachsen“ einer Applikation.

Durch einen umfangreichen Caching-Mechanismus hält das Apache Cocoon Framework dieser Forderung stand.

Verschiedene Zielplattformen (Cross-Media-Publishing)
Einer der größten Stärken von Cocoon ist die Fähigkeit, Inhalte in verschiedensten Formaten ausliefern zu können. Durch die Verwendung von XML als Beschreibungssprache sowie der in Cocoon vorhandenen Transformatoren ist die Darstellung der Daten in den gängigsten Formaten möglich (HTML/XHTML, PDF, PostScript, XML, XLS, SVG, Zip).

Verschiedene Datenquellen
Aufgrund der Tatsache, dass Cocoon in Java programmiert wurde, bietet sich für den Zugriff auf Datenbanken JDBC an. Durch JDBC lassen sich nahezu alle existierenden Datenbanken einbinden. Ein Austausch kann dabei ohne jegliche Änderungen am Code der Web-Applikation erfolgen.

Einfache und standardisierte Wartung
Die oben angesprochene Trennung von Daten, Logik und Layout ist Vorraussetzung für die einfache und standardisierte Wartung der Web-Applikation unter Cocoon.

33.2.3. Arbeitsweise von Cocoon

Request-Response-Zyklus
Cocoon arbeitet – ähnlich wie das HTTP-Protokoll – nach dem Request-Response-Zyklus. Dabei wird an Cocoon zunächst ein Request (Anfrage) gesendet, der anschließend durch einen entsprechenden Response (Antwort) beantwortet wird. Eine Antwort könnte z.B. eine angefragte HTML-Seite oder eine Fehlerseite sein, falls das Dokument nicht gefunden werden konnte.

Pipelines
Innerhalb von Cocoon existiert eine so genannte Pipeline-Technik. Dabei handelt es sich um verschiedene Komponenten, die wie einzelne Bausteine hintereinander geschaltet

werden können. Sie besitzen die Aufgabe, die Struktur und den Inhalt eines eingehenden XML-Dokumentes so zu verändern, das das Zielformat (z.B. HTML) erstellt bzw. das Zielsystem (Browser) die erzeugten XML-Daten auch auszuwerten kann. Die einzelnen Komponenten innerhalb einer solchen Pipeline kommunizieren die Daten immer in Form von XML untereinander.

33.2.4. Die Sitemap.xmap mit ihren Komponenten

Die Sitemap ist die „Kommandozentrale“ von Cocoon. In ihr wird bestimmt, welches XML-Dokument in welcher Reihenfolge und anhand welcher Kriterien eingelesen, transformiert und wie wieder ausgegeben werden soll. Die eigentlichen Aktionen führen die definierten Komponenten aus. Diese Komponenten sind unter anderem Generatoren, Transformatoren sowie Serializers und werden unter <map:components> registriert. Bei der Sitemap handelt es sich um eine XML-Datei, die sich standardmäßig im Wurzelverzeichnis von Cocoon befindet. Folgender Quelltext zeigt den typischen Aufbau einer Sitemap-Datei:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
3  <map:components>
4    <!-- Hier werden alle Komponenten registriert -->
5  </map:components>
6
7  <!-- Hier werden die Gruppierungen platziert -->
8
9  <map:pipelines>
10   <map:pipeline>
11     <!-- Hier werden die Match-Bedingungen definiert -->
12   </map:pipeline>
13 </map:pipelines>
14
15 </map:sitemap>
```

Quelltext 33.1: Portal: Sitemap.xmap

Generatoren
Generatoren wandeln beliebige Daten in einen SAX-Stream um. Dieser wird anschließend durchlaufen und auftretende SAX-Events werden verarbeitet. Zu diesen Daten können z.B. XML-Dateien von der Festplatte, über HTTP oder aus einer Datenbank gehören. Dabei kann es sich auch um beliebige andere Dateien anderer Formate handeln, die vorher mittels eines anderen Generators und SAX-Parser in eine XML-Struktur umgewandelt wurden.

Transformer
Transformatoren nehmen in der Regel SAX-Events von Generatoren oder anderen Transformatoren entgegen und transformieren diese anhand ihrer Aufgaben in andere SAX-Events bzw. leiten diese unverändert an eine nachfolgende Komponente weiter. Diese kann wiederum erneut ein Transformer oder ein Serializer sein. Transformatoren sind optional und müssen nicht zwingend in einer Pipeline enthalten sein.

Serializer

Nach der Erzeugung eines SAX-Streams und dessen eventuellen Transformierung erfolgt am Ende einer Pipeline die Ausgabe durch einen Serializer. Dabei werden die auftretenden SAX-Events in das jeweilige Zeilenformat gebracht die bereits anfangs erwähnt wurden.

Alle drei Komponenten bilden zusammen die Pipeline (siehe Abbildung 33.3).

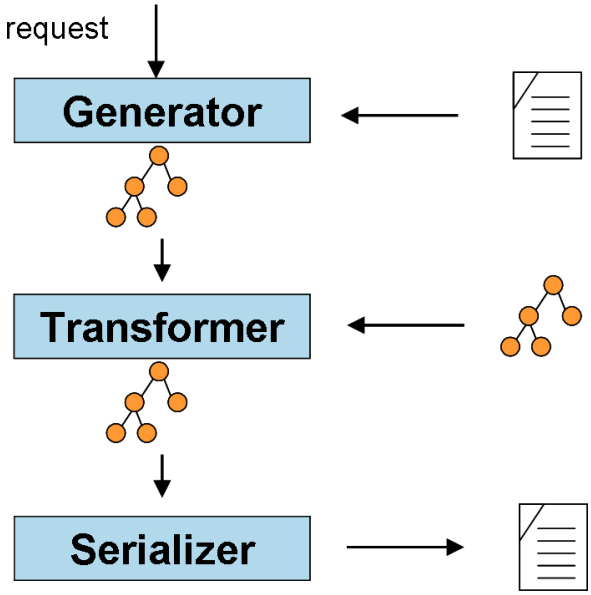


Abbildung 33.3.: Portal: Pipeline

Ein Beispiel für dieses so genannte „Pipeline Processing“ könnte z.B. die Transformierung eines gespeicherten XML-Dokumentes mittels XSL in ein HTML-Dokument sein. Abbildung 33.4 veranschaulicht diesen Ablauf.

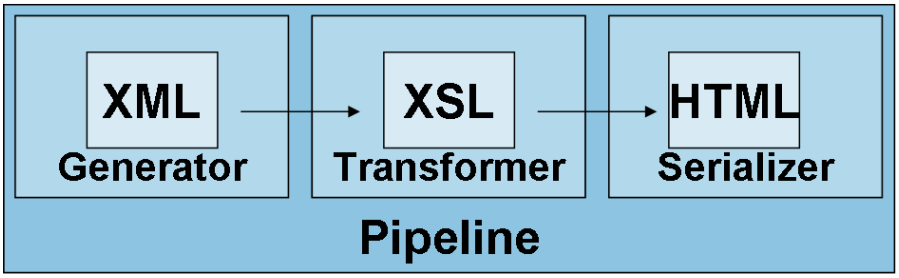


Abbildung 33.4.: Portal: Pipeline Processing

33.2.5. Installieren von Cocoon

Voraussetzung für die Verwendung des Apache Cocoon Frameworks sind folgende vorher zu erledigende Maßnahmen:

- Es muss eine aktuelle Java-Version auf dem Betriebssystem installiert sein. Diese wird auf den Seiten von Sun kostenlos zum Download zur Verfügung gestellt: http://java.sun.com/getjava/de
- Es muss ein Servlet-Container installiert sein. Hierfür verwendeten wir im Projekt BizWeb den Tomcat-Server von der Apache Jakarta Group: http://jakarta.apache.org/tomcat

Die aktuelle Cocoon Version kann von der Webseite http://xml.apache.org/cocoon heruntergeladen werden. Die in diesem Package vorhandene *cocoon.war* muß in das webapps-Verzeichnis von Tomcat kopiert werden. Nach der Installation muß Tomcat neu gestartet werden, um das Apache Cocoon Framework verwenden zu können. Mit dem Aufruf von „http://[servername]/cocoon“ gelangt man auf die Dokumentation- und Beispielseite von Cocoon.

33.3. Anpassung der Cocoon Umgebung für das zu erstellende Portal

Das von Apache zur Verfügung gestellte Cocoon Framework enthält eine Vielzahl von Beispielen aus allen möglichen mit Cocoon zu realisierenden Bereichen. Enthalten sind auch zwei Portale, wobei das Portal Block Beispiel für die Umsetzung des Cocoon Portals gewählt wurde. Um die Lauffähigkeit der Basis Version des Cocoon Portals zu gewährleisten sind die beiden Ordner „WEB-INF“ und „slide“ notwendig.

Der Ordner „WEB-INF“ enthält alle notwendigen Bibliotheken des Cocoon Frameworks, sowie Log-Dateien die alle zur Laufzeit durchgeführten Vorgänge dokumentieren und die Datei „web.xml“, welche die allgemeine Einbindung des Cocoon Portals und die des Kreuzfahrtbuchung Servlets an den Apache Tomcat enthält. Es wird dabei dem Tomcat mitgeteilt wo er diese beiden Dienste findet. Diese Umgebung war ausreichend, um darauf das zu realisierende Portal aufzubauen.

33.4. Erklärung der Cocoon BizWeb Ordnerstruktur

Alle Ordnerstrukturen bis auf die Ordner der Cocoon Basis Version sind frei wählbar und definierbar. Die grobe Struktur des erstellten BizWeb Portals orientiert sich an der Ordnerstruktur des von Apache bereitgestellten Portal Block Beispiels des Cocoon Frameworks. Im Folgenden werden alle enthaltenen Ordner mit ihren Funktionen erläutert. Siehe hierzu Abbildung 33.5.

Die Ordner „coplets“, „profiles“, „recources“, „skins“ und „styles“ sind für die Funktionalität des BizWeb Portals notwendig. Die Ordner „WEB-INF“ und „slide“ sind, wie bereits erwähnt, für die Funktionalität des Portals verantwortlich. Die verbleibenden Ordner werden für die Einbindung der Cocoon Forms Technologie in das Portal benötigt. Diese Ordner und ihre Funktionen werden hier nicht genau erläutert. Auf die Einbindung der Cocoon Forms Technologie wird unter dem Punkt „Einbindung des Kreuzfahrtbuchung Web Service mit Cocoon Forms“ eingegangen.

Der Ordner „coplets“ stellt die Basis des Dateninhalts des Portals dar. Er enthält neben sämtlichen XML-Dateien, welche sich im Unterordner „docs“ befinden, noch die dazugehörigen XSL-Stylesheetdateien im Ordner „docs/styles“. Diese Dateien sind für den Inhalt der Coplets und die Darstellung dieser zuständig. Die im Ordner „docs“ enthaltene Sitemap-Datei weist jeder XML-Datei eine entsprechende Stylesheet-Datei zu.

Der Ordner „profiles“ ist für die Anordnung und Darstellung der Coplets im Portal zuständig. Enthalten sind die Ordner „copletbasedata“, „copletdata“, „copletinstancedata“ und „layout“. Bis auf den Ordner „layout“, welcher für die räumliche Anordnung der Coplets im Portal zuständig ist, enthalten die Ordner Informationen über die Benennung der einzelnen Coplets und deren Inhalt, in Form der Pfade der darzustellenden XML-Dateien.

Der Ordner „recources“ enthält Informationen für das Login am Portal. Es werden z.B. die Benutzernamen mit den dazugehörigen Passwörtern gespeichert und es sind Fehlerseiten für fehlgeschlagene Logins vorhanden.

Der Ordner „skins/common“ beinhaltet neben Stylesheet-Dateien im Ordner „styles“, welche für das allgemeine Aussehen des ständig gleich bleibenden Hintergrundes des Portals verantwortlich sind, außerdem alle im Portal eingefügten Bilder und Images im Ordner „images“ und eine CSS-Datei im Ordner „css“, welche den allgemein verwendeten Font und dessen Eigenschaften definiert.

Die genannten Strukturen sind, solange sie nicht verändert werden, beliebig erweiterbar. Neue Ordner oder sogar Ordnerstrukturen müssen in die entsprechenden Sitemap-Dateien eingefügt und somit für das Cocoon-Portal erkennbar gemacht werden. Die Einbindung neuer Strukturen ist nicht trivial erkennbar und benötigt einige Einarbeitungszeit.

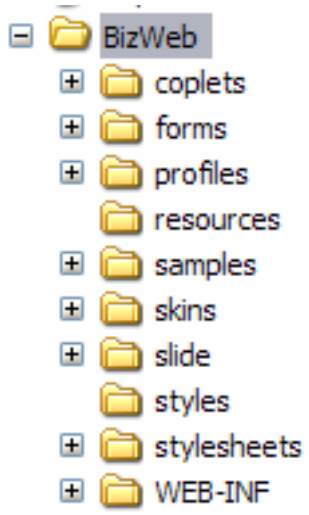


Abbildung 33.5.: Portal: Portal-Ordnerstruktur

33.5. Vorstellung des Cocoon BizWeb Portals

Das BizWeb-Portal wurde wie bereits erwähnt mit dem Apache Cocoon Framework erstellt und dient in erster Linie zur Präsentation der Ergebnisse des BizWeb-Projektes. Die wichtigste Funktion ist der Zugriff auf die erstellten Web Services zur Kreuzfahrtbuchung. Weiterhin wird über das Portal der Zugriff auf die komplette Dokumentation des Projektes, die einzelnen Schulungen, alle erstellten Beispiel-Web Services, eine umfangreiche Linksammlung und weitere das Projekt betreffende Informationen ermöglicht.

Um auf das Portal zugreifen zu können, muss sich der Benutzer auf der Startseite mit den jeweiligen Benutzerdaten einloggen. Im Portalbereich hat der Benutzer dann Zugriff auf die eigentlichen Funktionen. Die Navigation erfolgt dabei über ein Reitersystem. Es gibt sechs verschiedene Reiter, bei denen die Inhalte mit Hilfe von „Coplets“ (= Cocoon Portlets) generiert wurden. Informationen über die einzelnen Reiter stehen in den Punkten 33.5.2 bis 33.5.7.

Das Design des Portals ist durchgehend einheitlich. Da sich das BizWeb-Projekt auf eine Kreuzfahrtbuchung bezieht, wurde hauptsächlich mit blauen und weißen Farben gearbeitet. Am oberen Bildrand befindet sich ein blauer Balken, in den das BizWeb-Logo eingearbeitet worden ist. Darunter befindet sich der veränderbare Bereich mit den Inhalten, ganz unten befindet sich dann wieder ein blauer Balken. Die jeweiligen Coplets sind ebenfalls in blau-weiß gehalten.

Das BizWeb-Portal ist im Internet über die Adresse http://195.37.183.100/BizWeb/portal erreichbar und hat den folgenden Aufbau:

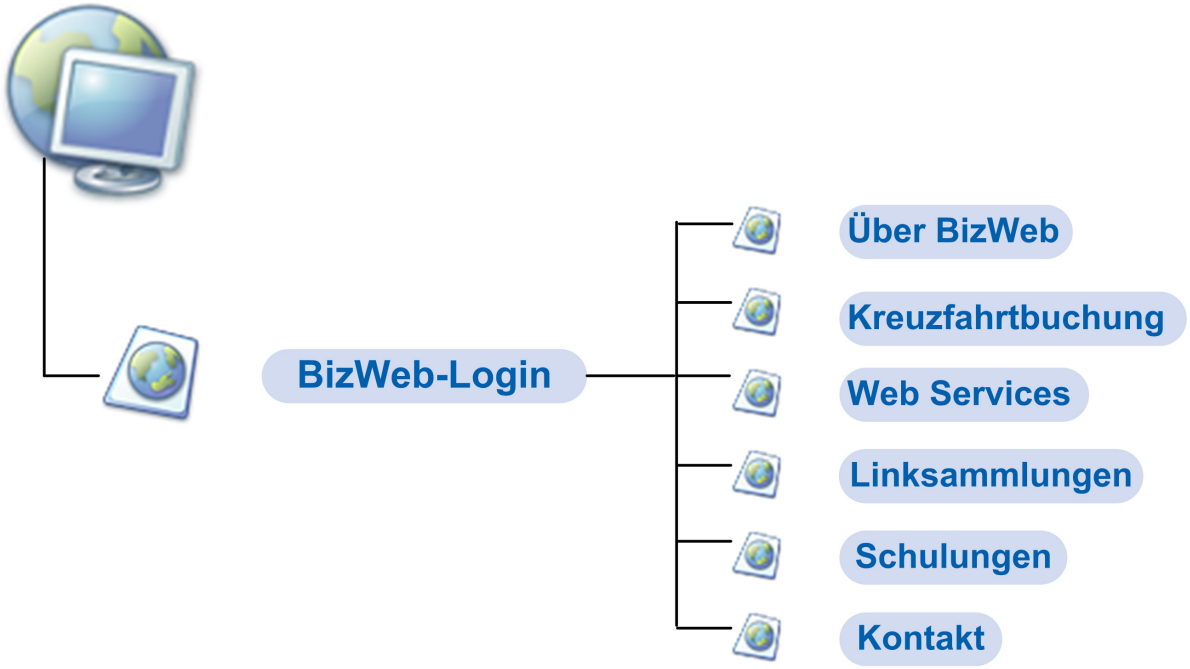


Abbildung 33.6.: Portal: Aufbau des BizWeb-Portals

33.5.1. Login-Bereich

Bereits auf der Startseite finden sich einige allgemeine Informationen über das Projekt. Diese wurden über ein Coplet in die linke Seite des Fensters eingebunden. Auf der rechten Seite befindet sich der Login-Bereich. Hier können sich die Benutzer mit ihren jeweiligen Daten einloggen um zu den eigentlichen Inhalten zu gelangen.

Abbildung 33.7.: Portal: Startseite

Bei einer fehlerhaften Eingabe der Benutzerdaten erscheint ein Bildschirm mit der Information, dass die Eingabe nicht korrekt war. Über einen Link gelangt der Benutzer wieder zur Startseite.

Abbildung 33.8.: Portal: Fehlgeschlagener Login

33.5.2. Über BizWeb

Auf dieser Seite finden sich ausführliche Informationen über das Projekt. Es wird kurz erklärt, was die Ziele des Projektes waren, welche Programme und Techniken genutzt wurden und wie die zeitliche Aufteilung des Projektes war. Weiterhin ist ein Link auf die umfangreiche Dokumentation des Projektes vorhanden. Im unteren Bereich der Seite wurde schließlich noch ein Bild von den Projektmitarbeitern eingefügt.

Abbildung 33.9.: Portal: Über BizWeb (1)

Übungs-Web Services:

- Taschenrechner
- Währungsrechner
- Flugbestätigung
- Wertpapierkurs
- Kreuzfahrtliste
- Hotelbuchung
- Textanalysierer
- Literatursuche
- Produktkatalog
- Stückliste
- Weltuhrzeit
- Kreuzfahrtbuchung (Test)

Weitere Informationen zu diesen Web Services finden Sie im Bereich **Web Services** in diesem Portal.

2. Realisierungsphase

Teilprojekte:

- Datenbankmodell SAP
- SAP-Java-Connector
- Web Service "Kreuzfahrtbuchung" und Servlet
- Intermediaries
- Cocoon-Portal

Projektmitarbeiter:
Von links:Benny Bräuer, Bastian Onken, Remo Fulle, Christopher Schrade,
Jan Stelmaszek, Alfred Schmidt, Marco Junge, Dominik Buhlmann,
Nicole Siedek, Thomas Wäsch, Christian Eggers, Christian Lefke,
Viktor Schlegel (nicht auf dem Bild: Prof. Dr. Dieter Viefhues-Veensma)

Abbildung 33.10.: Portal: Über BizWeb (2)

33.5.3. Kreuzfahrtbuchung

Auf dieser Seite sollten ursprünglich die erstellten Kreuzfahrt-Web Services als XSP-Dateien über Cocoon-Forms in die Coplets eingebunden werden (Siehe Kapitel 33.7). Da das nicht ohne weiteres möglich war, wurden hier einige Links eingefügt die zum einen auf die extra erstellten Servlets, zum anderen auf die Lösung mit XSP und Cocoon-Forms verweisen.

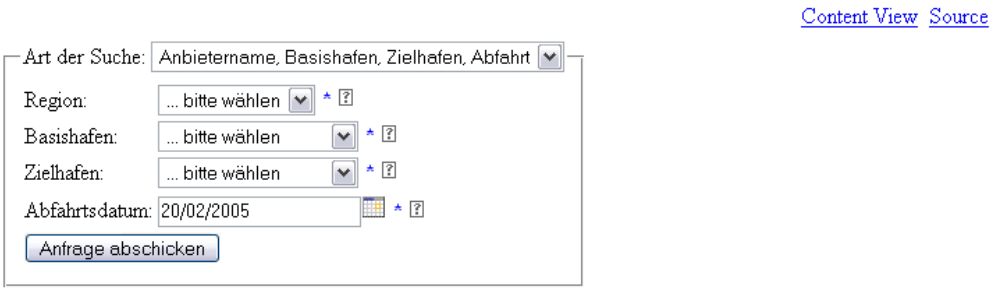


Abbildung 33.11.: Portal: Kreuzfahrtsuche als XSP-Datei

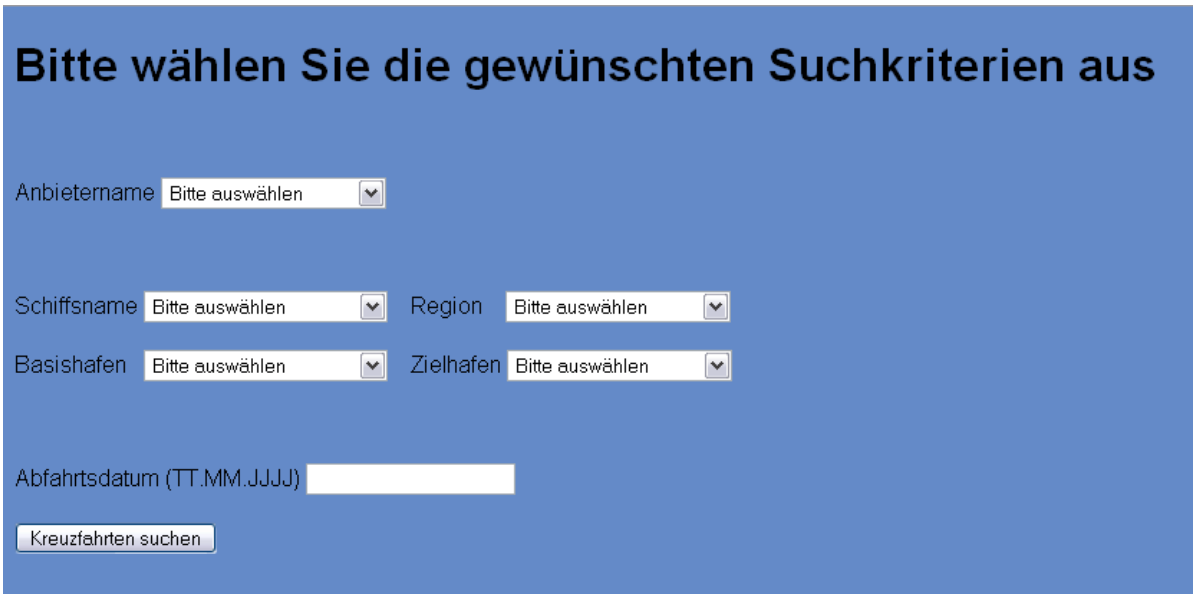


Abbildung 33.12.: Portal: Kreuzfahrtsuche als Servlet

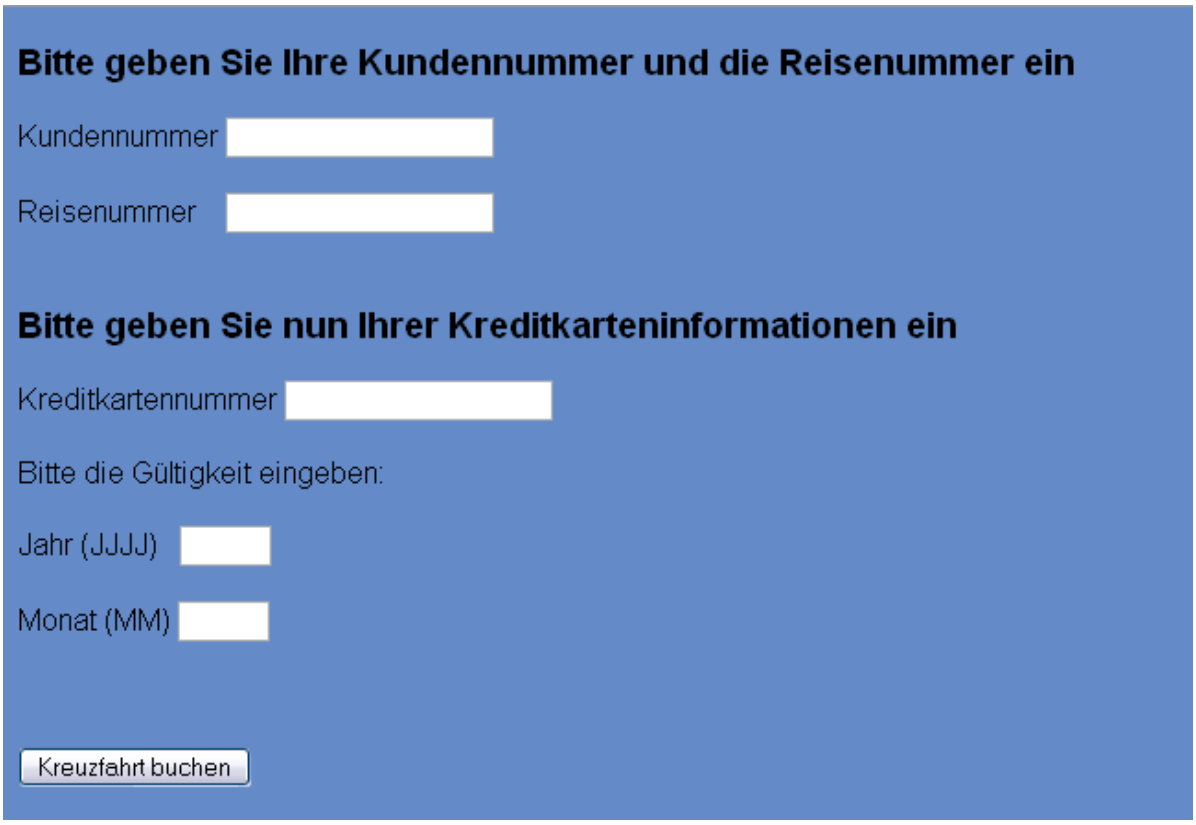


Abbildung 33.13.: Portal: Kreuzfahrtbuchung als Servlet

33.5.4. Web Services

Jeder Projektmitarbeiter musste im Laufe des Projektes einen eigenen Web Service erstellen. In diesem Bereich gibt es zu jedem erstellten Web Service eine kurze Erklärung, sowie Links zur JWS-Datei, zur WSDL-Datei, zur Dokumentation und zur UDDI-Veröffentlichung.

Web Service	Beschreibung	JWS-Datei	WSDL-Datei	Dokumentation	UDDI
Flugbestätigung (Buchungsbest)	Simulation einer Flugbuchung	JWS	WSDL		UDDI
Hotelbuchung (Hotelbuchung)	Simulation einer Hotelbuchung	JWS	WSDL		UDDI
Kreuzfahrtbuchung (Kreuzfahrtbuchung_test)	Simulation einer Kreuzfahrtbuchung	JWS	WSDL		UDDI
Kreuzfahrtliste (RouteList)	Ausgabe von Schiffsrouten aus einer MySQL-Datenbank	JWS	WSDL		
Literatursuche (literatursuche)	Beispielhafte Literatursuche in einer XML-Datenbank	JWS	WSDL		UDDI
Produktkatalog (Produktsuche)	Beispielhafte Produktsuche in einer XML-Datenbank	JWS	WSDL		UDDI
Stückliste (steucklistedb)	Stücklisten von Produkten aus einer XML-Datenbank abrufen	JWS	WSDL		UDDI
Taschenrechner (Taschenrech)	Taschenrechner für einfache Grundrechenarten	JWS	WSDL		
Textanalysierer (TextAnalyser)	Web Service zum Untersuchen von Textstrukturen	JWS	WSDL		UDDI
Währungsrechner (Currency)	Web Service zum Umrechnen verschiedener Währungen	JWS	WSDL		UDDI
Weltuhrzeit (weltuhrzeit)	Ermittlung von Uhrzeiten in verschiedenen Regionen	JWS	WSDL		UDDI
Wertpapierkurs (StockWS)	Abrufen von Wertpapierkursen aus einer MySQL-Datenbank	JWS	WSDL		UDDI

Abbildung 33.14.: Portal: Übungs-Web Services

33.5.5. Linksammlung

Auf dieser Seite befinden sich eine Liste der genutzten Literatur, sowie eine große Linksammlung zu allen das Projekt betreffenden Bereichen. Die Links wurden dabei in folgende Bereiche unterteilt: XML, Web Service, Software, Spezifikationen, Java Connector und SAP Developer.

Abbildung 33.15.: Portal: Linksammlung

33.5.6. Schulungen

Im Rahmen des Projektes wurden Schulungen zu den Grundlagen der Web Service-erstellung durchgeführt. Auf dieser Seite befinden sich die Präsentationen mit den dazu gehörigen Ausarbeitungen, sowie einige Beispiele. Der Benutzer kann sich hier einen kurzen Überblick über die genutzten Programme und Techniken verschaffen.

Abbildung 33.16.: Portal: Schulungen

33.5.7. Kontakt

Hier befinden sich schließlich die Adresse der Hochschule Bremerhaven, sowie die Kontaktdaten der Projektleiter, mit E-Mail-Adresse und Telefonnummer.

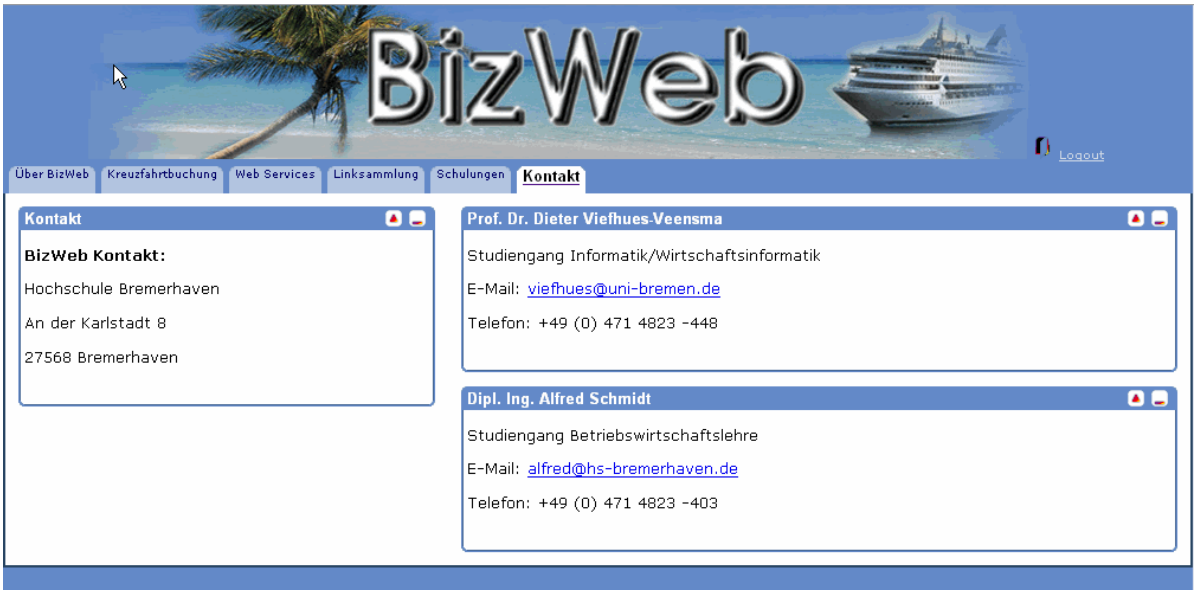


Abbildung 33.17.: Portal: Kontakt

33.6. Erläuterung der Funktionen der verwendeten Dateitypen

Bei der Bearbeitung und Anpassung des BizWeb-Portals mussten viele verschiedene Dateien bearbeitet, bzw. neu erstellt werden. In diesen Dateien wurden unter anderem das Design des Portals, die Aufteilung und die Inhalte der Coplets und weitere Anpassungen festgelegt.

Insgesamt wurde mit 4 verschiedenen Arten von Dateien gearbeitet: Sitemap.xmap-Dateien, XML-Dateien, XSL-Dateien und CSS-Dateien.

33.6.1. Sitemap.xmap

Die Funktionen der Sitemap.xmap-Dateien sind dem Punkt 33.2.4 dieser Dokumentation zu entnehmen. Es ist festzuhalten, dass es zwei unterschiedliche Arten von Sitemap-Dateien in dem BizWeb Portal gibt.

Die eine ist die Haupt-Sitemap-Datei, welche sich im Wurzelverzeichnis des Portals befindet und sämtliche für das Portal benötigte „Components“ enthält sowie Pipelines,

welche der allgemeinen Definition der Ordnerstrukturen des Portals dienen, wie z.B. der Verweis wo sämtliche Bilder des Portals gespeichert sind.

Die zweite Art von Sitemap-Datei dient lediglich der Weiterleitung oder der Definition von „Match Patterns“. Eine Weiterleitung per Sitemap-Datei wird benötigt, sobald eine Verzeichnisstruktur verwendet wird, welche nicht in der Haupt-Sitemap-Datei definiert ist. Erfolgt keine Weiterleitung kann diese Struktur vom Portal nicht gefunden werden. „Match Patterns“ werden verwendet um dem Portal mitzuteilen welche Stylesheet-Datei mit welcher XML-Datei zu verwenden ist. Hierbei wird mitgeteilt welcher „Generator“, „Transformer“ und „Serializer“ für diese Umwandlung zu verwenden ist.

33.6.2. XML

Die wohl wichigsten XML-Dateien sind die „portal.xml-Dateien“, die in verschiedenen Ordnern unter „/BizWeb/profiles“ liegen. In ihnen wird der Aufbau des Portalbereiches festgelegt, wie die Anzahl der verwendeten Reiter, die Anzahl der Coplets pro Seite und die Aufteilung der Coplets pro Seite. Die „login.xml“-Datei im Ordner „/BizWeb/coplets/login“ legt die Bezeichnungen, Feldtypen und Längen der Login-Felder fest, die XML-Dateien im „/BizWeb/resources“-Ordner befassen sich mit den weiteren Login-Funktionen, wie Benutzerverwaltung, Fehlgeschlagenen Logins, usw. Die XML-Dateien im Ordner „/BizWeb/coplets/docs“ und an verschiedenen weiteren Stellen dienen zum Festlegen sämtlicher Inhalte der einzelnen Seiten und Coplets, die im Portal genutzt werden.

Wie zu erkennen ist, werden also alle Portalinhalte in XML-Dateien gespeichert. Das untere Beispiel zeigt einen Ausschnitt der *schulungen.xml*, in der sämtliche Daten vorhanden sind, die im HTML-Dokument dargestellt werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="C:\Programme\java\Tomcat 5.0\webapps\
BizWeb-ohne-rss\coplets\docs\styles\table.xsl"?>
<document>
  <body>
    <links xmlns="http://195.37.183.100" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance">
      <schulungen>
        <thema>XML Grundlagen Teil 1   </thema>
        <beschreibung>Einführung in XML, Validierung (DTD, Schema), CSS, XSL
          </beschreibung>
        <präsentation>http://195.37.183.100/docs/praesentation/XML_Grundlagen_Teil1.ppt
          </präsentation>
        <präs_icon>images/PPT_icon.gif</präs_icon>
        <ausarbeitung>http://195.37.183.100/docs/pdfs/schul02xml_grundlagen.pdf
          </ausarbeitung>
        <ausarb_icon>images/PDF_icon.gif</ausarb_icon>
        <beispiele>http://195.37.183.100/docs/beispiele/DTD_Schema_XSL.zip</beispiele>
        <beisp_icon>images/ZIP_icon.jpg</beisp_icon>
      </schulungen>
      <schulungen>
        <thema>XML Grundlagen Teil 2</thema>
        <beschreibung>XML Parser (SAX, DOM, JDOM)</beschreibung>
        <präsentation>http://195.37.183.100/docs/praesentation/XML_Grundlagen_Teil2.ppt
          </präsentation>
```

```
26          <präs_icon>images/PPT_icon.gif</präs_icon>
27          <ausarbeitung>http://195.37.183.100/docs/pdfs/schul03xml_parser.pdf
28            </ausarbeitung>
29          <ausarb_icon>images/PDF_icon.gif</ausarb_icon>
30          <beispiele>http://195.37.183.100/docs/beispiele/XMLParser.zip</beispiele>
31          <beisp_icon>images/ZIP_icon.jpg</beisp_icon>
32        </schulungen>
33        ...
34      </links>
35    </body>
36  </document>
```

Quelltext 33.2: Portal: XML

33.6.3. XSL

In den XSL-Dateien wird der Grundsätzliche Stil des Portal-, Seiten- und Copletaufbaus festgelegt. Die XSL-Dateien sind weiterhin für die Interpretation der XML-Datei-Inhalte verantwortlich und legen fest, wie die Inhalte dargestellt werden, welche Feldtypen verwendet werden dürfen, wie die Schachtelung der XML-Dateien sein muss usw. Weiterhin dienen sie zum Darstellen verschiedener Inhalte im HTML-Format. Das untere Beispiel zeigt die *table.xsl*, die die *schulungen.xml* in Tabellenform als HTML-Dokument ausgibt.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4                  xmlns:bns="http://195.37.183.100">
5  <xsl:template match="document/body">
6     <html>
7        <head><title>Schulungsunterlagen</title></head>
8        <body>
9           <table border="1">
10             <!--<tr bgcolor="#648ac8">-->
11             <tr>
12                <th align="left">Thema</th>
13                <th align="left">Beschreibung</th>
14                <th align="left">Präsentation</th>
15                <th align="left">Ausarbeitung</th>
16                <th align="left">Beispiele</th>
17             </tr>
18             <xsl:for-each select="bns:links/bns:schulungen">
19                <tr>
20                   <td><xsl:value-of select="bns:thema"/></td>
21                   <td><xsl:value-of select="bns:beschreibung"/></td>
22                   <td align="center">
23          <xsl:if test="bns:präsentation">
24           <a href="{bns:präsentation}">
25            <img src="{bns:präs_icon}" width="25" height="25" border="0" />
26           </a>
27          </xsl:if>
28    </td>
29
30                   <td align="center">
31          <xsl:if test="bns:ausarbeitung">
32                <a href="{bns:ausarbeitung}">
33             <img src="{bns:ausarb_icon}" width="25" height="25" border="0" />
34            </a>
35           </xsl:if>
36                   </td>
```

```
37                    <td align="center">
38          <xsl:if test="bns:beispiele">
39           <a href="{bns:beispiele}">
40            <img src="{bns:beisp_icon}" width="25" height="25" border="0" />
41           </a>
42          </xsl:if>
43
44                  </td>
45                </tr>
46              </xsl:for-each>
47          </table>
48        </body>
49    </html>
50 </xsl:template>
51 </xsl:stylesheet>
```

Quelltext 33.3: Portal: XSL

33.6.4. CSS

Die CSS-Dateien im Portal dienen zur Textgestaltung. In ihnen werden Werte für die Textgröße, Textfarbe, Umrandungen, Hintergrundfarben, Schriftart usw. festgelegt. Für die Textinhalte in den Protal-Coplets ist dabei einzig die Datei „page.css“ im Ordner “/BizWeb/skins/common/css“ zuständig. Für die Cocoon-Forms gibt es weitere CSS-Dateien an verschiedenen Stellen im Forms-Ordner.

33.7. Einbindung des Kreuzfahrtbuchung Web Service mit Cocoon Forms

Cocoon Forms ist eine Technologie welche es ermöglichen soll Formularseiten in ein Cocoon Portal zu integrieren. Da es nicht möglich ist Servlets direkt in ein solches Portal einzubinden musste die Umsetzung mit Cocoon Forms geschehen. Leider blieb es bei dem Versuch einer Umsetzung.

Cocoon Forms nutzt die Technologien XML, XSL, JavaScript und XSP. Notwendig für eine Cocoon Forms Anwendung sind ein Forms Modell (form1.xml) und ein Forms Template (form1_template.xml). Für die Umsetzung der Anbindung an den Web Service soll es möglich sein, die an den Web Service zu übergebenden Parameter in Form eines Drop-Down-Menüs auszuwählen und über einen Submit-Button an den Web Service zu übertragen. Das Ergebnis soll nun an das Portal zurückgegeben werden und dargestellt werden.

Es ist anzumerken, dass Cocoon Forms sich noch in der Entwicklungsphase befindet und von Apache noch keine stabile Version veröffentlicht worden ist.

Abbildung 33.18 zeigt wie ein Cocoon Forms Beispiel in der Regel funktioniert und abgearbeitet wird.

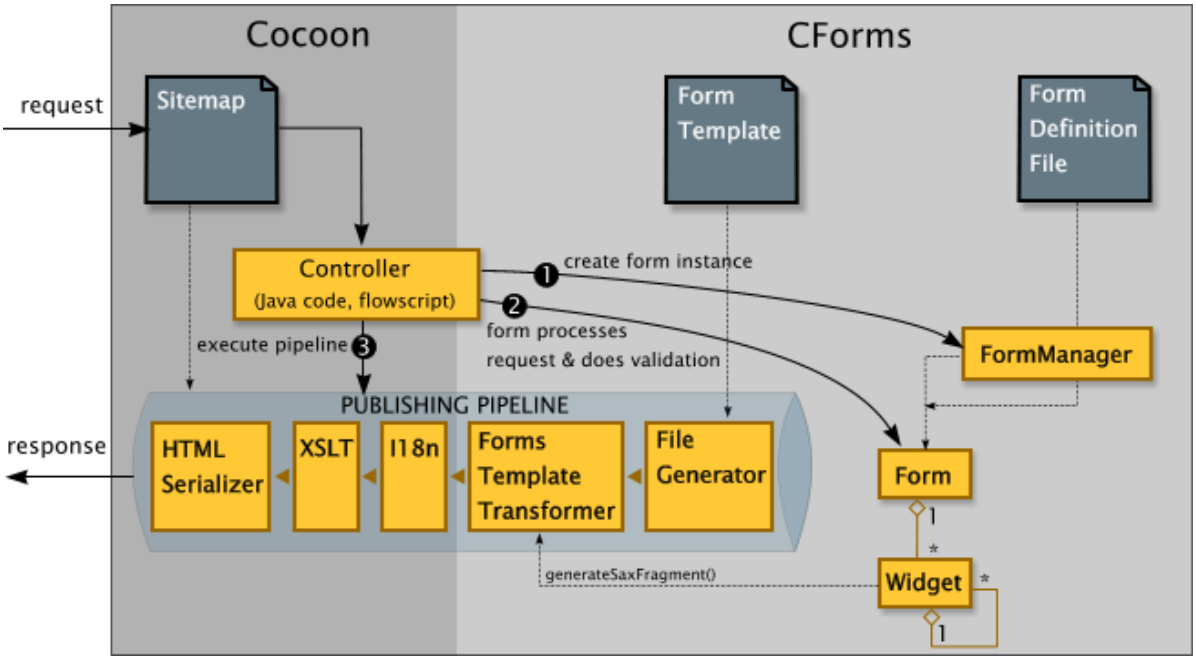


Abbildung 33.18.: Portal: Beschreibung eines typischen Cocoon Forms Beispiels

Das Drop-Down-Menü wird in der Datei form1_template.xml erstellt. Jeder Menüpunkt bekommt hier eine Widget-ID zugewiesen. Widgets sind vereinfacht ausgedrückt Objekte welche ihren eigenen Wert an einen Request übergeben, sich selbst validieren und eine XML Repräsentation von sich selbst erstellen können. Sie sind in der Datei „forms1.xml“ im Ordner „forms/forms“ definiert.

Diese Widgets werden in der XSP-Datei „form1_success.xsp“ ausgelesen und in Java-Variablen gespeichert. Cocoon Forms nutzt bei dieser Umsetzung JavaScript im Ordner „forms/flow“ um die Variablen der Widgets mit Default-Werten zu belegen und um die Form-Übergabe als Request zu speichern, da XSP die Übergabe als Flow nicht erkennen würde. Bisher wurde lediglich eine Abfrage und anschließende Ausgabe der übergebenen Werte durchgeführt. Probleme bei der Einbindung von Java in XSP verhinderten an diese Stelle eine Weiterarbeit.

Die Definiton eines Widgets sieht in der Datei „form1.xml“ wie folgt aus:

```
<fd:widgets>
   <fd:field id="Schiff" required="true">
      <fd:label>Schiffsname:</fd:label>
      <fd:datatype base="string"/>
      <fd:help>Bitte wählen Sie den Schiffsnamen!</fd:help>
      <fd:selection-list>
        <fd:item value="... bitte wählen"/>
        <fd:item value="Aidacara"/>
        <fd:item value="Albatros"/>
        <fd:item value="Caribbean Princess"/>
        <fd:item value="MS Bremen"/>
        <fd:item value="MS Europa"/>
        <fd:item value="Queen Mary 2"/>
      </fd:selection-list>
   </fd:field>
</fd:widgets>
```

Quelltext 33.4: Portal: XSL

Der über das Menü ausgewählte Wert des Widgets SSchiff"wird in der XSP-Datei folgendermaßen ausgelesen und ausgegeben:

```
<xsp:page language="java"
  xmlns:xsp="http://apache.org/xsp">

  <xsp:structure>
    <xsp:include>org.apache.cocoon.forms.formmodel.*</xsp:include>
  </xsp:structure>

  <page>
    <title>Kreuzfahrtsuchanfrage</title>
    <content>
      <xsp:logic>
        // get reference to form and some of the widgets on it
        Form form = (Form)request.getAttribute("form1");
        Field field = (Field)form.lookupWidget("Schiff");
</xsp:logic>

Schiff has the following value:
      <xsp:expr>field.getValue()</xsp:expr>
<br/>
    </content>
  </page>

</xsp:page>
```

Quelltext 33.5: Portal: XSL

Der Logic-Teil enthält den Java-Code der XSP-Datei. Nur hier ist Java-Code erlaubt. Der Rest der Datei verhält sich wie eine XML-Datei.

Abbildung 33.19.: Portal: Umsetzung von Cocoon Forms im BizWeb Portal

Abbildung 33.19 zeigt die Einbindung des Cocoon Forms Formulars in das BizWeb Portal. Leider waren die Wechsel zwischen den Tabs, wie in der nächsten Abbildung zu sehen, im Portal nicht umzusetzen. Der Grund ist nicht bekannt, da Cocoon Forms sich noch in einem relativ frühen Stadium befindet und somit wenig bis keine Dokumentation vorhanden ist.

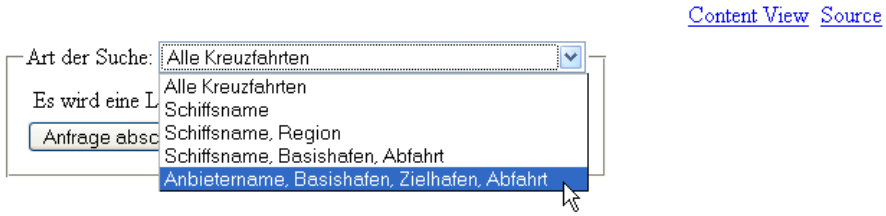


Abbildung 33.20.: Portal: Externe Umsetzung der Cocoon Forms Lösung

Abbildung 33.20 zeigt den Wechsel zwischen den verschiedenen Tabs die jeweils eine andere Auswahl der an den Web Service zu übergebenen Parameter enthalten. Je nach Auswahl soll die entsprechende überladene Methode des Web Service aufgerufen werden.

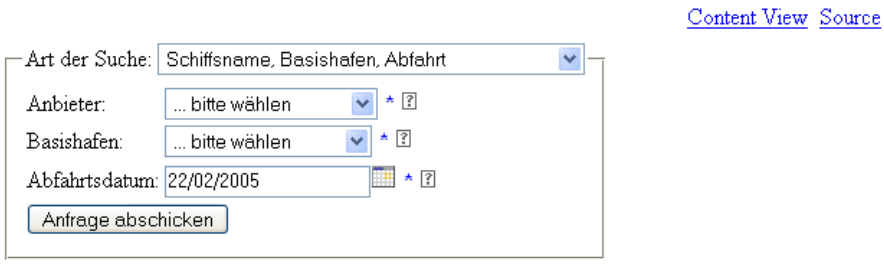


Abbildung 33.21.: Portal: Darstellung des Cocoon Forms Formulars mit drei Parametern

Jedes Feld eines Formulars ist mittels des Sterns an der rechten Seite als Pflichtfeld gekennzeichnet. Das Fragezeichen öffnet bei einem Mausklick aus selbiges ein Hilfsfenster das den Inhalt des Formularfeldes erklärt. Über den Button „Anfrage senden“ werden die eingegebenen Daten übergeben.

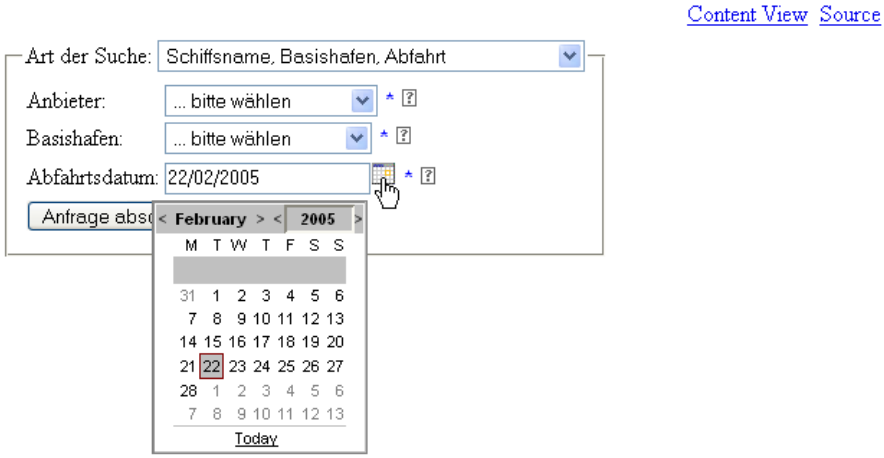


Abbildung 33.22.: Portal: Darstellung des Cocoon Forms Formulars mit Kalender Funktion

Abbildung 33.22 zeigt die in das Formular eingebaute Kalenderfunktion. Sie erhält über die JavaScript-Datei das aktuelle Datum als Vorgabewert.

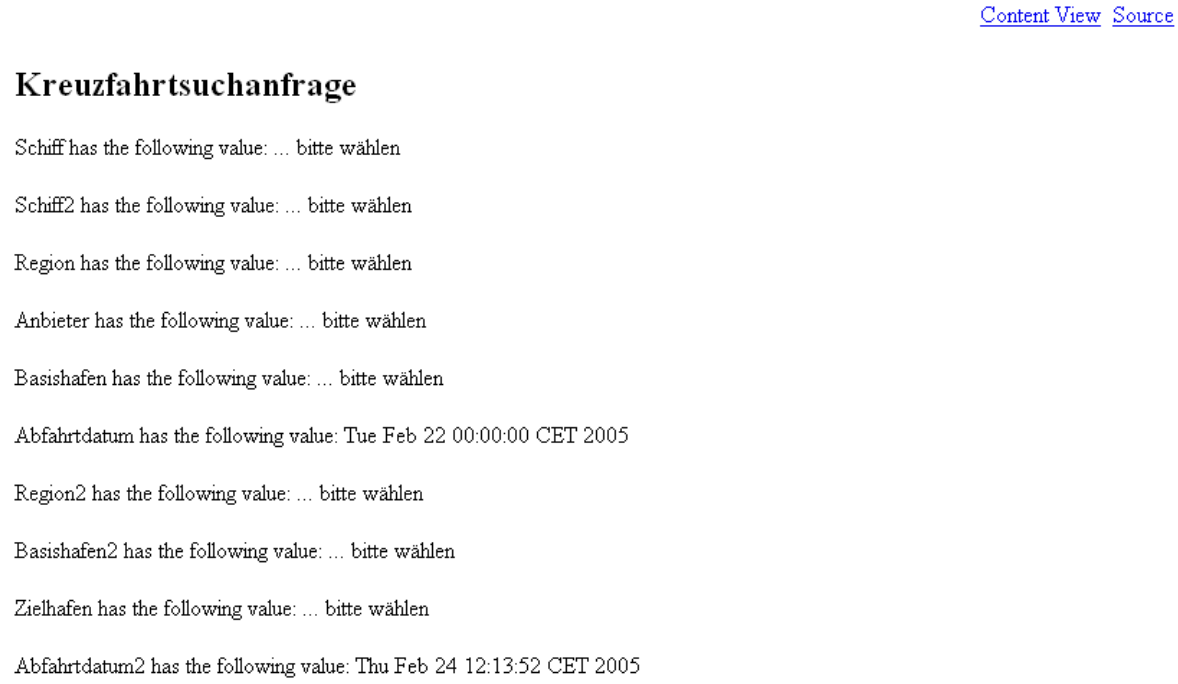


Abbildung 33.23.: Portal: Darstellung des Ergebnisses der Parameterübergabe

Abbildung 33.23 zeigt die vorläufige Überprüfung der Parameterdaten an die XSP-Datei. Das Beispiel zeigt, dass lediglich die Variable Abfahrtdatum einen Wert enthält. Abfahrtdatum 2 enthält das als Default-Wert übergebene aktuelle Datum.

34. Abschlussbesprechnung

Ergebnisorientierung und vorbildliche Teamarbeit – Das sind die Worte, die man benutzen könnte, um die Durchführung der Projektarbeit in den Gruppen untereinander zu beschreiben. Es gibt ausschließlich Positives über den Projektablauf zu sagen.

Dass die Projektteilnehmer dieses ähnlich sehen, wird in den abschließenden Bemerkungen der Teams im Folgenden ebenso deutlich.

34.1. Resümee des Datenbank–Teams

Rückblickend auf das Projekt wird ein strukturierter Ablauf sichtbar. Es war sinnvoll die Schulungen in den ersten Phasenbereich des Projekts zu legen um Grundlagen zu erwerben und in der zweiten Phase die Umsetzung des Projekts. Die Schulung an sich hat sehr viel Zeit in Anspruch genommen, welches kein Kritikpunkt ist. Projekte in der freien Wirtschaft nehmen sicherlich ähnliche Dimensionen an. Der Schulungsteil bei Projekten darf für Projektmitglieder nicht vernachlässigt werden, denn umso geringer die Grundkenntnisse jedes einzelnen Mitglieds sind umso eher steckt das Projekt irgendwann in einem Detailproblem fest und verliert auf diese Weise sehr viel Zeit.

Alle Projektmitglieder haben gut miteinander gearbeitet, wobei der Teamzusammenhalt deutlich spürbar wurde. Dieser Zusammenhalt ist sehr wichtig für das Gelingen des Projekts, da jedes Mitglied seine eigenen Erfahrungen mitbringt und bei den Schulungen die Informationen unterschiedlich aufnimmt. Selten fallen alle Problemlösungen einem einzelnen Teilnehmer ein.
Trotz der Schulungen hat man bei den einzelnen Web Service Übungen mit den Web Services in der Umsetzung Schwierigkeiten gehabt. Es war nicht einfach alles unter einem Hut zu bringen und dass sich hierbei gerade an dieser Stelle der Teamwork bezahlt macht. Die Schulungen haben meist den Kern einer Technik genauestens gezeigt aber oftmals die Schnittstellen weniger verdeutlicht. Hieraus ergaben sich oft Schnittstellen– und Verständnisprobleme zwischen den einzelnen Techniken.

Persönlich war es sehr Interessant an diesem Projekt mitzuarbeiten. Es hat sehr viel Spaß gemacht im Team zu arbeiten, seine eigenen Erfahrungen miteinzubringen und seine Kenntnisse zu erweitern.

— *V. Schlegel, N. Siedek*

34.2. Resümee des JCo–Teams

In unserem zwei Personen umfassenden Team hat die Bearbeitung des Java–Connectors aufgrund jahrelanger Zusammenarbeit einwandfrei geklappt. Es gibt mehrere Gründe, warum wir uns gerade für den JCo entschieden haben. Zum einen wollten wir diese Technologie sehr gerne kennen lernen und zum anderen hat dabei speziell die Kombination mehrerer Techniken den Reiz an diesem Teilgebiet des Projekts ausgemacht. Hier wurden Java–, SAP–, ABAP– und Datenbankkenntnisse wieder aufgefrischt und verinnerlicht.

Die größte Herausforderung lag darin, mit den anderen Teilgruppen seitens SAP und seitens des Web Services zusammenzuarbeiten und eine gemeinsame Schnittstelle zu definieren, was uns schließlich auch gelungen ist. Die Handhabung des Problems der inkompatiblen Datentypen von Java und ABAP stellte eine weitere kleine Herausforderung da.

Die Zusammenarbeit mit den anderen Teams hat immer sehr gut geklappt und die Arbeit hat großen Spaß gemacht.

— D. Buhlmann, C. Eggers

34.3. Resümee des Web Service–Teams

Die gesamte Umsetzung des Web Services mit der Konzeption bis hin zur Realisierung gestaltete sich als äußerst aufwändig. Als hilfreich erwiesen sich die vorher aufgestellte Zeitplanung und die konzeptionelle Vorarbeit. Vor allem einige Versionsprobleme, die beim Deployment des Web Services auf dem Server entstanden, nahmen teilweise sehr viel Zeit in Anspruch. Trotz dieser Schwierigkeiten war diese Aufgabe interessant und forderte uns persönlich heraus.

Die Schulung und die gestellten Übungen in der ersten Phase des Projektes bildeten eine breite und gute Grundlage für die Implementierung des Web Services in der zweiten Phase.

Wir sind der Meinung, dass die Technologie Web Services noch lange nicht ausgeschöpft ist, sondern in Zukunft hinsichtlich der Digitalisierung von Geschäftsprozessen eine immer größere Rolle spielen werden. Aus diesem Grund sehen wir diese Projektarbeit nicht als eine Pflichtveranstaltung im Hauptstudium, sondern als eine gute Bewerbungsgrundlage für den Berufseinstieg.

Die gute Zusammenarbeit sowohl innerhalb der eigenen Gruppe als auch innerhalb des gesamten Projektteams steuerte zum Gesamterfolg des Projekts bei. Die Vorarbeiten aller Teilnehmer waren sehr gut umgesetzt und wurden fristgerecht abgegeben, so dass es zu keiner Verzögerung unserer Arbeit gekommen ist. Die Umsetzung des Projektes hat uns mit der gesamten Gruppe sehr viel Spaß gemacht.

Abschließend sind wir der Meinung, dass dieses erfolgreich abgeschlossene Projekt als eine sehr gute Grundlage für Folgeprojekte dienen kann.

— *M. Junge, C. Lefke, J. Stelmaszek*

34.4. Resümee des Intermediary–Teams

Ein abschließendes Fazit lässt sich in zwei Teile gliedern. Zum Einen wäre das die Arbeit in der einzelnen Gruppe, zum Anderen die Zusammenarbeit des gesamten Projekts. Die Arbeit in der Teilgruppe war sehr gut, schnell wurde das Problem erkannt, welches mit der Kreditkartenprüfung verbunden war. Effizient und effektiv wurde eine Lösung erarbeitet. Diese Lösung ließ sich ohne Probleme in die Arbeit der anderen Gruppen integrieren, was nicht zuletzt auf die exzellente Kommunikation unter den jeweiligen Gruppen als auch den einzelnen Mitgliedern des Projektes zurückzuführen ist.

— *B. Bräuer, B. Onken*

34.5. Resümee des Portal–Teams

Zum Schluss sei festzuhalten, dass wir das Projekt zu unserer vollsten Zufriedenheit erfolgreich abschließen konnten. Es stellte eine große Herausforderung dar, in diesem Projekt mitzuwirken und eine der jeweils sehr komplexen Teilaufgaben übernehmen zu können. Der Zusammenhalt der Gruppe war hervorragend. Alle von uns geplanten Meilensteine konnten wir durch die perfekte Teamarbeit und Teamfähigkeit eines jeden Einzelnen erreichen. Bei der Entwicklung des Portals mittels Cocoon wurde sich an die Hinweise aus dem Buch „Cocoon 2 mit Tomcat“ von Stefan Niedermeyer orientiert. Eine weitere Hilfe war dessen Tutorial, welches mit nützlichen Informationen zum allgemeinen Einsatz von Cocoon oft hilfreich war. Leider war kaum Literatur hinsichtlich der Portal–Engine vorhanden. So mussten wir uns die einzelnen Technologien selber aneignen und trafen dabei immer wieder auf Probleme, da die komplexen Quelltexte teilweise sehr unübersichtlich, nicht ausreichend dokumentiert und nicht trivial waren. Trotz dieser erschwerten Bedingungen ist es uns gelungen, ein den Anforderungen dieses Projektes gerecht werdendes Portal zu erstellen. Dabei haben wir neue und vor allem aktuelle Technologien erfolgreich erlernt und angewendet, die uns bei unserem weiteren beruflichen Werdegang sehr nützlich sein werden.

Zusammenfassend kann man dieses Projekt wohl als den Höhepunkt des Studiums bezeichnen und als einen äußerst spannenden und gelungenen Studienabschluss.

— *R. Fulle, C. Schrade, T. Wäsch*

Teil V.

Anhang

A. Schulungen

A.1. Eclipse

A.1.1. Das 1. Java-Projekt: Ein einfaches Swing-Fenster (Button und Textfeld) mit dem Visual Editor erstellen

```
1  import java.awt.*;
2  import javax.swing.JFrame;
3
4  public class SwingTest extends JFrame {
5    private javax.swing.JPanel jContentPane = null;
6    private javax.swing.JTextPane jTextPane = null;
7    private javax.swing.JButton jButton = null;
8    public SwingTest() // Default Constructor
9    {
10     super();
11     initialize();
12   }
13   private void initialize() {
14     this.setSize(300, 200);
15     this.setContentPane(getJContentPane());
16     setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
17     this.setVisible(true);
18   }
19   private javax.swing.JPanel getJContentPane() // initializes jContentPane
20   {
21     if (jContentPane == null) {
22       jContentPane = new javax.swing.JPanel();
23       jContentPane.setLayout(new GridLayout(2, 1)); // Zeilen, Spalten
24       jContentPane.add(getJTextPane(), null);
25       jContentPane.add(getJButton(), null);
26     }
27     return jContentPane;
28   }
29   private javax.swing.JTextPane getJTextPane() // method initializes
30                         // jTextPane @return
31                         // javax.swing.JTextPane
32   {
33     if (jTextPane == null) {
34       jTextPane = new javax.swing.JTextPane();
35     }
36     jTextPane.setEditable(false);
37     return jTextPane;
38   }
39   private javax.swing.JButton getJButton() // method initializes jButton,
40                       // @return javax.swing.JButton
41   {
42     if (jButton == null) { // Ereignis bei Klick
43       jButton = new javax.swing.JButton();
44     }
45     jButton.setText("test");
```

```
46      jButton.addMouseListener(new java.awt.event.MouseListener() {
47        public void mouseClicked(java.awt.event.MouseEvent e) {
48          System.out.println("mouseClicked()");
49          jTextPane.setText("juhu");
50        }
51        public void mouseEntered(java.awt.event.MouseEvent e) {
52        }
53        public void mouseExited(java.awt.event.MouseEvent e) {
54        }
55        public void mousePressed(java.awt.event.MouseEvent e) {
56        }
57        public void mouseReleased(java.awt.event.MouseEvent e) {
58        }
59      });
60      return jButton;
61    }
62    public static void main(String[] args) {
63      SwingTest calculator = new SwingTest();
64    }
65  }
```

Quelltext A.1: Eclipse: SwingTest.java

A.1.2. Das 2. Java-Projekt: Einfacher Taschenrechner in Swing mit Hilfe des Visual Editors

```
1  import java.awt.*;
2  import javax.swing.JFrame;
3
4  public class Taschenrechner3 extends JFrame {
5    private javax.swing.JPanel jPInhalt = null;
6    private javax.swing.JTextField jTFZahl1 = null;
7    private javax.swing.JLabel jLabel = null;
8    private javax.swing.JLabel jLabel1 = null;
9    private javax.swing.JTextField jTFZahl2 = null;
10   private javax.swing.JTextField jTFErgebnis = null;
11   private javax.swing.JButton jBAddieren = null;
12   private javax.swing.JButton jBSubtrahieren = null;
13   private javax.swing.JButton jBDividieren = null;
14   private javax.swing.JButton jBDividieren_real = null;
15   private javax.swing.JLabel jLabel2 = null;
16   private javax.swing.JButton jBMultiplizieren = null;
17   private javax.swing.JPanel jPDisplay = null;
18   private javax.swing.JPanel jPButtons = null;
19   public static void main(String[] args) {
20     Taschenrechner3 calculator = new Taschenrechner3();
21   }
22   public Taschenrechner3() // Default Constructor
23   {
24     super();
25     initialize();
26   }
27   private javax.swing.JPanel getJPInhalt() // initializes jPInhalt
28   {
29     if (jPInhalt == null) {
30       jPInhalt = new javax.swing.JPanel();
31       jPInhalt.add(getJPDisplay(), null);
32       jPInhalt.add(getJPButtons(), null);
33       jPInhalt.setLayout(new GridLayout(1, 2)); // Zeilen, Spalten
34     }
35     return jPInhalt;
```

```
36    }
37    private void initialize() {
38      this.setSize(300, 200);
39      this.setContentPane(getJPInhalt());
40      this.setTitle(" Taschenrechner in Swing");
41      setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
42      this.setVisible(true);
43    }
44    private javax.swing.JTextField getJTFZahl1() {
45      if (jTFZahl1 == null) {
46        jTFZahl1 = new javax.swing.JTextField();
47        jTFZahl1.setText("");
48      }
49      return jTFZahl1;
50    }
51    private javax.swing.JLabel getJLabel() {
52      if (jLabel == null) {
53        jLabel = new javax.swing.JLabel();
54        jLabel.setText("Zahl 1");
55      }
56      return jLabel;
57    }
58    private javax.swing.JLabel getJLabel1() {
59      if (jLabel1 == null) {
60        jLabel1 = new javax.swing.JLabel();
61        jLabel1.setText("Zahl 2");
62      }
63      return jLabel1;
64    }
65    private javax.swing.JTextField getJTFZahl2() {
66      if (jTFZahl2 == null) {
67        jTFZahl2 = new javax.swing.JTextField();
68      }
69      return jTFZahl2;
70    }
71    private javax.swing.JTextField getJTFErgebnis() {
72      if (jTFErgebnis == null) {
73        jTFErgebnis = new javax.swing.JTextField();
74      }
75      return jTFErgebnis;
76    }
77    private javax.swing.JButton getJBAddieren() {
78      if (jBAddieren == null) {
79        jBAddieren = new javax.swing.JButton();
80        jBAddieren.setText("+");
81        jBAddieren.addActionListener(new java.awt.event.ActionListener() {
82          public void actionPerformed(java.awt.event.ActionEvent e) {
83            System.out.println(" + actionPerformed()");
84            int iZahl1, iZahl2;
85            iZahl1 = Integer.parseInt(jTFZahl1.getText());
86            iZahl2 = Integer.parseInt(jTFZahl2.getText());
87            jTFErgebnis.setText((iZahl1 + iZahl2) + " ");
88          }
89        });
90      }
91      return jBAddieren;
92    }
93    private javax.swing.JButton getJButton1() {
94      if (jBSubtrahieren == null) {
95        jBSubtrahieren = new javax.swing.JButton();
96        jBSubtrahieren.setText("-");
97        jBSubtrahieren
98            .addActionListener(new java.awt.event.ActionListener() {
99              public void actionPerformed(java.awt.event.ActionEvent e) {
100               System.out
101                   .println("Subtraktion - actionPerformed()");
```

```
102               int iZahl1, iZahl2;
103               iZahl1 = Integer.parseInt(jTFZahl1.getText());
104               iZahl2 = Integer.parseInt(jTFZahl2.getText());
105               jTFErgebnis.setText((iZahl1 - iZahl2) + " ");
106             }
107           });
108     }
109     return jBSubtrahieren;
110   }
111   private javax.swing.JButton getJBSubtrahieren() {
112     if (jBSubtrahieren == null) {
113       jBSubtrahieren = new javax.swing.JButton();
114       jBSubtrahieren.setText("*");
115     }
116     return jBSubtrahieren;
117   }
118   private javax.swing.JButton getJBDividieren() {
119     if (jBDividieren == null) {
120       jBDividieren = new javax.swing.JButton();
121       jBDividieren.setText("Div (int)");
122       jBDividieren.addActionListener(new java.awt.event.ActionListener() {
123         public void actionPerformed(java.awt.event.ActionEvent e) {
124           System.out
125               .println("Division durch int - actionPerformed()");
126           int iZahl1, iZahl2;
127           iZahl1 = Integer.parseInt(jTFZahl1.getText());
128           iZahl2 = Integer.parseInt(jTFZahl2.getText());
129           jTFErgebnis.setText((iZahl1 / iZahl2) + " ");
130         }
131       });
132     }
133     return jBDividieren;
134   }
135   private javax.swing.JButton getJBDividieren_real() {
136     if (jBDividieren_real == null) {
137       jBDividieren_real = new javax.swing.JButton();
138       jBDividieren_real.setText("Div (real)");
139       jBDividieren_real
140           .addActionListener(new java.awt.event.ActionListener() {
141             public void actionPerformed(java.awt.event.ActionEvent e) {
142               System.out
143                   .println("Division durch real - actionPerformed()");
144               int iZahl1, iZahl2;
145               double dbZahl2 = 1;
146               iZahl1 = Integer.parseInt(jTFZahl1.getText());
147               iZahl2 = Integer.parseInt(jTFZahl2.getText());
148               jTFErgebnis.setText((iZahl1 / dbZahl2) + " ");
149             }
150           });
151     }
152     return jBDividieren_real;
153   }
154   private javax.swing.JLabel getJLabel2() {
155     if (jLabel2 == null) {
156       jLabel2 = new javax.swing.JLabel();
157       jLabel2.setText("Ergebnis");
158     }
159     return jLabel2;
160   }
161   private javax.swing.JButton getJBMultiplizieren() {
162     if (jBMultiplizieren == null) {
163       jBMultiplizieren = new javax.swing.JButton();
164       jBMultiplizieren.setText("x");
165       jBMultiplizieren
166           .addActionListener(new java.awt.event.ActionListener() {
167             public void actionPerformed(java.awt.event.ActionEvent e) {
```

```
168                 System.out
169                     .println("Multiplikation - actionPerformed()");
170                 int iZahl1, iZahl2;
171                 iZahl1 = Integer.parseInt(jTFZahl1.getText());
172                 iZahl2 = Integer.parseInt(jTFZahl2.getText());
173                 jTFErgebnis.setText((iZahl1 * iZahl2) + " ");
174               }
175             });
176     }
177     return jBMultiplizieren;
178   }
179   private javax.swing.JPanel getJPDisplay() {
180     if (jPDisplay == null) {
181       jPDisplay = new javax.swing.JPanel();
182       jPDisplay.add(getJLabel(), null);
183       jPDisplay.add(getJTFZahl1(), null);
184       jPDisplay.add(getJLabel1(), null);
185       jPDisplay.add(getJTFZahl2(), null);
186       jPDisplay.add(getJLabel2(), null);
187       jPDisplay.add(getJTFErgebnis(), null);
188       jPDisplay.setLayout(new GridLayout(6, 1)); // Zeilen, Spalten
189     }
190     return jPDisplay;
191   }
192   private javax.swing.JPanel getJPButtons() {
193     if (jPButtons == null) {
194       jPButtons = new javax.swing.JPanel();
195       jPButtons.add(getJBAddieren(), null);
196       jPButtons.add(getJButton1(), null);
197       jPButtons.add(getJBMultiplizieren(), null);
198       jPButtons.add(getJBDividieren(), null);
199       jPButtons.add(getJBDividieren_real(), null);
200       jPButtons.setLayout(new GridLayout(5, 1)); // Zeilen, Spalten
201     }
202     return jPButtons;
203   }
204 } // @jve:visual-info decl-index=0 visual-constraint="17,16"
```

Quelltext A.2: Eclipse: Taschenrechner3.java

A.1.3. Das 3. Java-Projekt: Einfacher Taschenrechner in Swing vereinfacht und ohne Unterstützung des Visual Editors

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class Taschenrechner extends JFrame implements ActionListener {
6    private JPanel[] p = new JPanel[2];
7    private JButton[] buttons = new JButton[5];
8    private JTextField tfZahl1 = new JTextField(10);
9    private JTextField tfZahl2 = new JTextField(10);
10   private JTextField resultField = new JTextField(10);
11   public Taschenrechner() // Konstruktor
12   {
13     getContentPane().setLayout(new GridLayout(1, 4));
14     // Grid(Zeilen, Spalten)
15     p[0] = new JPanel(new GridLayout(6, 1)); // Operanten/Ergebnis-Feld
16     p[1] = new JPanel(new GridLayout(5, 1)); // Operationsbuttons
17     p[0].add(new JLabel("Zahl A")); // Eingabe und Ergebnisfelder
18     p[0].add(tfZahl1);
```

```
19      p[0].add(new JLabel("Zahl b"));
20      p[0].add(tfZahl2);
21      p[0].add(new JLabel("Ergebnis"));
22      p[0].add(resultField);
23      buttons[0] = new JButton("+"); // Buttons vorbereiten
24      buttons[0].setToolTipText("Addieren");
25      buttons[1] = new JButton("-");
26      buttons[1].setToolTipText("Subtrahieren");
27      buttons[2] = new JButton("*");
28      buttons[2].setToolTipText("Multiplizieren");
29      buttons[3] = new JButton("int /");
30      buttons[3].setToolTipText("int-Zahlen dividieren");
31      buttons[4] = new JButton("real /");
32      buttons[4].setToolTipText("real-Zahlen dividieren");
33      buttons[0].addActionListener(this);
34      buttons[1].addActionListener(this);
35      buttons[2].addActionListener(this);
36      buttons[3].addActionListener(this);
37      buttons[4].addActionListener(this);
38      for (int i = 0; i < 5; i++) {
39        p[1].add(buttons[i]);
40      }
41      for (int i = 0; i < 2; i++) // Panels zum Frame hinzufuegen
42      {
43        getContentPane().add(p[i]);
44      }
45      addWindowListener(new WindowAdapter() // Window schliessen
46      {
47        public void windowClosing(WindowEvent e) {
48          System.exit(0);
49        }
50      });
51      setTitle(" Taschenrechner in Swing");
52      setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
53      setSize(300, 200);
54      show();
55    } // Taschenrechner() Konstruktor
56    public void actionPerformed(ActionEvent event) {
57      String command = event.getActionCommand();
58      int iZahl1, iZahl2;
59      double dbZahl2;
60      if (command == "+") {
61        iZahl1 = Integer.parseInt(tfZahl1.getText());
62        iZahl2 = Integer.parseInt(tfZahl2.getText());
63        resultField.setText((iZahl1 + iZahl2) + " ");
64      } else if (command == "-") {
65        iZahl1 = Integer.parseInt(tfZahl1.getText());
66        iZahl2 = Integer.parseInt(tfZahl2.getText());
67        resultField.setText((iZahl1 - iZahl2) + " ");
68      } else if (command == "*") {
69        iZahl1 = Integer.parseInt(tfZahl1.getText());
70        iZahl2 = Integer.parseInt(tfZahl2.getText());
71        resultField.setText((iZahl1 * iZahl2) + " ");
72      } else if (command == "int /") {
73        iZahl1 = Integer.parseInt(tfZahl1.getText());
74        iZahl2 = Integer.parseInt(tfZahl2.getText());
75        try {
76          resultField.setText((iZahl1 / iZahl2) + " ");
77        } catch (ArithmeticException ae) {
78          resultField.setText(" Division durch Null ");
79          System.out.println("Divisions-Problem");
80          System.out.println(ae);
81        }
82      } else if (command == "real /") {
83        iZahl1 = Integer.parseInt(tfZahl1.getText());
84        dbZahl2 = Integer.parseInt(tfZahl2.getText());
```

```
85        try {
86          resultField.setText((iZahl1 / dbZahl2) + " ");
87        } catch (ArithmeticException ae) {
88          resultField.setText(" Division durch Null ");
89          System.out.println("Divisions-Problem");
90          System.out.println(ae);
91        }
92      } // if
93    } // actionPerformed()
94    public static void main(String[] args) {
95      Taschenrechner calculator = new Taschenrechner();
96    }
97  } // @jve:visual-info decl-index=0 visual-constraint="14,10" class
98    // Taschenrechner
```

Quelltext A.3: Eclipse: Taschenrechner.java

A.2. SOAP / AXIS

A.2.1. Quellcodes JavaSoapExample

```
1  /*
2   * Created on 11.05.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  package javaSoapExample;
8
9  /**
10  * @author Marco Junge
11  *
12  * To change the template for this generated type comment go to
13  * Window - Preferences - Java - Code Generation - Code and Comments
14  */
15 public class Calculator {
16   public int add(int param1, int param2){
17     return (param1 + param2);
18   }
19 }
```

Quelltext A.4: SOAP: Calculator.java

```
1  /*
2   * Created on 11.05.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  package javaSoapExample;
8
9  /**
10  * @author Marco Junge
11  *
12  * To change the template for this generated type comment go to
13  * Window - Preferences - Java - Code Generation - Code and Comments
14  */
15 import java.net.URL;
16 import java.util.Vector;
17 import org.apache.soap.*;
18 import org.apache.soap.rpc.*;
19
20 public class InvokeCalculator {
21   public static void main(String[] args) throws Exception{
22     URL url = new URL("http://localhost:8080/soap/servlet/rpcrouter");
23
24     Call myCall = new Call();
25     myCall.setTargetObjectURI("urn:CallCalculator");
26     myCall.setMethodName("add");
27     myCall.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
28
29     Vector myParams = new Vector();
30     myParams.addElement(new Parameter("param1", Integer.class, "427", null));
31     myParams.addElement(new Parameter("param2", Integer.class, "7142", null));
32
33     myCall.setParams(myParams);
34
35     Response resp = myCall.invoke(url, "");
36
37     if(resp.generatedFault()){
```

```
38        Fault fault = resp.getFault();
39        System.out.println("Fault-Code: " + fault.getFaultCode());
40        System.out.println("Fault-String: " + fault.getFaultString());
41      } else {
42        Parameter ret = resp.getReturnValue();
43        Integer value = (Integer)ret.getValue();
44        System.out.println("Result is " + value);
45      }
46    }
47  }
```

Quelltext A.5: SOAP: InvokeCalculator.java

```
1  <isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
2            id="urn:CallCalculator">
3    <isd:provider type="java" scope="Application" methods="add">
4    <isd:java class="javaSoapExample.Calculator"
5      static="false"/>
6    </isd:provider>
7    <isd:faultListener>
8      org.apache.soap.DOMFaultListener
9    </isd:faultListener>
10 </isd:service>
```

Quelltext A.6: SOAP: CallCalculator.dd

A.2.2. Quellcodes JavaAxisExample

```
1  /*
2   * Created on 11.05.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  /**
8   * @author Marco Junge
9   *
10  * To change the template for this generated type comment go to
11  * Window - Preferences - Java - Code Generation - Code and Comments
12  */
13 public class Calculator {
14   public int add(int param1, int param2){
15     return (param1 + param2);
16   }
17 }
```

Quelltext A.7: SOAP: Calculator.java

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions targetNamespace="http://localhost:8080/axis/Calculator.jws"
3      xmlns="http://schemas.xmlsoap.org/wsdl/"
4      xmlns:apachesoap="http://xml.apache.org/xml-soap"
5      xmlns:impl="http://localhost:8080/axis/Calculator.jws"
6      xmlns:intf="http://localhost:8080/axis/Calculator.jws"
7      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
8      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9      xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
10     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11   <wsdl:message name="addResponse">
12     <wsdl:part name="addReturn" type="xsd:int"/>
13   </wsdl:message>
```

```
14   <wsdl:message name="addRequest">
15     <wsdl:part name="param1" type="xsd:int"/>
16     <wsdl:part name="param2" type="xsd:int"/>
17   </wsdl:message>
18   <wsdl:portType name="Calculator">
19     <wsdl:operation name="add" parameterOrder="param1 param2">
20       <wsdl:input message="impl:addRequest" name="addRequest"/>
21       <wsdl:output message="impl:addResponse" name="addResponse"/>
22     </wsdl:operation>
23   </wsdl:portType>
24   <wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
25     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
26     <wsdl:operation name="add">
27       <wsdlsoap:operation soapAction=""/>
28       <wsdl:input name="addRequest">
29         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
30         namespace="http://DefaultNamespace" use="encoded"/>
31       </wsdl:input>
32       <wsdl:output name="addResponse">
33         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
34         namespace="http://localhost:8080/axis/Calculator.jws" use="encoded"/>
35       </wsdl:output>
36     </wsdl:operation>
37   </wsdl:binding>
38   <wsdl:service name="CalculatorService">
39     <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
40       <wsdlsoap:address location="http://localhost:8080/axis/Calculator.jws"/>
41     </wsdl:port>
42   </wsdl:service>
43 </wsdl:definitions>
```

Quelltext A.8: SOAP: Calculator.wsdl

```
1 <undeployment xlmns="http://xml.apache.org/axis/wsdd/">
2   <service name="CalculatorService"/>
3 </undeployment>
```

Quelltext A.9: SOAP: CalculatorUndeploy.wsdl

```
1  /**
2   * Calculator.java
3   *
4   * This file was auto-generated from WSDL
5   * by the Apache Axis WSDL2Java emitter.
6   */
7
8  package localhost.axis.Calculator_jws;
9
10 public interface Calculator extends java.rmi.Remote {
11     public int add(int param1, int param2) throws java.rmi.RemoteException;
12 }
```

Quelltext A.10: SOAP: localhost\axis\Calculator_jws\Calculator.java

```
1  /**
2   * CalculatorService.java
3   *
4   * This file was auto-generated from WSDL
5   * by the Apache Axis WSDL2Java emitter.
6   */
7
8  package localhost.axis.Calculator_jws;
9
10 public interface CalculatorService extends javax.xml.rpc.Service {
```

```
11      public java.lang.String getCalculatorAddress();
12
13      public localhost.axis.Calculator_jws.Calculator getCalculator()
14        throws javax.xml.rpc.ServiceException;
15
16      public localhost.axis.Calculator_jws.Calculator
17        getCalculator(java.net.URL portAddress)
18        throws javax.xml.rpc.ServiceException;
19  }
```

Quelltext A.11: SOAP: localhost\axis\Calculator_jws\CalculatorService.java

```
1   /**
2    * CalculatorServiceLocator.java
3    *
4    * This file was auto-generated from WSDL
5    * by the Apache Axis WSDL2Java emitter.
6    */
7
8   package localhost.axis.Calculator_jws;
9
10  public class CalculatorServiceLocator extends org.apache.axis.client.Service
11        implements localhost.axis.Calculator_jws.CalculatorService {
12
13      // Use to get a proxy class for Calculator
14      private final java.lang.String Calculator_address =
15        "http://localhost:8080/axis/Calculator.jws";
16
17      public java.lang.String getCalculatorAddress() {
18          return Calculator_address;
19      }
20
21      // The WSDD service name defaults to the port name.
22      private java.lang.String CalculatorWSDDServiceName = "Calculator";
23
24      public java.lang.String getCalculatorWSDDServiceName() {
25          return CalculatorWSDDServiceName;
26      }
27
28      public void setCalculatorWSDDServiceName(java.lang.String name) {
29          CalculatorWSDDServiceName = name;
30      }
31
32      public localhost.axis.Calculator_jws.Calculator getCalculator()
33          throws javax.xml.rpc.ServiceException {
34         java.net.URL endpoint;
35          try {
36              endpoint = new java.net.URL(Calculator_address);
37          }
38          catch (java.net.MalformedURLException e) {
39              throw new javax.xml.rpc.ServiceException(e);
40          }
41          return getCalculator(endpoint);
42      }
43
44      public localhost.axis.Calculator_jws.Calculator
45          getCalculator(java.net.URL portAddress)
46          throws javax.xml.rpc.ServiceException {
47          try {
48              localhost.axis.Calculator_jws.CalculatorSoapBindingStub _stub =
49            new localhost.axis.Calculator_jws.CalculatorSoapBindingStub(portAddress, this);
50              _stub.setPortName(getCalculatorWSDDServiceName());
51              return _stub;
52          }
53          catch (org.apache.axis.AxisFault e) {
54              return null;
```

```
55          }
56      }
57
58      /**
59       * For the given interface, get the stub implementation.
60       * If this service has no port for the given interface,
61       * then ServiceException is thrown.
62       */
63      public java.rmi.Remote getPort(Class serviceEndpointInterface)
64          throws javax.xml.rpc.ServiceException {
65          try {
66              if (localhost.axis.Calculator_jws.Calculator.class.
67              isAssignableFrom(serviceEndpointInterface)) {
68                  localhost.axis.Calculator_jws.CalculatorSoapBindingStub _stub =
69              new localhost.axis.Calculator_jws.CalculatorSoapBindingStub(new
70                  java.net.URL(Calculator_address), this);
71                  _stub.setPortName(getCalculatorWSDDServiceName());
72                  return _stub;
73              }
74          }
75          catch (java.lang.Throwable t) {
76              throw new javax.xml.rpc.ServiceException(t);
77          }
78          throw new javax.xml.rpc.ServiceException(
79          "There is no stub implementation for the interface:  "
80          + (serviceEndpointInterface == null ? "null"
81            : serviceEndpointInterface.getName()));
82      }
83
84      /**
85       * For the given interface, get the stub implementation.
86       * If this service has no port for the given interface,
87       * then ServiceException is thrown.
88       */
89      public java.rmi.Remote getPort(javax.xml.namespace.QName portName,
90          Class serviceEndpointInterface) throws javax.xml.rpc.ServiceException {
91          if (portName == null) {
92              return getPort(serviceEndpointInterface);
93          }
94          String inputPortName = portName.getLocalPart();
95          if ("Calculator".equals(inputPortName)) {
96              return getCalculator();
97          }
98          else  {
99              java.rmi.Remote _stub = getPort(serviceEndpointInterface);
100             ((org.apache.axis.client.Stub) _stub).setPortName(portName);
101             return _stub;
102         }
103     }
104
105     public javax.xml.namespace.QName getServiceName() {
106         return new javax.xml.namespace.QName("http://localhost:8080/axis/Calculator.jws"
107         , "CalculatorService");
108     }
109
110     private java.util.HashSet ports = null;
111
112     public java.util.Iterator getPorts() {
113         if (ports == null) {
114             ports = new java.util.HashSet();
115             ports.add(new javax.xml.namespace.QName("Calculator"));
116         }
117         return ports.iterator();
118     }
119
120 }
```

Quelltext A.12: SOAP: localhost\axis\Calculator_jws\CalculatorServiceLocator.java

```
/**
 * CalculatorSoapBindingStub.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

package localhost.axis.Calculator_jws;

public class CalculatorSoapBindingStub extends org.apache.axis.client.Stub
      implements localhost.axis.Calculator_jws.Calculator {
    private java.util.Vector cachedSerClasses = new java.util.Vector();
    private java.util.Vector cachedSerQNames = new java.util.Vector();
    private java.util.Vector cachedSerFactories = new java.util.Vector();
    private java.util.Vector cachedDeserFactories = new java.util.Vector();

    static org.apache.axis.description.OperationDesc [] _operations;

    static {
        _operations = new org.apache.axis.description.OperationDesc[1];
        org.apache.axis.description.OperationDesc oper;
        oper = new org.apache.axis.description.OperationDesc();
        oper.setName("add");
        oper.addParameter(new javax.xml.namespace.QName("", "param1"),
        new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "int"),
        int.class, org.apache.axis.description.ParameterDesc.IN, false, false);
        oper.addParameter(new javax.xml.namespace.QName("", "param2"),
        new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "int"),
        int.class, org.apache.axis.description.ParameterDesc.IN, false, false);
        oper.setReturnType(new javax.xml.namespace.QName(
        "http://www.w3.org/2001/XMLSchema", "int"));
        oper.setReturnClass(int.class);
        oper.setReturnQName(new javax.xml.namespace.QName("", "addReturn"));
        oper.setStyle(org.apache.axis.enum.Style.RPC);
        oper.setUse(org.apache.axis.enum.Use.ENCODED);
        _operations[0] = oper;

    }

    public CalculatorSoapBindingStub() throws org.apache.axis.AxisFault {
         this(null);
    }

    public CalculatorSoapBindingStub(
        java.net.URL endpointURL, javax.xml.rpc.Service service)
        throws org.apache.axis.AxisFault {
         this(service);
         super.cachedEndpoint = endpointURL;
    }

    public CalculatorSoapBindingStub(javax.xml.rpc.Service service)
        throws org.apache.axis.AxisFault {
        if (service == null) {
            super.service = new org.apache.axis.client.Service();
        } else {
            super.service = service;
        }
    }

    private org.apache.axis.client.Call createCall() throws java.rmi.RemoteException {
        try {
            org.apache.axis.client.Call _call =
```

```
63                      (org.apache.axis.client.Call) super.service.createCall();
64              if (super.maintainSessionSet) {
65                  _call.setMaintainSession(super.maintainSession);
66              }
67              if (super.cachedUsername != null) {
68                  _call.setUsername(super.cachedUsername);
69              }
70              if (super.cachedPassword != null) {
71                  _call.setPassword(super.cachedPassword);
72              }
73              if (super.cachedEndpoint != null) {
74                  _call.setTargetEndpointAddress(super.cachedEndpoint);
75              }
76              if (super.cachedTimeout != null) {
77                  _call.setTimeout(super.cachedTimeout);
78              }
79              if (super.cachedPortName != null) {
80                  _call.setPortName(super.cachedPortName);
81              }
82              java.util.Enumeration keys = super.cachedProperties.keys();
83              while (keys.hasMoreElements()) {
84                  java.lang.String key = (java.lang.String) keys.nextElement();
85                  _call.setProperty(key, super.cachedProperties.get(key));
86              }
87              return _call;
88          }
89          catch (java.lang.Throwable t) {
90              throw new org.apache.axis.AxisFault("Failure trying to get the Call object",
91            t);
92          }
93      }
94
95      public int add(int param1, int param2) throws java.rmi.RemoteException {
96          if (super.cachedEndpoint == null) {
97              throw new org.apache.axis.NoEndPointException();
98          }
99          org.apache.axis.client.Call _call = createCall();
100         _call.setOperation(_operations[0]);
101         _call.setUseSOAPAction(true);
102         _call.setSOAPActionURI("");
103         _call.setSOAPVersion(org.apache.axis.soap.SOAPConstants.SOAP11_CONSTANTS);
104         _call.setOperationName(new javax.xml.namespace.QName("http://DefaultNamespace",
105         "add"));
106
107         setRequestHeaders(_call);
108         setAttachments(_call);
109         java.lang.Object _resp = _call.invoke(new java.lang.Object[] {
110         new java.lang.Integer(param1), new java.lang.Integer(param2)});
111
112         if (_resp instanceof java.rmi.RemoteException) {
113             throw (java.rmi.RemoteException)_resp;
114         }
115         else {
116             extractAttachments(_call);
117             try {
118                 return ((java.lang.Integer) _resp).intValue();
119             } catch (java.lang.Exception _exception) {
120                 return ((java.lang.Integer) org.apache.axis.utils.JavaUtils.convert(
121             _resp,int.class)).intValue();
122             }
123         }
124     }
125
126 }
```

Quelltext A.13: SOAP: localhost\axis\Calculator_jws\CalculatorSoapBindingStub.java

```
1  /*
2   * Created on 12.05.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  /**
8   * @author Marco Junge
9   *
10  * To change the template for this generated type comment go to
11  * Window - Preferences - Java - Code Generation - Code and Comments
12  */
13 package localhost.axis.Calculator_jws;
14
15 public class InvokeCalculator {
16   public static void main(String[] args) throws Exception{
17   CalculatorService myService = new CalculatorServiceLocator();
18   Calculator myPort = myService.getCalculator();
19
20   System.out.println("Das Ergebnis lautet: " + myPort.add(12,5));
21   }
22 }
```

Quelltext A.14: SOAP: localhost\axis\Calculator_jws\InvokeCalculator.java

A.2.3. Axis–Installationsschritte

Download

Die ca. 10 MB großen Apache–Axis–Binaries herunterladen unter:

ftp://www.bwl.hs-bremerhaven.de/pub/bizweb/tools/axis/
oder alternativ unter
http://mirrorspace.org/apache/ws/axis/1_1/axis-1_1.zip

Zusätzlich sollten auch die Xerces–Binaries (5,6 MB) heruntergeladen werden (optional):

http://mirrorspace.org/apache/xml/xerces-j/Xerces-J-bin.2.6.2.zip

Anmerkung: Die Xerces–Datei auf unserem FTP–Server ist fehlerhaft.

Entpacken

Die Datei `axis-1_1.zip` muss nach C:/Java/axis/ und die Datei `xerces-2_6_2` muss in das Verzeichnis C:/Java/xerces/ entpackt werden. Alternativ könnten hier natürlich auch andere Zielverzeichnisse angegeben werden.

In den Tomcat–Server einbinden

Den `webapps`–Ordner aus dem AXIS–Verzeichnis ins Tomcat–Verzeichnis kopieren, sodass dann im Tomcat `webapps`–Ordner jetzt auch ein Ordner `axis` vorhanden ist.

Weitere Libraries einbinden (optional, s.o.)

Die nachfolgenden Libraries sollten zusätzlich in das folgende Verzeichnis kopiert werden: C:/Java/tomcat/webapps/axis/lib:

- `xercesImpl.jar`
- `xercesSamples.jar`
- `xml-apis.jar`
- `xmlParserAPIs.jar`

Alle Dateien sind Bestandteil des Xerces–Verzeichnisses

Umgebungsvariablen setzen

Unter Arbeitsplatz, „Systeminformationen anzeigen“ auswählen...

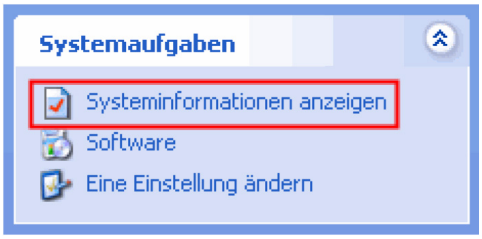


Abbildung A.1.: SOAP: Systeminformationen anzeigen

...dann...

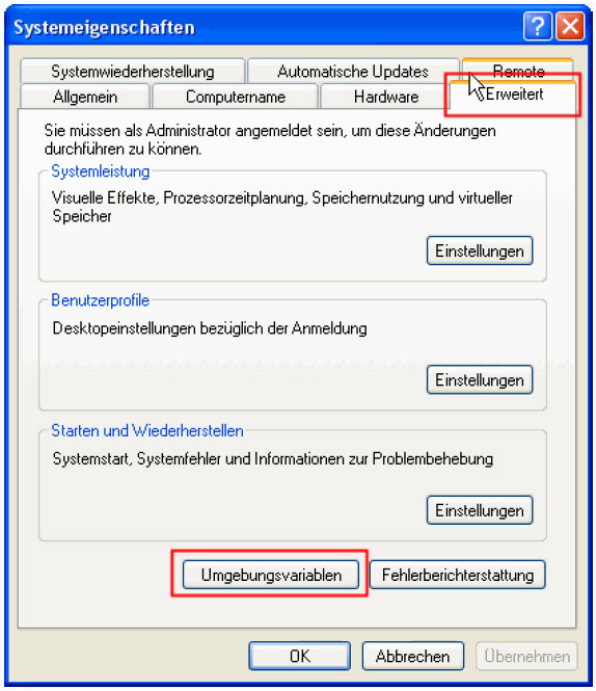


Abbildung A.2.: SOAP: Systemeigenschaften

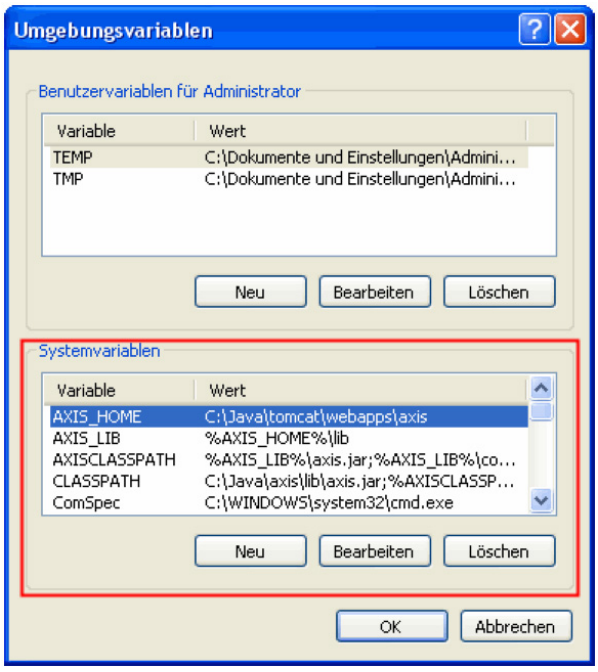


Abbildung A.3.: SOAP: Umgebungsvariablen

Es müssen folgende Umgebungsvariablen gesetzt werden:

AXIS_HOME	=	C:\Java\tomcat\webapps\axis
AXIS_LIB	=	%AXIS_HOME%\lib
AXIS_CLASSPATH	=	%AXIS_LIB%\axis.jar; %AXIS_LIB%\axis-ant.jar; %AXIS_LIB%\commons-discovery.jar; %AXIS_LIB%\commons-logging.jar; %AXIS_LIB%\jaxrpc.jar; %AXIS_LIB%\log4j-1.2.8.jar; %AXIS_LIB%\saaj.jar; %AXIS_LIB%\wsdl4j.jar *optional, s.o.:* %AXIS_LIB%\xercesImpl.jar; %AXIS_LIB%\xercesSamples.jar; %AXIS_LIB%\xml-apis.jar; %AXIS_LIB%\xmlParserAPIs.jar
JAVA_HOME (bzw. JAVA)	=	Pfad zum JDK, z.B. C:\Java\AppServer\jdk
CLASSPATH	=	%AXIS_CLASSPATH% (ggf. vorhandene Einträge um diesen erweitern)

Anmerkung: Zum Setzen von Umgebungsvariablen sind Administratorrechte erforderlich!

Tomcat–Server starten

Den Tomcat–Server starten (z.B. über Eclipse) und im Browser die folgende Seite aufrufen: http://localhost:8080/axis. Die Axis–Startseite gibt Aufschluss darüber, ob alles geklappt hat:

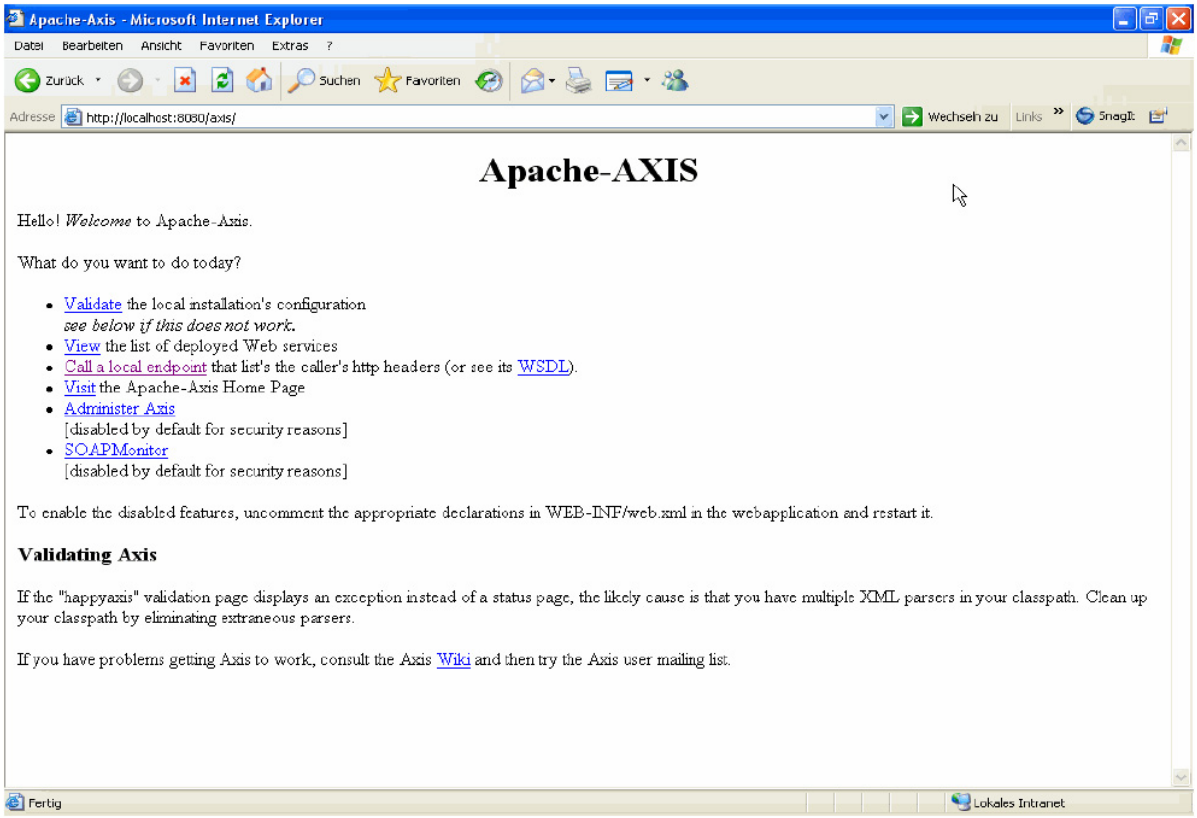


Abbildung A.4.: SOAP: AXIS–Startseite

Webservices mit Ecplise

Die Programmierung der Webservices erfordert das Einbinden der Build–Paths (wichtig, sonst läuft nichts!). Dazu auf das Java–Projekt mit der rechten Maustaste, dann unter Properties → Java Build Path → Libraries → Add JARs. . . die JARs aus dem AXIS–Lib– Verzeichnis einbinden.
Alternativ kann auch über Add Variable. . . eine entsprechende Variable konfiguriert werden und mit Extend. . . dann die JARs aus dem lib–Verzeichnis einbinden.

B. Beispielentwicklungen

B.1. Taschenrechner

B.1.1. Listing 1: Die Java-Klasse: Oberfläche

```
1  /*
2   * Created on 19.10.2004
3   *
4   * To change the template for this generated file go to
5   * Window - Preferences - Java - Code Generation - Code and Comments
6   */
7  package localhost.axis.Taschenrech_jws;
8
9  import javax.swing.*;
10 import java.awt.event.*;
11
12 /**
13  * @author Christian Lefke
14  *
15  * GUI für den Webservice Taschenrechner
16  */
17
18
19 public class Oberflaeche extends JFrame {
20   double wert1;
21   double wert2;
22   double erg;
23   JTextField Eingabe1;
24   JTextField Eingabe2;
25   JTextField Ergebniss;
26   JComboBox combo;
27   JLabel jLabel1 = new JLabel();
28   JLabel jLabel2 = new JLabel();
29   JLabel jLabel3 = new JLabel();
30   JLabel jLabel4 = new JLabel();
31   JLabel jLabel5 = new JLabel();
32   private static final String[] Operatoren
33     ={"Addition","Subtraktion","Multiplikation","Division","Prozent","Wurzel"};
34
35   public Oberflaeche ()
36   {
37     super();
38     initialize();
39   }
40
41   private void initialize() {
42     this.setContentPane(getJPanel());
43     this.setSize(600, 400);
44     this.setTitle("Taschenrechner");
45     this.setVisible(true);
46     this.setLocation(200, 100);
47     this.setResizable(false);
```

```
48
49      //    EventHandler für Fenster schliessen einrichten
50      this.addWindowListener(new WindowAdapter() {
51        public void windowClosing(WindowEvent e) {
52          setVisible(false);
53          dispose();
54        }
55      });
56    }
57
58    private JPanel getJPanel() {
59      JPanel panel = new JPanel();
60      panel.setLayout(null);
61      panel.add(getJButton(), null);
62      panel.add(getEingabe1(), null);
63      panel.add(getEingabe2(), null);
64      panel.add(getErgebniss(), null);
65      panel.add(getcombo(), null);
66      jLabel1.setText("1. Wert:");
67      jLabel1.setSize(50, 20);
68      jLabel1.setLocation(150, 50);
69      panel.add(jLabel1, null);
70      jLabel2.setText("2. Wert:");
71      jLabel2.setSize(50, 20);
72      jLabel2.setLocation(150, 200);
73      panel.add(jLabel2, null);
74      jLabel3.setText("Operator wählen:");
75      jLabel3.setSize(100, 20);
76      jLabel3.setLocation(98, 100);
77      panel.add(jLabel3, null);
78      jLabel4.setText("Hinweis Prozentrechnung: 1.Wert Ist-Wert und 2.Wert maximal Wert!");
79      jLabel4.setSize(600, 50);
80      jLabel4.setLocation(50, 320);
81      panel.add(jLabel4, null);
82      jLabel5.setText("Hinweis Wurzelrechnung: Es muss nur ein Wert eingegeben werden!");
83      jLabel5.setSize(600, 20);
84      jLabel5.setLocation(50, 350);
85      panel.add(jLabel5, null);
86      pack();
87      return panel;
88    }
89
90    private JTextField getEingabe1(){
91      if(Eingabe1 == null) {
92        Eingabe1 = new JTextField();
93        Eingabe1.setSize(150, 20);
94        Eingabe1.setLocation(220, 50);
95      }
96      return Eingabe1;
97    }
98
99    private JComboBox getcombo(){
100     if(combo == null) {
101       combo = new JComboBox(Operatoren);
102       combo.setSize(110,20);
103       combo.setLocation(240,100);
104     }
105     return combo;
106   }
107
108   private JTextField getEingabe2(){
109     if(Eingabe2 == null) {
110       Eingabe2 = new JTextField();
111       Eingabe2.setSize(150, 20);
112       Eingabe2.setLocation(220, 200);
113     }
```

```
114     return Eingabe2;
115   }
116
117   private JTextField getErgebniss(){
118     if(Ergebniss == null) {
119       Ergebniss = new JTextField();
120       Ergebniss.setSize(150, 20);
121       Ergebniss.setLocation(220, 300);
122     }
123     return Ergebniss;
124   }
125
126   private JButton getJButton() {
127     JButton jButton = new JButton();
128     jButton = new javax.swing.JButton();
129     jButton.setText("Ergebnis");
130     jButton.setSize(100, 30);
131     jButton.setLocation(250,250);
132     jButton.addActionListener(new ActionListener(){
133       public void actionPerformed (ActionEvent event){
134       {try{
135         TaschenrechService myService = new TaschenrechServiceLocator();
136         Taschenrech myPort = myService.getTaschenrech();
137         wert1 = Double.parseDouble(getEingabe1().getText());
138         wert2 = Double.parseDouble(getEingabe2().getText));
139         if(getcombo().getSelectedItem().equals("Addition")){
140           erg = myPort.add(wert1, wert2);
141           getErgebniss().setText(Double.toString(erg));
142         }
143         if(getcombo().getSelectedItem().equals("Subtraktion")){
144           erg = myPort.sub(wert1, wert2);
145           getErgebniss().setText(Double.toString(erg));
146         }
147         if(getcombo().getSelectedItem().equals("Multiplikation")){
148           erg = myPort.mult(wert1, wert2);
149           getErgebniss().setText(Double.toString(erg));
150         }
151         if(getcombo().getSelectedItem().equals("Division")){
152           erg = myPort.div(wert1, wert2);
153           getErgebniss().setText(Double.toString(erg));
154         }
155         if(getcombo().getSelectedItem().equals("Prozent")){
156           erg = myPort.prozent(wert1, wert2);
157           getErgebniss().setText(Double.toString(erg));
158         }
159         if(getcombo().getSelectedItem().equals("Wurzel")){
160           erg = myPort.wurzel(wert1);
161           getErgebniss().setText(Double.toString(erg));
162         }
163       }catch(Exception er){};
164       }}});
165     return jButton;
166   }
167
168 public static void main(String[] args) {
169     Oberflaeche app = new Oberflaeche();
170     app.show();
171   }
172
173 }
```

Quelltext B.1: Taschenrechner: Oberfläche.java

B.2. Kreuzfahrtliste

B.2.1. Listing 1: Die Java-Klasse: RouteList

```
1  package de.bizweb.uebung;
2
3  /*
4   * Created on 10.10.2004
5   *
6   * Diese Datei ist Teil der Lösung einer Übungsaufgabe im Bizweb-Projekt
7   * an der Hochschule Bremerhaven (WS 2004/2005).
8   *
9   */
10
11 import java.sql.Connection;
12 import java.sql.DriverManager;
13 import java.sql.PreparedStatement;
14 import java.sql.ResultSet;
15 import java.sql.SQLException;
16 import java.util.Vector;
17
18 /**
19  * Module:  RouteList.java
20  * Purpose: Defines the Provider-Class for the WebService
21  *              ResultList
22  *
23  * @author Marco Junge
24  *
25  * Das Programm liefert eine Liste von Schiffsrouten eines Schiffes
26  * zurück. Die Daten werden dazu aus den Tabellen einer MySQL-
27  * Datenbank ausgelesen.
28  *
29  */
30 public class RouteList {
31     private ListElement[] m_elemList;
32
33     /**
34      * Methode, die die Daten aus der MySQL-Datenbank ausliest und
35      * als ListElement[] zurückliefert.
36      *
37      * @param paramShip: Das Schiff, zu der die Abfrage erfolgen
38      *                   soll.
39      * @return ListElement[]: Der Rückgabetyp ist ein Array der
40      *                        Bean-Klasse ListElement.
41      */
42     public ListElement[] getList(String paramShip)
43       throws NoSuchShipException{
44         try {
45            Class.forName("com.mysql.jdbc.Driver");
46            Connection con =
47                 DriverManager.getConnection
48                    ("jdbc:mysql://localhost/bizweb",
49                     "root",
50                     "");
51            String query =
52               "SELECT r.route_id,s.shipping_company,s.ship_name,"
53             + "p.port_name, p2.port_name FROM routes r JOIN ships s "
54             + "ON r.ship_id=s.ship_id JOIN ports p ON "
55             + "r.route_from=p.port_id "
56             + "JOIN ports p2 ON r.route_to=p2.port_id WHERE "
57             + "s.ship_name = ?";
58            PreparedStatement pStmt = con.prepareStatement(query);
59
60            pStmt.setString(1,paramShip);
```

```
61              ResultSet rs = pStmt.executeQuery();
62
63            //Zunächst wird das ResultSet in einem Vector gespeichert...
64              Vector resVec = new Vector();
65
66              while (rs.next()){
67                  ListElement elem = new ListElement(
68                          rs.getInt(1),
69                          rs.getString(2),
70                          rs.getString(3),
71                          rs.getString(4),
72                          rs.getString(5));
73                  resVec.addElement(elem);
74              }
75
76              rs.close();
77              pStmt.close();
78              con.close();
79
80             //...wenn keine Abfrageergebnisse, dann eine Exception
81              if (resVec.size() == 0){
82                 throw new NoSuchShipException(
83                    "Keine Ergebnisse zu dieser Eingabe");
84              }
85             //...sonst wird es wieder in ein ListElement[] umgewandelt
86              m_elemList = new ListElement[resVec.size()];
87
88              for(int i=0; i<m_elemList.length; i++){
89                  m_elemList[i] = (ListElement)resVec.elementAt(i);
90              }
91
92              return m_elemList; //und zurückgeliefert.
93
94          } catch (SQLException e){
95               System.out.println("--- SQLException eingetreten ---\n");
96               do{
97                 System.out.println("Fehlermeldung: " + e.getMessage());
98                 System.out.println("XOPEN-Fehlercode: "
99                                    + e.getSQLState());
100         } while ( (e = e.getNextException()) != null );
101         } catch (ClassNotFoundException e){
102             System.out.println("--- Fehler bei der "
103                                + "Treiberregistrierung ---\n");
104         } finally { }
105
106         //um NullPointerExceptions zu verhindern:
107         m_elemList = new ListElement[0];
108         return m_elemList;
109
110     }
111 }
```

Quelltext B.2: Kreuzfahrtliste: RouteList.java

B.2.2. Listing 2: Die Java-Klasse: ListElement

```
1 package de.bizweb.uebung;
2
3 /*
4  * Created on 10.10.2004
5  *
6  * Diese Datei ist Teil der Lösung einer Übungsaufgabe im Bizweb-
```

```
7   * Projekt an der Hochschule Bremerhaven (WS 2004/2005).
8   *
9   */
10
11  /**
12   * Module:  ListElement.java
13   * Purpose: Defines the Bean-Class to Map the WebService ResultList
14   *
15   * @author Marco Junge
16   *
17   * In dieser Bean-Klasse ist die Datenstruktur ListElement
18   * implementiert. ListElement beinhaltet die Informationen zu
19   * Schiffsrouten eines Schiffes.
20   *
21   */
22  public class ListElement {
23      private int m_routeID;
24      private String m_shipName;
25      private String m_shippingCompany;
26      private String m_portFrom;
27      private String m_portTo;
28
29
30      /**
31       * Konstruktor zum Setzen der Membervariablen.
32       *
33       * @param routeid:  Routennummer
34       * @param name:   Schiffsname
35       * @param company:  Reederei
36       * @param from:   Starthafen
37       * @param to:     Zielhafen
38       */
39      public ListElement(int routeid,
40                         String name,
41                         String company,
42                         String from,
43                         String to) {
44          m_routeID = routeid;
45          m_shipName = name;
46          m_shippingCompany = company;
47          m_portFrom = from;
48          m_portTo = to;
49      }
50
51
52      /**
53       * @return Returns the m_portFrom.
54       */
55      public String getPortFrom() {
56          return m_portFrom;
57      }
58      /**
59       * @param from The m_portFrom to set.
60       */
61      public void setPortFrom(String from) {
62          m_portFrom = from;
63      }
64      /**
65       * @return Returns the m_portTo.
66       */
67      public String getPortTo() {
68          return m_portTo;
69      }
70      /**
71       * @param to The m_portTo to set.
72       */
```

```
73      public void setPortTo(String to) {
74          m_portTo = to;
75      }
76      /**
77       * @return Returns the m_routeID.
78       */
79      public int getRouteID() {
80          return m_routeID;
81      }
82      /**
83       * @param m_routeid The m_routeID to set.
84       */
85      public void setRouteID(int m_routeid) {
86          m_routeID = m_routeid;
87      }
88      /**
89       * @return Returns the m_shipName.
90       */
91      public String getShipName() {
92          return m_shipName;
93      }
94      /**
95       * @param name The m_shipName to set.
96       */
97      public void setShipName(String name) {
98          m_shipName = name;
99      }
100     /**
101      * @return Returns the m_shippingCompany.
102      */
103     public String getShippingCompany() {
104         return m_shippingCompany;
105     }
106     /**
107      * @param company The m_shippingCompany to set.
108      */
109     public void setShippingCompany(String company) {
110         m_shippingCompany = company;
111     }
112 }
```

Quelltext B.3: Kreuzfahrtliste: ListElement.java

B.2.3. Listing 3: Die Java-Klasse: NoSuchShipException

```
1  package de.bizweb.uebung;
2
3  /*
4   * Created on 27.10.2004
5   *
6   * Diese Datei ist Teil der Lösung einer Übungsaufgabe im Bizweb-
7   * Projekt an der Hochschule Bremerhaven (WS 2004/2005).
8   *
9   */
10
11 import java.rmi.RemoteException;
12
13 /**
14  * Module:  NoSuchShipException.java
15  * Purpose: Defines an Exception-Class instead of NULL-Results
16  *
17  * @author Marco Junge
```

```
18  *
19  * Diese Exception-Klasse dient zum Abfangen von NULL-Ergebnissen
20  * bei der Datenbankabfrage.
21  *
22  */
23 public class NoSuchShipException extends RemoteException {
24     public NoSuchShipException(String message){
25         super(message);
26     }
27 }
```

Quelltext B.4: Kreuzfahrtliste: NoSuchShipException.java

B.2.4. Listing 4: Die Java-Klasse: Start

```
1  package de.bizweb.uebung.application;
2
3  /*
4   * Created on 10.10.2004
5   *
6   * Diese Datei ist Teil der Lösung einer Übungsaufgabe im Bizweb-
7   * Projekt an der Hochschule Bremerhaven (WS 2004/2005).
8   *
9   */
10
11 import javax.swing.JFrame;
12
13 import javax.swing.JTable;
14 import javax.swing.JScrollPane;
15 import javax.swing.JLabel;
16 import javax.swing.JPanel;
17 import javax.swing.JButton;
18 import javax.swing.JTextField;
19 import org.apache.axis.AxisFault;
20
21 /**
22  * Package: de.bizweb.uebung.application
23  * Module:  Start.java
24  * Purpose: Defines the GUI for a Route-List for ships
25  *
26  * @author Marco Junge
27  *
28  * Programm zur Ausgabe von Routeninformationen zu einem Schiff in
29  * einer JTable. Die MySQL-Daten werden über einen WebService
30  * ermittelt.
31  *
32  **/
33 public class Start extends JFrame {
34
35   private javax.swing.JPanel jContentPane = null;
36   private JPanel jPanel = null;
37   private JLabel jLabel = null;
38   private JLabel message = null;
39   private JTextField jTextField = null;
40   private JButton jButton = null;
41
42   private JScrollPane jScrollPane = null;
43   private JTable jTable = null;
44
45   private TableControlModel tableControlModel = new TableControlModel();
46
47
```

```
48  /**
49   * This method initializes jTable
50   *
51   * @return javax.swing.JTable
52   */
53  private JTable getJTable() {
54    if (jTable == null) {
55      jTable = new JTable();
56    }
57    return jTable;
58  }
59  /**
60   * This method initializes jScrollPane
61   *
62   * @return javax.swing.JScrollPane
63   */
64  private JScrollPane getJScrollPane() {
65    if (jScrollPane == null) {
66      jScrollPane = new JScrollPane();
67      jScrollPane.setViewportView(getJTable());
68    }
69    return jScrollPane;
70  }
71  /**
72   * This method initializes jLabel
73   *
74   * @return javax.swing.JLabel
75   */
76  private JLabel getJLabel() {
77    if (jLabel == null) {
78      jLabel = new JLabel();
79      jLabel.setText("Schiffsname");
80    }
81    return jLabel;
82  }
83
84  private JLabel getMessage() {
85    if (message == null) {
86      message = new JLabel();
87    }
88    return message;
89  }
90  /**
91   * This method initializes jButton
92   *
93   * @return javax.swing.JButton
94   */
95  private JButton getJButton() {
96    if (jButton == null) {
97      jButton = new JButton();
98      jButton.setText("Refresh");
99      jButton.addActionListener(
100             new java.awt.event.ActionListener() {
101       public void actionPerformed(
102                   java.awt.event.ActionEvent e) {
103         try {
104             tableControlModel =
105                 new TableControlModel(
106                     tableControlModel.getListData(
107                             getJTextField().getText()));
108
109             getJTable().setModel(tableControlModel);
110           getJTable().updateUI();
111           message.setText("Verbindung zur Datenbank
112                                           erfolgreich!");
113         } catch (AxisFault af) {
```

```
114                message.setText(af.getMessage());
115
116              } catch (Exception err) {
117                message.setText("Es ist ein Fehler
118                                                  aufgetreten!");
119                err.printStackTrace();
120              }
121
122            }
123          });
124        }
125        return jButton;
126      }
127      /**
128       * This method initializes jTextField
129       *
130       * @return javax.swing.JTextField
131       */
132      private JTextField getJTextField() {
133        if (jTextField == null) {
134          jTextField = new JTextField();
135          jTextField.setColumns(15);
136        }
137        return jTextField;
138      }
139        /**
140       * This method initializes jPanel
141       *
142       * @return javax.swing.JPanel
143       */
144          public static void main(String[] args) {
145            new Start();
146      }
147      /**
148       * This is the default constructor
149       */
150      public Start() {
151        super();
152        initialize();
153      }
154      /**
155       * This method initializes this
156       *
157       * @return void
158       */
159      private void initialize() {
160        this.setTitle("Routen-Liste eines Schiffes...");
161        this.setSize(600,400);
162        this.setContentPane(getJContentPane());
163        this.addWindowListener(new java.awt.event.WindowAdapter() {
164          public void windowClosing(java.awt.event.WindowEvent e) {
165            System.exit(0);
166          }
167        });
168        this.setVisible(true);
169      }
170      /**
171       * This method initializes jContentPane
172       *
173       * @return javax.swing.JPanel
174       */
175      private javax.swing.JPanel getJContentPane() {
176        if(jContentPane == null) {
177          JPanel query = new JPanel();
178          query.add(getJLabel());
179          query.add(getJTextField());
```

```
180         query.add(getJButton());
181         jContentPane = new javax.swing.JPanel();
182         jContentPane.setLayout(new java.awt.BorderLayout());
183         jContentPane.add(query, java.awt.BorderLayout.NORTH);
184         jContentPane.add(getJScrollPane(),
185                                 java.awt.BorderLayout.CENTER);
186         jContentPane.add(getMessage(),
187                                 java.awt.BorderLayout.SOUTH);
188       }
189       return jContentPane;
190     }
191 }
```

Quelltext B.5: Kreuzfahrtliste: Start.java

B.2.5. Listing 5: Die Java-Klasse TableControlModel

```
1  /*
2   * Created on 10.10.2004
3   *
4   * Diese Datei ist Teil der Lösung einer Übungsaufgabe im
5   * Bizweb-Projekt an der Hochschule Bremerhaven (WS 2004/2005).
6   *
7   */
8
9  package de.bizweb.uebung.application;
10 import javax.swing.table.AbstractTableModel;
11 import localhost.axis.services.RouteList.RouteList;
12 import localhost.axis.services.RouteList.RouteListService;
13 import localhost.axis.services.RouteList.RouteListServiceLocator;
14 import de.bizweb.uebung.beans.ListElement;
15
16 /**
17  * Module:  TableControlModel.java
18  * Extends: javax.swing.table.AbstractTableModel
19  * Purpose: Defines the TableModel for a JTable
20  *
21  * @author Marco Junge
22  *
23  * Hilfsklasse, mit der die Informationen aus der Datenbankabfrage für
24  * die Ausgabe in einer JTable aufbereitet werden.
25  *
26  **/
27 public class TableControlModel extends AbstractTableModel{
28   ListElement[] resultList;
29   String columnNames[] = new String[0];
30   Object data[][] = new Object[0][0];
31
32   /**
33    * Konstruktor für TableControlModel
34    *
35    * @param paramData: Daten, die ausgegeben werden sollen.
36    */
37   public TableControlModel(ListElement[] paramData){
38       columnNames =
39         new String[]{"RouteID","Ship","Company","Start","Destination"};
40
41       Object[][] setData = new Object[paramData.length][5];
42
43       for (int i=0;i<paramData.length;i++){
44           setData[i][0] = (Object)(Integer.toString(
45                                       paramData[i].getRouteID()));
```

```
46            setData[i][1] = (Object)paramData[i].getShipName();
47            setData[i][2] = (Object)paramData[i].getShippingCompany();
48            setData[i][3] = (Object)paramData[i].getPortFrom();
49            setData[i][4] = (Object)paramData[i].getPortTo();
50        }
51
52
53        data = setData;
54
55
56    }
57
58    public TableControlModel(){ }
59
60    /**
61     * Methode, die die Daten zu einem Selektionskriterium zurückliefert.
62     *
63     * @param paramShip: Schiffsname, zu dem selektiert werden soll.
64     * @return resultList: Ergebnis der Selektion
65     * @throws Exception
66     */
67    public ListElement[] getListData(String paramShip) throws Exception{
68        RouteListService myService = new RouteListServiceLocator();
69        RouteList myPort = myService.getRouteList();
70
71        resultList = myPort.getList(paramShip);
72        return resultList;
73    }
74
75    /**
76     * (Überladene) Funktion, die die max. Spaltenanzahl zurückliefert.
77     *
78     * @return int, max. Anzahl der Spalten des TableModels
79     */
80    public int getColumnCount(){
81      return data[0].length;
82    }
83
84    /**
85     * (Überladene) Funktion, die die max. Zeilenanzahl zurückliefert.
86     *
87     * @return int, max. Anzahl der Zeilen des TableModels
88     */
89    public int getRowCount(){
90      return data.length;
91    }
92
93    /**
94     * (Überladene) Funktion, die ein bestimmtes Objekt des TableModels
95     * zurückliefert.
96     *
97     * @param int row, Zeile
98     * @param int col, Spalte
99     * @return Object
100    */
101
102   public Object getValueAt(int row, int col){
103     return data[row][col];
104   }
105
106   /**
107    * (Überladene) Funktion, die eine Spaltenüberschrift des TableModels
108    * zurückliefert.
109    *
110    * @param int i, Iterator der Spalte
111    * @return String, Spaltenüberschrift
```

```
112     */
113   public String getColumnName(int i){
114     return columnNames[i];
115   }
116 
117   /**
118    * (Überladene) Funktion, mit der bestimmt werden kann, ob eine Zelle
119    * editierbar sein soll.
120    *
121    * Normierung:
122    *  Alle Spalten sollen nicht-editierbar sein.
123    *
124    * @param int row, Zeile
125    * @param int col, Spalte
126    * @return boolean
127    */
128   public boolean isCellEditable(int row, int col){
129     return false;
130   }
131 }
```

Quelltext B.6: Kreuzfahrtliste: TableControlModel.java

C. Projektschritte

C.1. Tabellen

C.1.1. ZBW_Kreditkarten

Transparente Tabelle ZBW_KREDITKARTEN aktiv

Kurzbeschreibung Kreditkarten

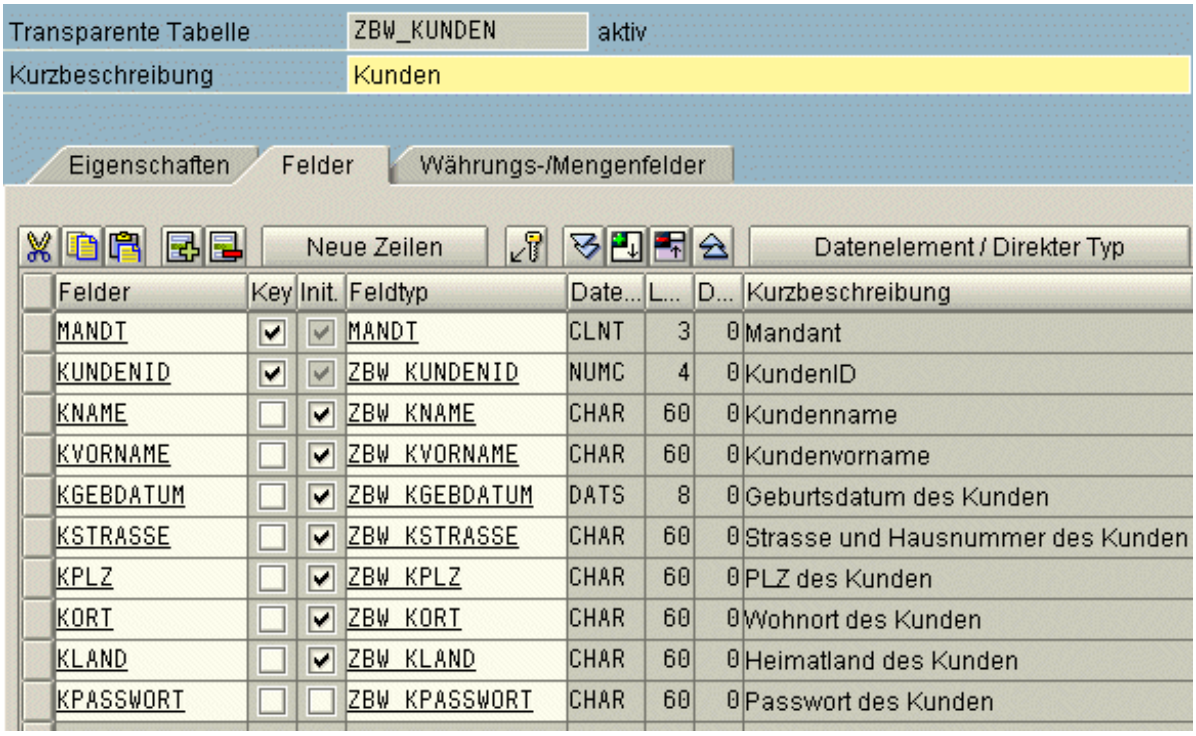
Eigenschaften | Felder | Währungs-/Mengenfelder

Neue Zeilen | Datenelement / Direkter Typ

Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	☑	☑	MANDT	CLNT	3	0	Mandant
KKID	☑	☑	ZBW_KKID	NUMC	16	0	KreditkartenID
KKPRUEFNUMMER	☐	☑	ZBW_KKPRUEFNUMER	NUMC	3	0	Kreditkarten Pruefnummer
KUNDENID	☐	☑	ZBW_KUNDENID	NUMC	4	0	KundenID
KKHERAUSGEBER	☐	☑	ZBW_KKHERAUSGEBER	CHAR	30	0	Kreditkaren Herausgeber
KKGUELTIGKEIT	☐	☑	ZBW_KKGUELTIGKEIT	DATS	8	0	Kreditkarten Gueltigkeit

Abbildung C.1.: Datenbank: Tabelle ZBW_Kreditkarten

C.1.2. ZBW_Kunden



Transparente Tabelle: ZBW_KUNDEN aktiv
Kurzbeschreibung: Kunden

Eigenschaften | Felder | Währungs-/Mengenfelder

Neue Zeilen | Datenelement / Direkter Typ

Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	✓	✓	MANDT	CLNT	3	0	Mandant
KUNDENID	✓	✓	ZBW_KUNDENID	NUMC	4	0	KundenID
KNAME		✓	ZBW_KNAME	CHAR	60	0	Kundenname
KVORNAME		✓	ZBW_KVORNAME	CHAR	60	0	Kundenvorname
KGEBDATUM		✓	ZBW_KGEBDATUM	DATS	8	0	Geburtsdatum des Kunden
KSTRASSE		✓	ZBW_KSTRASSE	CHAR	60	0	Strasse und Hausnummer des Kunden
KPLZ		✓	ZBW_KPLZ	CHAR	60	0	PLZ des Kunden
KORT		✓	ZBW_KORT	CHAR	60	0	Wohnort des Kunden
KLAND		✓	ZBW_KLAND	CHAR	60	0	Heimatland des Kunden
KPASSWORT			ZBW_KPASSWORT	CHAR	60	0	Passwort des Kunden

Abbildung C.2.: Datenbank: Tabelle ZBW_Kunden

C.1.3. ZBW_Schiffe

Transparente Tabelle: ZBW_SCHIFFE aktiv
Kurzbeschreibung: Schiffe

Eigenschaften | Felder | Währungs-/Mengenfelder

Neue Zeilen | Datenelement /

Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	✓	✓	MANDT	CLNT	3	0	Mandant
SCHIFFSID	✓	✓	ZBW_SCHIFFSID	NUMC	4	0	SchiffsID
SCHIFFSNAME		✓	ZBW_SCHIFFSNAME	CHAR	60	0	Schiffsname

Abbildung C.3.: Datenbank: Tabelle ZBW_Schiffe

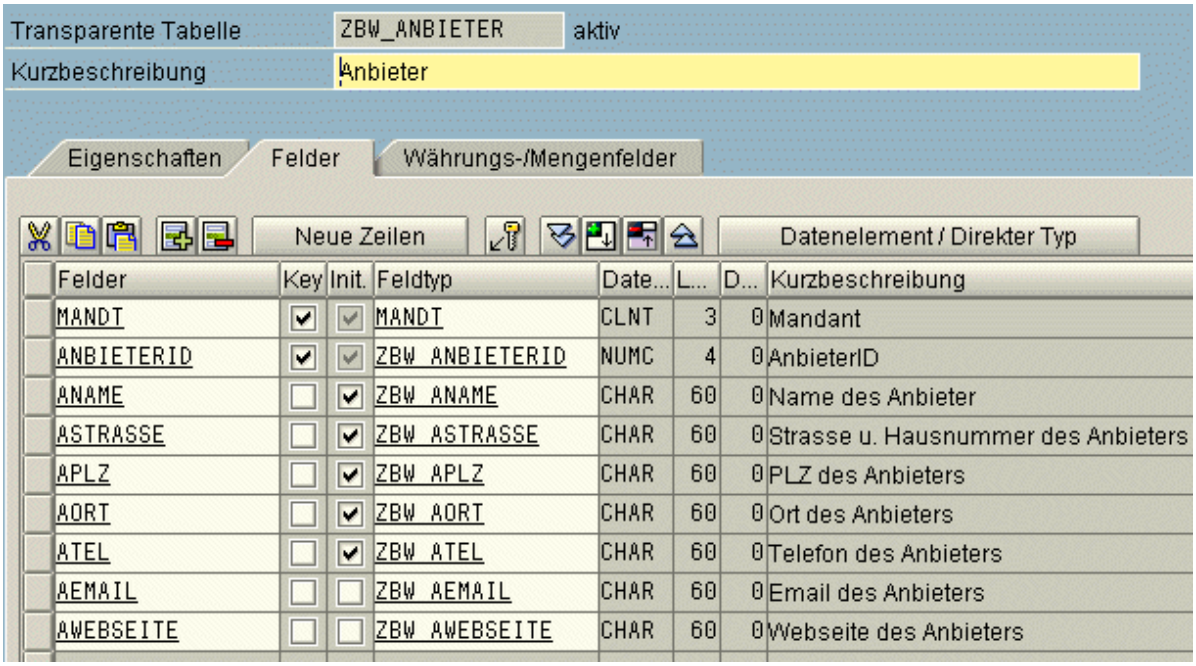
C.1.4. ZBW_Kabinentypen

Abbildung C.4.: Datenbank: Tabelle ZBW_Kabinentypen

C.1.5. ZBW_SKabinen

Abbildung C.5.: Datenbank: Tabelle ZBW_SKabinen

C.1.6. ZBW_Anbieter



Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	☑	☑	MANDT	CLNT	3	0	Mandant
ANBIETERID	☑	☑	ZBW_ANBIETERID	NUMC	4	0	AnbieterID
ANAME	☐	☑	ZBW_ANAME	CHAR	60	0	Name des Anbieter
ASTRASSE	☐	☑	ZBW_ASTRASSE	CHAR	60	0	Strasse u. Hausnummer des Anbieters
APLZ	☐	☑	ZBW_APLZ	CHAR	60	0	PLZ des Anbieters
AORT	☐	☑	ZBW_AORT	CHAR	60	0	Ort des Anbieters
ATEL	☐	☑	ZBW_ATEL	CHAR	60	0	Telefon des Anbieters
AEMAIL	☐	☐	ZBW_AEMAIL	CHAR	60	0	Email des Anbieters
AWEBSEITE	☐	☐	ZBW_AWEBSEITE	CHAR	60	0	Webseite des Anbieters

Abbildung C.6.: Datenbank: Tabelle ZBW_Anbieter

C.1.7. ZBW_Regionen

Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	☑	☑	MANDT	CLNT	3	0	Mandant
REGIONID	☑	☑	ZBW_REGIONID	NUMC	4	0	RegionID
REGIONNAME	☐	☑	ZBW_REGIONNAME	CHAR	60	0	Name der Region

Abbildung C.7.: Datenbank: Tabelle ZBW_Regionen

C.1.8. ZBW_Reisen

Transparente Tabelle ZBW_REISEN aktiv
Kurzbeschreibung Reisen

Eigenschaften | Felder | Währungs-/Mengenfelder

Neue Zeilen | Datenelement

Felder	Key	Init.	Feldtyp	Date...	L...	D...	Kurzbeschreibung
MANDT	☑	☑	MANDT	CLNT	3	0	Mandant
REISEID	☑	☑	ZBW_REISEID	NUMC	4	0	ReiseID
ANBIETERID	☐	☑	ZBW_ANBIETERID	NUMC	4	0	AnbieterID
REGIONID	☐	☑	ZBW_REGIONID	NUMC	4	0	RegionID
SCHIFFSID	☐	☑	ZBW_SCHIFFSID	NUMC	4	0	SchiffsID
REISEABFAHRT	☐	☑	ZBW_REISEABFAHRT	DATS	8	0	Beginn der Reise
REISEANKUNFT	☐	☑	ZBW_REISEANKUNFT	DATS	8	0	Ende der Reise
BASISHAFEN	☐	☑	ZBW_BASISHAFEN	CHAR	60	0	Basishafen
ZIELHAFEN	☐	☑	ZBW_ZIELHAFEN	CHAR	60	0	Zielhafen

Abbildung C.8.: Datenbank: Tabelle ZBW_Reisen

C.1.9. ZBW_Buchung

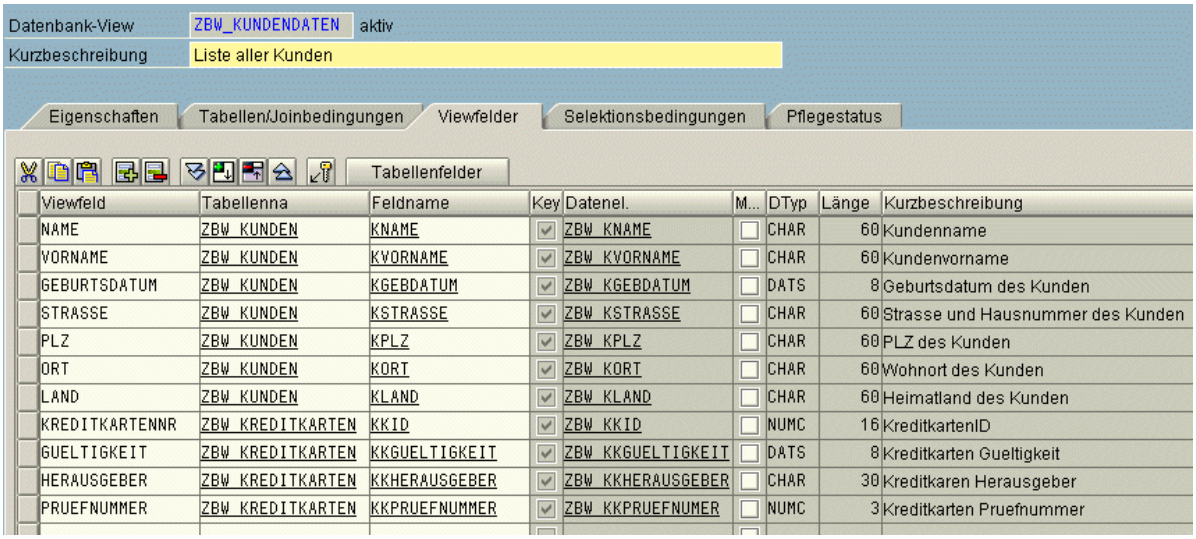
Abbildung C.9.: Datenbank: Tabelle ZBW_Buchung

C.1.10. Auflistung der Tabellen in SE80

Abbildung C.10.: Datenbank: Auflistung der Tabellen in SE80

C.2. Views

C.2.1. ZBW_Kundendaten



Datenbank-View ZBW_KUNDENDATEN aktiv
Kurzbeschreibung Liste aller Kunden

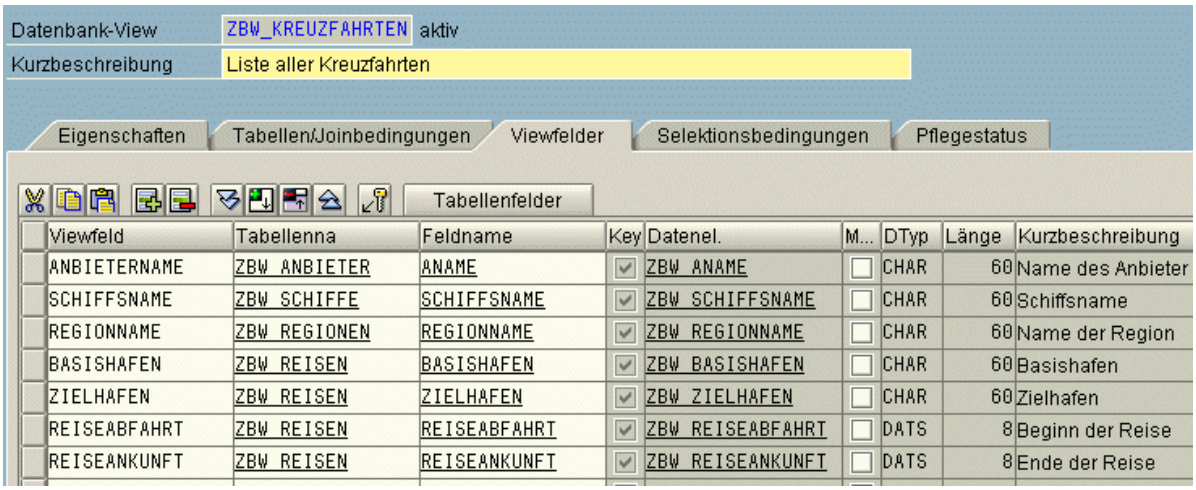
Eigenschaften | Tabellen/Joinbedingungen | Viewfelder | Selektionsbedingungen | Pflegestatus

Tabellenfelder

Viewfeld	Tabellenna	Feldname	Key	Datenel.	M...	DTyp	Länge	Kurzbeschreibung
NAME	ZBW_KUNDEN	KNAME	☑	ZBW_KNAME	☐	CHAR	60	Kundenname
VORNAME	ZBW_KUNDEN	KVORNAME	☑	ZBW_KVORNAME	☐	CHAR	60	Kundenvorname
GEBURTSDATUM	ZBW_KUNDEN	KGEBDATUM	☑	ZBW_KGEBDATUM	☐	DATS	8	Geburtsdatum des Kunden
STRASSE	ZBW_KUNDEN	KSTRASSE	☑	ZBW_KSTRASSE	☐	CHAR	60	Strasse und Hausnummer des Kunden
PLZ	ZBW_KUNDEN	KPLZ	☑	ZBW_KPLZ	☐	CHAR	60	PLZ des Kunden
ORT	ZBW_KUNDEN	KORT	☑	ZBW_KORT	☐	CHAR	60	Wohnort des Kunden
LAND	ZBW_KUNDEN	KLAND	☑	ZBW_KLAND	☐	CHAR	60	Heimatland des Kunden
KREDITKARTENNR	ZBW_KREDITKARTEN	KKID	☑	ZBW_KKID	☐	NUMC	16	KreditkartenID
GUELTIGKEIT	ZBW_KREDITKARTEN	KKGUELTIGKEIT	☑	ZBW_KKGUELTIGKEIT	☐	DATS	8	Kreditkarten Gueltigkeit
HERAUSGEBER	ZBW_KREDITKARTEN	KKHERAUSGEBER	☑	ZBW_KKHERAUSGEBER	☐	CHAR	30	Kreditkaren Herausgeber
PRUEFNUMMER	ZBW_KREDITKARTEN	KKPRUEFNUMMER	☑	ZBW_KKPRUEFNUMER	☐	NUMC	3	Kreditkarten Pruefnummer

Abbildung C.11.: Datenbank: View ZBW_Kundendaten

C.2.2. ZBW_Kreuzfahrten



Datenbank-View ZBW_KREUZFAHRTEN aktiv
Kurzbeschreibung Liste aller Kreuzfahrten

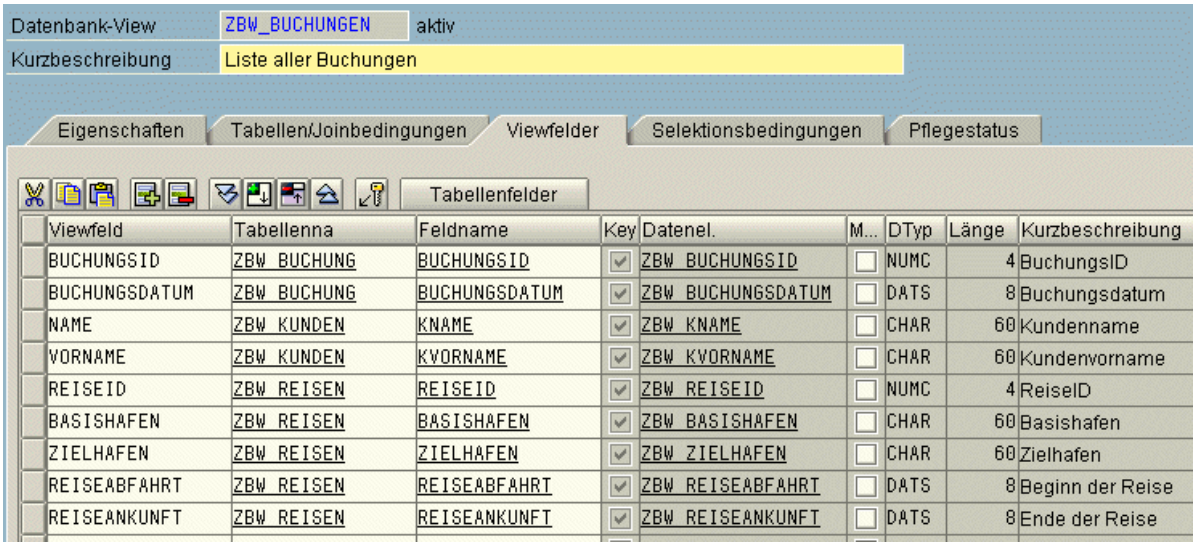
Eigenschaften | Tabellen/Joinbedingungen | Viewfelder | Selektionsbedingungen | Pflegestatus

Tabellenfelder

Viewfeld	Tabellenna	Feldname	Key	Datenel.	M...	DTyp	Länge	Kurzbeschreibung
ANBIETERNAME	ZBW_ANBIETER	ANAME	☑	ZBW_ANAME	☐	CHAR	60	Name des Anbieter
SCHIFFSNAME	ZBW_SCHIFFE	SCHIFFSNAME	☑	ZBW_SCHIFFSNAME	☐	CHAR	60	Schiffsname
REGIONNAME	ZBW_REGIONEN	REGIONNAME	☑	ZBW_REGIONNAME	☐	CHAR	60	Name der Region
BASISHAFEN	ZBW_REISEN	BASISHAFEN	☑	ZBW_BASISHAFEN	☐	CHAR	60	Basishafen
ZIELHAFEN	ZBW_REISEN	ZIELHAFEN	☑	ZBW_ZIELHAFEN	☐	CHAR	60	Zielhafen
REISEABFAHRT	ZBW_REISEN	REISEABFAHRT	☑	ZBW_REISEABFAHRT	☐	DATS	8	Beginn der Reise
REISEANKUNFT	ZBW_REISEN	REISEANKUNFT	☑	ZBW_REISEANKUNFT	☐	DATS	8	Ende der Reise

Abbildung C.12.: Datenbank: View ZBW_Kreuzfahrten

C.2.3. ZBW_Buchungen



Viewfeld	Tabellenna	Feldname	Key	Datenel.	M...	DTyp	Länge	Kurzbeschreibung
BUCHUNGSID	ZBW_BUCHUNG	BUCHUNGSID	☑	ZBW_BUCHUNGSID	☐	NUMC	4	BuchungsID
BUCHUNGSDATUM	ZBW_BUCHUNG	BUCHUNGSDATUM	☑	ZBW_BUCHUNGSDATUM	☐	DATS	8	Buchungsdatum
NAME	ZBW_KUNDEN	KNAME	☑	ZBW_KNAME	☐	CHAR	60	Kundenname
VORNAME	ZBW_KUNDEN	KVORNAME	☑	ZBW_KVORNAME	☐	CHAR	60	Kundenvorname
REISEID	ZBW_REISEN	REISEID	☑	ZBW_REISEID	☐	NUMC	4	ReiseID
BASISHAFEN	ZBW_REISEN	BASISHAFEN	☑	ZBW_BASISHAFEN	☐	CHAR	60	Basishafen
ZIELHAFEN	ZBW_REISEN	ZIELHAFEN	☑	ZBW_ZIELHAFEN	☐	CHAR	60	Zielhafen
REISEABFAHRT	ZBW_REISEN	REISEABFAHRT	☑	ZBW_REISEABFAHRT	☐	DATS	8	Beginn der Reise
REISEANKUNFT	ZBW_REISEN	REISEANKUNFT	☑	ZBW_REISEANKUNFT	☐	DATS	8	Ende der Reise

Abbildung C.13.: Datenbank: View ZBW_Buchungen

C.3. Datenelemente

Kurzbeschreibung:	KreditkartenID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID	
Feldbezeichner		
kurz	4	KKID
mittel	14	KreditkartenID
lang	14	KreditkartenID
Überschrift	14	KreditkartenID

Tabelle C.1.: Datenbank: Datenelement #1: ZBW_KKID

Kurzbeschreibung:	KundenID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID	
Feldbezeichner		
kurz	4	KKID
mittel	8	KundenID
lang	8	KundenID
Überschrift	8	KundenID

Tabelle C.2.: Datenbank: Datenelement #2: ZBW_KundenID

Kurzbeschreibung:	Kreditkarten Herausgeber	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_KKHerausgeber	
	Feldbezeichner	
kurz	8	Herausg.
mittel	11	Herausgeber
lang	24	Kreditkarten-Herausgeber
Überschrift	24	Kreditkarten-Herausgeber

Tabelle C.3.: Datenbank: Datenelement #3: ZBW_KKHerausgeber

Kurzbeschreibung:	Kreditkarten Prüfnummer	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_Pruefnummer_DOM	
	Feldbezeichner	
kurz	8	Pruefnr.
mittel	11	Pruefnummer
lang	24	Kreditkarten-Pruefnummer
Überschrift	24	Kreditkarten-Pruefnummer

Tabelle C.4.: Datenbank: Datenelement #4: ZBW_KKPruefnummer

Kurzbeschreibung:	Kreditkarten Gültigkeit	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_DATUM_DOM	
	Feldbezeichner	
kurz	7	Gueltig
mittel	11	Gueltigkeit
lang	24	Kreditkarten-Gueltigkeit
Überschrift	24	Kreditkarten-Gueltigkeit

Tabelle C.5.: Datenbank: Datenelement #5: ZBW_KKGueltigkeit

Kurzbeschreibung:	Kreditkarten RegionID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
	Feldbezeichner	
kurz	5	RegID
mittel	8	RegionID
lang	8	RegionID
Überschrift	8	RegionID

Tabelle C.6.: Datenbank: Datenelement #6: ZBW_RegionID

Kurzbeschreibung:	Name der Region	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	6	Region
mittel	6	Region
lang	10	Regionname
Überschrift	10	Regionname

Tabelle C.7.: Datenbank: Datenelement #7: ZBW_RegionName

Kurzbeschreibung:	SchiffsID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
	Feldbezeichner	
kurz	3	SID
mittel	9	SchiffsID
lang	9	SchiffsID
Überschrift	9	SchiffsID

Tabelle C.8.: Datenbank: Datenelement #8: ZBW_SchiffsID

Kurzbeschreibung:	SchiffsName	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	6	Schiff
mittel	11	SchiffsName
lang	11	SchiffsName
Überschrift	11	SchiffsName

Tabelle C.9.: Datenbank: Datenelement #9: ZBW_SchiffsName

Kurzbeschreibung:	BuchungsID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
	Feldbezeichner	
kurz	10	BuchungsID
mittel	10	BuchungsID
lang	10	BuchungsID
Überschrift	10	BuchungsID

Tabelle C.10.: Datenbank: Datenelement #10: ZBW_BuchungsID

Kurzbeschreibung:	ReiseID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
Feldbezeichner		
kurz	7	ReiseID
mittel	7	ReiseID
lang	7	ReiseID
Überschrift	7	ReiseID

Tabelle C.11.: Datenbank: Datenelement #11: ZBW_ReiseID

Kurzbeschreibung:	BuchungsDatum	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
Feldbezeichner		
kurz	6	BDatum
mittel	13	Buchungsdatum
lang	13	Buchungsdatum
Überschrift	13	Buchungsdatum

Tabelle C.12.: Datenbank: Datenelement #12: ZBW_BuchungsDatum

Kurzbeschreibung:	KabinenID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
Feldbezeichner		
kurz	9	KabinenID
mittel	9	KabinenID
lang	9	KabinenID
Überschrift	9	KabinenID

Tabelle C.13.: Datenbank: Datenelement #13: ZBW_KabinenID

Kurzbeschreibung:	KabinenPreis	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
Feldbezeichner		
kurz	5	Preis
mittel	13	Kabinen-Preis
lang	13	Kabinen-Preis
Überschrift	13	Kabinen-Preis

Tabelle C.14.: Datenbank: Datenelement #14: ZBW_KabinenPreis

Kurzbeschreibung:	Geburtsdatum des Kunden	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_DATUM_DOM	
	Feldbezeichner	
kurz	8	GebDatum
mittel	12	Geburtsdatum
lang	12	Geburtsdatum
Überschrift	12	Geburtsdatum

Tabelle C.15.: Datenbank: Datenelement #15: ZBW_KGebDatum

Kurzbeschreibung:	Kundenname	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	5	Name
mittel	13	Kundenname
lang	13	Kundenname
Überschrift	13	Kundenname

Tabelle C.16.: Datenbank: Datenelement #16: ZBW_KName

Kurzbeschreibung:	Kundenvorname	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	7	Vorname
mittel	13	Kundenvorname
lang	13	Kundenvorname
Überschrift	13	Kundenvorname

Tabelle C.17.: Datenbank: Datenelement #17: ZBW_KVorname

Kurzbeschreibung:	Strasse und Hausnummer des Kunden	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	7	Strasse
mittel	14	Strasse u. Nr.
lang	21	Strasse u. Hausnummer
Überschrift	21	Strasse u. Hausnummer

Tabelle C.18.: Datenbank: Datenelement #18: ZBW_KStrasse

Kurzbeschreibung:	PLZ des Kunden	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	3	PLZ
mittel	3	PLZ
lang	10	Kunden_PLZ
Überschrift	10	Kunden_PLZ

Tabelle C.19.: Datenbank: Datenelement #19: ZBW_KPLZ

Kurzbeschreibung:	Wohnort des Kunden	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	3	Ort
mittel	7	Wohnort
lang	10	Kunden_Ort
Überschrift	10	Kunden_Ort

Tabelle C.20.: Datenbank: Datenelement #20: ZBW_KORT

Kurzbeschreibung:	Heimatland des Kunden	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	4	Land
mittel	4	Land
lang	4	Land
Überschrift	4	Land

Tabelle C.21.: Datenbank: Datenelement #21: ZBW_KLAND

Kurzbeschreibung:	Passwort	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	2	PW
mittel	8	Passwort
lang	8	Passwort
Überschrift	8	Passwort

Tabelle C.22.: Datenbank: Datenelement #22: ZBW_KPasswort

Kurzbeschreibung:	AnbieterID	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ID_DOM	
Feldbezeichner		
kurz	10	AnbieterID
mittel	10	AnbieterID
lang	10	AnbieterID
Überschrift	10	AnbieterID

Tabelle C.23.: Datenbank: Datenelement #23: ZBW_AnbieterID

Kurzbeschreibung:	Beginn der Reise	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_DATUM_DOM	
Feldbezeichner		
kurz	7	Abfahrt
mittel	12	Reiseabfahrt
lang	12	Reiseabfahrt
Überschrift	12	Reiseabfahrt

Tabelle C.24.: Datenbank: Datenelement #24: ZBW_ReiseAbfahrt

Kurzbeschreibung:	Ende der Reise	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_DATUM_DOM	
Feldbezeichner		
kurz	7	Ankunft
mittel	13	Reiseankunft
lang	13	Reiseankunft
Überschrift	13	Reiseankunft

Tabelle C.25.: Datenbank: Datenelement #25: ZBW_ReiseAnkunft

Kurzbeschreibung:	Basishafen	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
Feldbezeichner		
kurz	5	Basis
mittel	10	Basishafen
lang	10	Basishafen
Überschrift	10	Basishafen

Tabelle C.26.: Datenbank: Datenelement #26: ZBW_BasisHafen

Kurzbeschreibung:	ZielHafen	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	4	Ziel
mittel	9	Zielhafen
lang	9	Zielhafen
Überschrift	9	Zielhafen

Tabelle C.27.: Datenbank: Datenelement #27: ZBW_ZielHafen

Kurzbeschreibung:	Name des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_NAME_DOM	
	Feldbezeichner	
kurz	8	Anbieter
mittel	12	Anbietername
lang	12	Anbietername
Überschrift	12	Anbietername

Tabelle C.28.: Datenbank: Datenelement #28: ZBW_AName

Kurzbeschreibung:	Strasse u. Hausnummer des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_STRASSE_DOM	
	Feldbezeichner	
kurz	7	Strasse
mittel	16	Anbieter_Strasse
lang	16	Anbieter_Strasse
Überschrift	30	Anbieter_Strasse u. Hausnummer

Tabelle C.29.: Datenbank: Datenelement #29: ZBW_AStrasse

Kurzbeschreibung:	PLZ des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_PLZ_DOM	
	Feldbezeichner	
kurz	3	PLZ
mittel	12	Anbieter_PLZ
lang	12	Anbieter_PLZ
Überschrift	12	Anbieter_PLZ

Tabelle C.30.: Datenbank: Datenelement #30: ZBW_APLZ

Kurzbeschreibung:	Ort des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ORT_DOM	
	Feldbezeichner	
kurz	3	PLZ
mittel	12	Anbieter_Ort
lang	12	Anbieter_Ort
Überschrift	12	Anbieter_Ort

Tabelle C.31.: Datenbank: Datenelement #31: ZBW_AOrt

Kurzbeschreibung:	Telefon des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_TEL_DOM	
	Feldbezeichner	
kurz	4	PLZ
mittel	16	Anbieter_Telefon
lang	16	Anbieter_Telefon
Überschrift	16	Anbieter_Telefon

Tabelle C.32.: Datenbank: Datenelement #32: ZBW_ATel

Kurzbeschreibung:	Email des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_EMAIL_DOM	
	Feldbezeichner	
kurz	5	Email
mittel	15	Anbieter_E-mail
lang	15	Anbieter_E-mail
Überschrift	15	Anbieter_E-mail

Tabelle C.33.: Datenbank: Datenelement #33: ZBW_AEmail

Kurzbeschreibung:	Webseite des Anbieters	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_WEBSEITE_DOM	
	Feldbezeichner	
kurz	8	Webseite
mittel	17	Anbieter_Webseite
lang	17	Anbieter_Webseite
Überschrift	17	Anbieter_Webseite

Tabelle C.34.: Datenbank: Datenelement #34: ZBW_AWebseite

Kurzbeschreibung:	Kabinenanzahl	
Entwicklungsklasse:	ZB_BizWeb	
Domäne:	ZBW_ANZAHL_DOM	
	Feldbezeichner	
kurz	7	Kabinen
mittel	13	Kabinenanzahl
lang	13	Kabinenanzahl
Überschrift	13	Kabinenanzahl

Tabelle C.35.: Datenbank: Datenelement #35: ZBW_Kabinenanzahl

C.3.1. Auflistung der Datenelemente in SE80

Datenelemente	
ZBW_ABFAHRTSDATUM	Abfahrtsdatum
ZBW_AEMAIL	Email des Anbieters
ZBW_ANAME	Name des Anbieter
ZBW_ANBIETERID	AnbieterID
ZBW_ANKUNFTSDATUM	Ankunft im Zielhafen
ZBW_AORT	Ort des Anbieters
ZBW_APLZ	PLZ des Anbieters
ZBW_ASTRASSE	Strasse u. Hausnummer des Anbieters
ZBW_ATEL	Telefon des Anbieters
ZBW_AWEBSEITE	Webseite des Anbieters
ZBW_BASISHAFEN	Basishafen
ZBW_BUCHUNGSDATUM	Buchungsdatum
ZBW_BUCHUNGSID	BuchungsID
ZBW_KABINENANZAHL	Kabinenanzahl
ZBW_KABINENID	KABINENID
ZBW_KABINENNAME	Kabinenname
ZBW_KABINENPREIS	Kabinenpreis
ZBW_KGEBDATUM	Geburtsdatum des Kunden
ZBW_KKGUELTIGKEIT	Kreditkarten Gueltigkeit
ZBW_KKHERAUSGEBER	Kreditkaren Herausgeber
ZBW_KKID	KreditkartenID
ZBW_KKPRUEFNUMER	Kreditkarten Pruefnummer
ZBW_KLAND	Heimatland des Kunden
ZBW_KNAME	Kundenname
ZBW_KORT	Wohnort des Kunden
ZBW_KPASSWORT	Passwort des Kunden
ZBW_KPLZ	PLZ des Kunden
ZBW_KSTRASSE	Strasse und Hausnummer des Kunden
ZBW_KUNDENID	KundenID
ZBW_KVORNAME	Kundenvorname
ZBW_NAME	name
ZBW_REGIONID	RegionID
ZBW_REGIONNAME	Name der Region
ZBW_REISEABFAHRT	Beginn der Reise
ZBW_REISEANKUNFT	Ende der Reise
ZBW_REISEBEGINN	Beginn der Reise
ZBW_REISEID	ReiseID
ZBW_SCHIFFSID	SchiffsID
ZBW_SCHIFFSNAME	Schiffsname
ZBW_ZIELHAFEN	Zielhafen

Abbildung C.14.: Datenbank: Auflistung der Datenelemente in SE80

C.4. Domänen

Kurzbeschreibung:	StandardID				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		NUMC	4	0	4

Tabelle C.36.: Datenbank: Domäne #1: ZBW_ID_DOM

Kurzbeschreibung:	Kreditkarten Herausgeber				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		CHAR	30	0	30

Tabelle C.37.: Datenbank: Domäne #2: ZBW_KKHerausgeber_DOM

Kurzbeschreibung:	Kreditkarten Prüfnummer				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		NUMC	3	0	3

Tabelle C.38.: Datenbank: Domäne #3: ZBW_Pruefnummer_DOM

Kurzbeschreibung:	Datum				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		DATS	8	0	10

Tabelle C.39.: Datenbank: Domäne #4: ZBW_Datum_DOM

Kurzbeschreibung:	Namen				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		CHAR	60	0	60

Tabelle C.40.: Datenbank: Domäne #5: ZBW_Name_DOM

Kurzbeschreibung:	Preis				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		CURR	7	2	9

Tabelle C.41.: Datenbank: Domäne #6: ZBW_Preis_DOM

Kurzbeschreibung:	KreditkartenID				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		NUMC	16	0	16

Tabelle C.42.: Datenbank: Domäne #7: ZBW_KKID_DOM

Kurzbeschreibung:	Anzahl				
Entwicklungsklasse:	ZB_BizWeb				
Definition:		NUMC	4	0	4

Tabelle C.43.: Datenbank: Domäne #8: ZBW_ANZAHL_DOM

C.4.1. Auflistung der Domänen in SE80

Abbildung C.15.: Datenbank: Auflistung der Domänen in SE80

C.5. Inhalt der Tabellen

MANDT	ANBIETERID	ANAME	ASTRASSE	APLZ	AORT	ATEL	AEMAIL	AWEBSEITE
312	1	SEETOURS - GERMAN BRANCH OF CARNIVAL PLC	FRANKFURTER STRAßE 233	63263	NEU-ISENBURG	+49 (0) 6102 811-000	INFO@SEETOURS.DE	HTTP://WWW.SEETOURS.DE
312	2	CUNARD LINE	24303 TOWNE CENTER DRIVE	CA 91355	VALENCIA	+1 800.7.CUNARD	INFO@CUNARD.COM	HTTP://WWW.CUNARD.COM
312	3	PRINCESS CRUISE LTD.	25603 NEW ORLEANS STREET	CA 91215	SAN JOSE	+1 800.PRINCESS	INFO@PRINCESS.COM	HTTP://WWW.PRINCESS.COM

Tabelle C.44.: Datenbank: Inhalt der Tabelle: ZBW_Anbieter

MANDT	BUCHUNGSID	KUNDENID	REISEID	BUCHUNGSDATUM
312	1	12	11	27.10.2004
312	2	9	11	27.10.2004
312	3	1	1	27.10.2004
312	4	2	4	27.10.2004
312	5	3	2	27.10.2004
312	6	4	10	27.10.2004
312	7	5	3	28.10.2004
312	8	6	10	28.10.2004
312	9	7	11	28.10.2004
312	10	8	11	25.10.2004
312	11	10	12	26.10.2004
312	12	11	8	27.10.2004
312	13	13	5	28.10.2004
312	14	14	2	28.10.2004
312	15	15	9	28.10.2004
312	16	16	6	27.10.2004
312	17	1	2	20.01.2005
312	18	1	3	20.01.2005
312	19	1	4	20.01.2005
312	20	1	7	20.01.2005
312	21	7	36	27.01.2005
312	22	7	38	03.02.2005
312	23	8	1	16.02.2005
312	24	87	67	16.02.2005

Tabelle C.45.: Datenbank: Inhalt der Tabelle: ZBW_Buchung

MANDT	KABINENID	KABINENNAME	KABINENPREIS	WAEHRUNG
312	1	AUSSENKABINE	2.100,00	EURO
312	2	INNENKABINE	1.200,00	EURO

Tabelle C.46.: Datenbank: Inhalt der Tabelle: ZBW_Kabinentypen

MANDT	KKID	KKPRUEFNUMMER	KUNDENID	KKHERAUSGEBER	KKGUELTIGKEIT
312	3013246548765410	189	16	DINERS CLUB	03.09.2006
312	3445646546566450	189	15	AMERICAN EXPRESS	30.12.2005
312	3456565111223540	852	7	AMERICAN EXPRESS	30.04.2008
312	3478874158548780	123	4	AMERICAN EXPRESS	31.12.2006
312	3754693323699890	123	9	AMERICAN EXPRESS	31.12.2007
312	3845564654564580	622	3	DINERS CLUB	31.01.2008
312	4132465833211220	546	11	VISA	31.05.2006
312	4326356888875490	652	8	VISA	31.10.2007
312	4546545657985560	189	13	VISA	03.03.2007
312	5145685466221350	216	5	MASTER	18.10.2008
312	5211348731324540	329	16	MASTER	10.05.2009
312	5348888545345450	189	12	MASTER	01.02.2007
312	5443545645555430	189	13	MASTER	01.08.2007
312	6011232555982220	998	2	DISCOVER	31.12.2006
312	6011234552553220	532	1	DISCOVER	31.01.2007
312	6011243243452430	189	14	DISCOVER	01.06.2008
312	6011521564542310	555	6	DISCOVER	30.11.2008
312	6011526544443230	328	10	DISCOVER	30.11.2007

Tabelle C.47.: Datenbank: Inhalt der Tabelle: ZBW_Kreditkarten

MANDT	KUNDENID	KNAME	KVORNAME	KGEBDATUM	KSTRASSE	KPLZ	KORT	KLAND	KPASSWORT
312	1	BRAEUER	BENNY	03.06.1982	GATKENWEG 5	27635	MIDLEM	DEUTSCHLAND	BRAEUER
312	2	EGGERS	CHRISTIAN	08.12.1980	AN DER KARLSTADT 8	27568	BREMERHAVEN	DEUTSCHLAND	EGGERS
312	3	LEFKE	CHRISTIAN	03.09.1980	BREMERHAVENER STR. 99	25785	BREMERHAVEN	DEUTSCHLAND	LEFKE
312	4	SCHRADE	CHRISTOPHER	03.02.1981	ROGGENBRECHTSTR. 21	27579	BREMERHAVEN	DEUTSCHLAND	SCHRADE
312	5	BUHLMANN	DOMINIK	21.04.1979	DELMENSTR. 5	23471	DELMENHORST	DEUTSCHLAND	BUHLMANN
312	6	STELMASZEK	JAN	28.05.1980	GERHARDSTR. 93	28754	BREMEN	DEUTSCHLAND	STELMASZEK
312	7	ONKEN	BASTIAN	22.12.1979	BERGSTRASSE 23B	27613	STETEL	DEUTSCHLAND	ONKEN
312	8	JUNGE	MARCO	01.03.1978	AN DER KARLSTADT 16	27658	BREMERHAVEN	DEUTSCHLAND	JUNGE
312	9	SIEDEK	NICOLE	02.04.1980	BERTHOLD STRASSE 45	49082	OSNABRUECK	DEUTSCHLAND	SIEDEK
312	10	FULLE	REMO	08.02.1980	LUSTIGSTRASSE 45	28547	BREMERHAVEN	DEUTSCHLAND	FULLE
312	11	WAESCH	THOMAS	14.10.1981	STETTNER STR. 38	27617	BEVERSTADT	DEUTSCHLAND	WAESCH
312	12	SCHLEGEL	VIKTOR	25.05.1980	STEINGRUPPE 5	27589	BREMERHAVEN	DEUTSCHLAND	SCHLEGEL
312	13	SCHMIDT	ALFRED	01.05.1964	AN DER KARLSTADT 9	27568	BREMERHAVEN	DEUTSCHLAND	SCHMIDT
312	14	VIEFHUES	DIETER	06.07.1949	OSTENBERGER STR. 5	27746	WORPSHAUSEN	DEUTSCHLAND	VIEFHUES
312	15	SIMPSONS	HOMER	01.05.1993	SPRINGFIELDSTREET 5	1111	SPRINGFIELD	USA	BART
312	16	FLANDERS	NED	01.05.1965	SPRINGFIELDSTREET 4	1111	SPRINGFIELD	USA	FLANDERS

Tabelle C.48.: Datenbank: Inhalt der Tabelle: ZBW_Kunden

MANDT	REGIONID	REGIONNAME
312	1	BERMUDAS
312	2	MITTELAMERIKA
312	3	HAWAII
312	4	KANAREN
312	5	KARIBIK
312	6	MEXIKO
312	7	SÜD AMERIKA
312	8	WELTREISE
312	9	ALASKA
312	10	AFRIKA
312	11	EUROPA
312	12	MITTELMEER
312	13	ASIEN

Tabelle C.49.: Datenbank: Inhalt der Tabelle: ZBW_Regionen

MANDT	REISEID	ANBIETERID	REGIONID	SCHIFFSID	REISEABFAHRT	REISEANKUNFT	BASISHAFEN	ZIELHAFEN
312	1	1	12	3	05.11.2004	12.11.2004	PALMA DE MALLORCA	PALMA DE MALLORCA
312	2	1	12	3	09.11.2004	26.11.2004	PALMA DE MALLORCA	PALMA DE MALLORCA
312	3	1	12	3	03.12.2004	10.12.2004	PALMA DE MALLORCA	PALMA DE MALLORCA
312	4	2	2	20	16.04.2005	22.04.2005	SOUTHAMPTON	NEW YORK
312	5	2	2	20	28.04.2005	04.05.2005	SOUTHAMPTON	NEW YORK
312	6	2	2	20	22.05.2005	28.05.2005	SOUTHAMPTON	NEW YORK
312	7	3	5	10	18.12.2004	27.12.2004	FT. LAUDERDALE	FT. LAUDERDALE
312	8	3	5	10	26.12.2004	03.01.2005	FT. LAUDERDALE	FT. LAUDERDALE
312	9	1	7	18	24.11.2004	05.12.2005	RIO DE JANEIRO	BUENOS AIRES
312	10	1	13	18	09.03.2005	22.03.2005	MANILA	HONG KONG
312	11	1	8	5	22.12.2004	23.04.2005	MONTE CARLO	BREMERHAVEN
312	12	1	12	14	26.09.2005	08.10.2005	NIZZA	VENEDIG
312	13	2	12	4	14.05.2005	24.05.2005	MALAGA	ROM
312	14	1	3	13	17.09.2005	24.09.2005	LOS ANGELES	HONULULU
312	15	1	3	14	24.09.2005	01.10.2005	LOS ANGELES	HONULULU
312	16	2	12	13	15.07.2005	22.07.2005	MALAGA	ROM
312	17	3	12	13	01.07.2005	15.07.2005	BARCELONA	ATHEN
312	18	3	12	14	10.07.2005	17.07.2005	VALENCIA	INSTANBUL
312	19	3	12	14	12.08.2005	19.07.2005	VALENCIA	INSTANBUL
312	20	2	12	8	31.08.2005	11.09.2005	LISSABON	KAIRO
312	21	2	12	8	10.09.2005	24.09.2005	LISSABON	KAIRO
312	22	2	12	8	01.09.2005	14.09.2005	MARSEILLE	HERAKLION
312	23	2	12	8	06.07.2005	20.07.2005	MARSEILLE	HERAKLION
312	24	2	1	14	18.07.2005	08.08.2005	MIAMI	NASSAU
312	25	2	1	14	26.06.2005	03.07.2005	MIAMI	NASSAU
312	26	2	1	2	08.05.2005	15.05.2005	MIAMI	NASSAU
312	27	2	2	4	01.05.2005	08.05.2005	BLUEFIELDS	MARACALBO
312	28	2	1	3	24.03.2005	31.03.2005	FT. LAUDERDALE	NASSAU
312	29	2	2	4	06.05.2005	20.05.2005	MEXICO CITY	PANAMA
312	30	2	2	3	01.05.2005	08.05.2005	CARACAS	MONTEVIDEO
312	31	1	2	9	02.01.2005	16.01.2005	CAYENNE	LA PLATA
312	32	1	2	9	12.03.2005	26.03.2005	CAYENNE	LA PLATA
312	33	1	2	11	12.03.2005	26.03.2005	LIMA	VALPARAISO
312	34	1	2	11	31.03.2005	08.04.2005	LIMA	VALPARAISO
312	35	1	2	12	31.03.2005	08.04.2005	LIMA	BUENOS AIRES
312	36	1	2	12	04.07.2005	24.04.2005	LIMA	BUENOS AIRES
312	37	1	10	19	11.12.2005	18.12.2005	CAPE TOWN	KAIRO
312	38	1	4	15	05.08.2005	12.08.2005	ARRECIFE	VALVERDE
312	39	1	4	15	05.08.2005	12.08.2005	SANTA CRUZ DE LA PALMA	PUERTO DEL ROSARIO
312	40	1	4	16	26.06.2005	10.07.2005	SANTA CRUZ DE LA PALMA	PUERTO DEL ROSARIO
312	41	1	4	17	26.06.2005	10.07.2005	LAS PALMAS	PUERTO DE LA CRUZ
312	42	1	5	18	05.11.2005	12.11.2005	HAVANNA	SAN JUAN
312	43	1	5	19	02.12.2005	16.12.2005	SANTA DOMINGO	MIAMI
312	44	1	10	18	11.12.2005	18.12.2005	DAKAR	KAIRO
312	45	1	10	18	11.12.2005	18.12.2005	FREETOWN	TOAMASINA
312	46	1	10	17	11.11.2005	26.11.2005	FREETOWN	TOAMASINA
312	47	1	11	17	08.06.2005	22.06.2005	ST. PETERSBURG	LONDON
312	48	1	11	15	26.06.2005	03.07.2005	LIVERPOOL	RIVA
312	49	1	11	5	14.07.2005	28.07.2005	WARSCHAU	AMSTERDAM
312	50	1	11	5	05.08.2005	12.08.2005	PARIS	NANTES

Tabelle C.50.: Datenbank: Inhalt der Tabelle: ZBW_Reisen

MANDT	SCHIFFSID	SCHIFFSNAME
312	1	ADVENTURE OF THE SEAS
312	2	AIDAAURA
312	3	AIDACARA
312	4	AIDAVITA
312	5	ALBATROS
312	6	AROSA DONNA
312	7	AROSA MIA
312	8	CELEBRITY CRUISES SUMMIT
312	9	CELEBRITY CRUISES GALAXY
312	10	CARIBBEAN PRINCESS
312	11	COSTA EUROPA
312	12	COSTA ROMANTICA
312	13	MS BERLIN
312	14	MS BREMEN
312	15	ISLAND PRINCESS
312	16	MS DEUTSCHLAND
312	17	SUN PRINCESS
312	18	MS EUROPA
312	19	MS SWISS DIAMANT
312	20	QUEEN MARY 2

Tabelle C.51.: Datenbank: Inhalt der Tabelle: ZBW_Schiffe

MANDT	KABINENID	SCHIFFSID	KABINENANZAHL
312	1	1	939
312	1	2	391
312	1	3	391
312	1	4	420
312	1	5	380
312	1	6	100
312	1	7	100
312	1	8	780
312	1	9	639
312	1	10	928
312	1	11	495
312	1	12	428
312	1	13	158
312	1	14	82
312	1	15	692
312	1	16	224
312	1	17	603
312	1	18	204
312	1	19	75
312	1	20	1017
312	2	1	618
312	2	2	202
312	2	3	202
312	2	4	213
312	2	5	162
312	2	8	195
312	2	9	296
312	2	10	372
312	2	11	252
312	2	12	216
312	2	13	52
312	2	15	108
312	2	16	70
312	2	17	372
312	2	20	293

Tabelle C.52.: Datenbank: Inhalt der Tabelle: ZBW_Skabinen

C.6. JCo

C.6.1. convertData.java

```
1  /*
2   * Created on 23.11.2004
3   */
4  package de.bizweb.sap.jco;
5
6  //import java.text.DateFormat;
7  import java.text.ParseException;
8  import java.util.ArrayList;
9  import java.util.Date;
10 import java.util.GregorianCalendar;
11 import java.util.TimeZone;
12
13 import com.sap.mw.jco.util.SyncDateFormat;
14
15 /**
16  * Konvertiert die Inhalte des SAPReadData-Arrays anhand der ABAB-Datentypen
17  * in konkrete Java-Datentypen und liefert eine ArrayList zurück. Die ABAB-Datentypen
18  * werden mit Hilfe des SAPReadStructure-Arrays ermittelt.
19  *
20  * Konvertiert werden folgende ABAB-Datentypen: C, N, D, F und P.
21  *
22  * Gegenüberstellung der ABAB-, Java- und JCO-Datentypen:
23  *
24  *ABAB              Java    JCO
25  *b 1-Byte Integer         int     JCO.TYPE_INT1
26  *s 2-Byte Integer         int     JCO.TYPE_INT2
27  *l 4-Byte Integer         int     JCO.TYPE_INT
28  *C Zeichen           String    JCO.TYPE_CHAR
29  *N Numerisches Zeichen     String    JCO.TYPE_NUM
30  *P BCD (Binary Coded Decimal)  BigDecimal  JCO.TYPE_BCD
31  *D Datum             Date    JCO.TYPE_DATE
32  *T Zeit            Date    JCO.TYPE_TIME
33  *F Fließkommazahl        double    JCO.TYPE_FLOAT
34  *X Raw Data          byte[]    JCO.TYPE_BYTE
35  *g String (variable Länge)   String    JCO.TYPE_STRING
36  *y Raw Data (variable Länge)   byte[]    JCO.TYPE_XSTRING
37  *
38  *  @author Christian Eggers und Dominik Buhlmann
39  */
40 public class convertData
41 {
42     //~ Constructors -----------------------------------------------------------
43
44     public convertData() {}
45
46     //~ Methods ----------------------------------------------------------------
47
48     public static ArrayList getSAPTable(SAPReadData[] data, SAPReadStructure[] struct)
49     {
50         ArrayList liste = new ArrayList();
51         String s, str, t;
52         int c, start;
53
54         // Wandelt die Strings aus dem array
55         for(int i = 0; i < data.length; i++) {
56             s =  data[i].getWa();
57
58             start = 0;
59             str = null;
60             t = null;
```

```
61              c=0;
62
63              // den String auseinandernehmen, mit der Schleife drüber laufen
64              for(int j = 0; j < s.length(); j++) {
65                  if((s.charAt(j) == ';') || (j == s.length()-1)) {
66                      // Ist Zeilenende oder Seperator ";" erreicht?
67                      if((s.charAt(j) == ';'))
68                      {
69                          str = s.substring(start,j).trim();
70                        t = struct[c].getType();
71                        c++;
72                      }
73                      else
74                      {
75                          str = s.substring(start,j+1).trim();
76                          t = struct[c].getType();
77                      }
78
79                  if(t.equals("C")||(t.equals("N")&& str.length()>9))
80                  {
81                      liste.add(str);
82                      start = j+1;
83                  }
84                  else if (t.equals("N")&& str.length()<10)
85                  {
86                      liste.add(string2int(str));
87                    start = j+1;
88                  }
89                  else if (t.equals("D"))
90                  {
91                      liste.add(string2date(str));
92                      start = j+1;
93                  }
94                  else if (t.equals("F"))
95                  {
96                      liste.add(string2double(str));
97                      start = j+1;
98                  }
99                  else if (t.equals("P"))
100                 {
101                     liste.add(string2float(str));
102                     start = j+1;
103                 }
104               }
105             }
106         }
107         return liste;
108     }
109
110   public static Integer string2int(String input)
111   {
112       int change = Integer.parseInt(input);
113       Integer output = new Integer(change);
114       return output;
115   }
116
117   public static Double string2double(String input)
118   {
119       double change = Double.parseDouble(input);
120       Double output = new Double(change);
121
122       return output;
123   }
124
125   public static Float string2float(String input)
126   {
```

```
127         float change = Float.parseFloat(input);
128         Float output = new Float(change);
129 
130         return output;
131     }
132 
133     /**
134      * Der String input muss folgendes Format haben:
135      * yyyyMMdd
136      * @return Date
137      */
138     public static Date string2date(String input) {
139 
140         Date output = null;
141         GregorianCalendar cal = null;
142 
143         SyncDateFormat format = new SyncDateFormat( "yyyyMMdd" );
144 
145           try
146           {
147               output = format.parse(input);
148               cal = new GregorianCalendar( TimeZone.getTimeZone("ECT") );
149               cal.setTime(output);
150           }
151           catch (ParseException e)
152           {
153               e.printStackTrace();
154           }
155 
156           /*
157            *  Umwandlung von Kalender in String, z.B.:
158            *
159            *  DateFormat formater2 = DateFormat.getDateInstance(DateFormat.MEDIUM);
160            *  System.out.println( formater2.format(cal.getTime()));
161            *
162            */
163 
164 //          return cal;
165           return output;
166     }
167 
168     public GregorianCalendar date2calendar(Date date) {
169       GregorianCalendar gregorianCalendar =
170       new GregorianCalendar(TimeZone.getTimeZone("ECT"));
171 
172       gregorianCalendar.setTime(date);
173 
174       return gregorianCalendar;
175     }
176 }
```

Quelltext C.1: JCo: convertData.java

C.6.2. RFCReadTable.java

```
1 /*
2  * Created on 02.12.2004
3  */
4 
5 /**
6  * @author Christian Eggers und Dominik Buhlmann
7  */
```

```
8
9  package de.bizweb.sap.jco;
10 import java.text.SimpleDateFormat;
11 import java.util.ArrayList;
12 import java.util.Date;
13
14 public class RFCReadTable {
15   SAPLogon    logon     = null;
16   SAPReadData[] tableRows   = null;
17   SAPReadStructure[] tableFields  = null;
18   private final String CRUISETABLE  = "ZBW_KREUZFAHRTEN";
19
20   /* Konstruktor */
21   public RFCReadTable() {
22     /* Anmeldungsdaten */
23     logon = new SAPLogon("312",
24                          "test",
25                 "alfred",
26                 "D",
27                 "r3psap.hs-bremerhaven.de",
28                 "00");
29     /* Anmeldung */
30     logon.connect();
31   }
32
33   /**
34    *
35    * @return tableRows
36    */
37   private SAPReadData[] getRows(String[] options, String[] fields) {
38     try {
39             tableRows = SAPReadData.getList(logon.getConnection(),
40                     logon.getRepository(),
41                 CRUISETABLE,
42                     ";",                // delimiter
43                 "0",                // rowskips
44                     "0",                // rowcount
45                 options,
46                 fields);
47         } catch (Exception e) {
48             e.printStackTrace();
49         }
50
51         return tableRows;
52   }
53
54   /**
55    *
56    * @return tableFields
57    */
58   private SAPReadStructure[] getFields(String[] fields) {
59     try {
60             tableFields = SAPReadStructure.getList(logon.getConnection(),
61                     logon.getRepository(),
62                 CRUISETABLE,    // table
63                     ";",                // delimiter
64                 "0",                // rowskips
65                     "0",                // rowcount
66                 "",                 // options
67                 fields);
68         } catch (Exception e)
69         {
70             e.printStackTrace();
71         }
72
73     return tableFields;
```

```
74    }
75
76    /**
77     *
78     * @param tabelle
79     * @return liste
80     */
81    public ArrayList getSAPList() {
82
83        String[] opts = {};
84        String[] flds = {};
85        ArrayList liste = convertData.getSAPTable(getRows(opts,flds),
86                                  getFields(flds));
87        /* Abmeldung */
88        logon.disconnect();
89
90        return liste;
91    }
92
93    /**
94     *
95     * @param tabelle
96     * @param schiffsname
97     * @return liste
98     */
99    public ArrayList getSAPList(String schiffsname) {
100
101       String[] opts = {"SCHIFFSNAME = '"+schiffsname.trim().toUpperCase()+"'"};
102       String[] flds = {};
103       ArrayList liste = convertData.getSAPTable(getRows(opts,flds),
104                                 getFields(flds));
105       /* Abmeldung */
106       logon.disconnect();
107
108       return liste;
109   }
110
111   /**
112    *
113    * @param tabelle
114    * @param schiffsname
115    * @param regionname
116    * @return liste
117    */
118   public ArrayList getSAPList(String schiffsname,
119                 String regionname) {
120
121       String[] opts = {"SCHIFFSNAME = '"+schiffsname.trim().toUpperCase()+"'",
122                    "AND REGIONNAME = '"+regionname.trim().toUpperCase()+"'"};
123       String[] flds = {};
124       ArrayList liste = convertData.getSAPTable(getRows(opts,flds),
125                                 getFields(flds));
126       /* Abmeldung */
127       logon.disconnect();
128
129       return liste;
130   }
131
132   /**
133    *
134    * @param tabelle
135    * @param schiffsname
136    * @param basishafen
137    * @param reiseabfahrt
138    * @return liste
139    */
```

```
140   public ArrayList getSAPList(String schiffsname,
141                 String basishafen,
142                 Date reiseabfahrt) {
143
144       SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
145       String abfahrt = df.format(reiseabfahrt);
146
147       String[] opts = {"SCHIFFSNAME = '"+schiffsname.trim().toUpperCase()+"'",
148                   "AND BASISHAFEN = '"+basishafen.trim().toUpperCase()+"'",
149                   "AND REISEABFAHRT = '"+abfahrt+"'"};
150       String[] flds = {};
151       ArrayList liste = convertData.getSAPTable(getRows(opts,flds),
152                           getFields(flds));
153       /* Abmeldung */
154       logon.disconnect();
155
156       return liste;
157   }
158
159   /**
160    *
161    * @param tabelle
162    * @param anbieter
163    * @param basishafen
164    * @param zielhafen
165    * @param reiseabfahrt
166    * @return liste
167    */
168   public ArrayList getSAPList(String anbieter,
169                 String basishafen,
170                 String zielhafen,
171                 Date reiseabfahrt) {
172
173       SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
174       String abfahrt = df.format(reiseabfahrt);
175
176       String[] opts = {"ANBIETERNAME = '"+anbieter.trim().toUpperCase()+"'",
177                   "AND BASISHAFEN = '"+basishafen.trim().toUpperCase()+"'",
178                   "AND ZIELHAFEN = '"+zielhafen.trim().toUpperCase()+"'",
179                   "AND REISEABFAHRT = '"+abfahrt+"'"};
180       String[] flds = {};
181       ArrayList liste = convertData.getSAPTable(getRows(opts,flds),
182                           getFields(flds));
183       /* Abmeldung */
184       logon.disconnect();
185
186       return liste;
187   }
188
189   /**
190    *
191    * @param kundenID
192    * @param reiseID
193    * @return bid
194    */
195   public int bookingSAP(int kundenID,
196                 int reiseID){
197
198       String k = String.valueOf(kundenID);
199       String r = String.valueOf(reiseID);
200
201       int bid=-1;
202
203       try
204         {
205             bid = SAPCreateBooking.book(logon.getConnection(),
```

```
206                                 logon.getRepository(),
207                                 k, r);
208           }
209         catch (Exception e)
210           {
211               e.printStackTrace();
212           }
213         /* Abmeldung */
214         logon.disconnect();
215
216       return bid;
217     }
218 }
```

Quelltext C.2: JCo: RFCReadTable.java

C.6.3. SAPCreateBooking.java

```
1  package de.bizweb.sap.jco;
2
3  import com.sap.mw.jco.*;
4
5  public class SAPCreateBooking extends Object
6  {
7      public static int book(JCO.Client connection,
8                             IRepository repository,
9                             String kid,
10                 String rid)
11                            throws Exception
12     {
13         JCO.Function function =
14         repository.getFunctionTemplate("ZBW_BOOKING").getFunction();
15         function.getImportParameterList().setValue(kid, "KID");
16         function.getImportParameterList().setValue(rid, "RID");
17
18         JCO.Field bid = function.getExportParameterList().getField("BID");
19
20         connection.execute(function);
21
22         return bid.getInt();
23     }
24 }
```

Quelltext C.3: JCo: SAPCreateBooking.java

C.6.4. SAPLogon.java

```
1  package de.bizweb.sap.jco;
2
3  import com.sap.mw.jco.*;
4
5  public class SAPLogon extends Object
6  {
7      //~ Static fields/initializers ---------------------------------------------
8
9      private static final String SID = "Pool";
10
11     //~ Instance fields --------------------------------------------------------
```

```
12
13      private JCO.Client     mConnection;
14      private JCO.Repository mRepository;
15      private String clnt = null;
16      private String user = null;
17      private String pass = null;
18      private String lang = null;
19      private String host = null;
20      private String sysn = null;
21
22      //~ Constructors ------------------------------------------------------------
23
24      public SAPLogon(String clnt,String user,String pass,String lang,
25                      String host,String sysn)
26      {
27          this.clnt     = clnt;
28          this.user     = user;
29          this.pass     = pass;
30          this.lang     = lang;
31          this.host     = host;
32          this.sysn     = sysn;
33      }
34
35      //~ Methods ------------------------------------------------------------------
36
37      public JCO.Client getConnection()
38      {
39          return mConnection;
40      }
41
42      public JCO.Repository getRepository()
43      {
44          return mRepository;
45      }
46
47      // Verbindung aufbauen, Platz im Pool "reservieren"
48      public void connect()
49      {
50          if(JCO.getClientPoolManager().getPool(SID) == null)
51              JCO.addClientPool(SID, // Anmeldungspool
52                                10, // max. Anzahl der Verbindungen
53                                clnt,user,pass,lang,host,sysn);
54
55          /*
56           * Im Repository befinden sich alle Metainformation zu den
57           * Remote-fähigen Funktionsbausteinen, die genutzt
58           * werden können
59           */
60          mRepository     = new JCO.Repository("AS314",SID);
61          mConnection     = JCO.getClient(SID,true);
62      }
63
64      public void disconnect()
65      {
66          // Platz im Pool "freigeben"
67          JCO.releaseClient(mConnection);
68      }
69  }
```

Quelltext C.4: JCo: SAPLogon.java

C.6.5. SAPReadData.java

```
1  package de.bizweb.sap.jco;
2
3  import com.sap.mw.jco.*;
4
5  public class SAPReadData extends Object
6  {
7      //~ Instance fields --------------------------------------------------------
8
9      private String wa;
10
11     //~ Constructors -----------------------------------------------------------
12
13     public SAPReadData(String wa)
14     {
15         super();
16         this.wa = wa;
17     }
18
19     //~ Methods ----------------------------------------------------------------
20
21     /**
22      * @return Returns string wa
23      */
24     public String getWa()
25     {
26         return wa;
27     }
28
29     public static SAPReadData[] getList(JCO.Client connection,
30                                         IRepository repository, String table,
31                                         String delimiter, String rowskips,
32                                         String rowcount, String[] options,
33                                         String[] fields)
34                                  throws Exception
35     {
36
37
38         JCO.Function function =
39             repository.getFunctionTemplate("ZBW_READ_TABLE").getFunction();
40
41         function.getImportParameterList().setValue(table.toUpperCase(),
42                                                    "QUERY_TABLE");
43
44         //falls kein Begrenzer gesetzt wurde, als Default das Komma nehmen
45         if(delimiter.equals(null) || delimiter.equals(""))
46             delimiter = ",";
47
48         function.getImportParameterList().setValue(delimiter,"DELIMITER");
49         function.getImportParameterList().setValue(rowskips,"ROWSKIPS");
50         function.getImportParameterList().setValue(rowcount,"ROWCOUNT");
51
52         JCO.Table options_table =
53           function.getTableParameterList().getTable("OPTIONS");
54         for(int i=0; i < options.length; i++) {
55           options_table.appendRow();
56           options_table.setValue(options[i].toUpperCase(),"TEXT");
57         }
58
59         JCO.Table fields_table =
60           function.getTableParameterList().getTable("FIELDS");
61         for(int i=0; i < fields.length; i++) {
62           fields_table.appendRow();
63           fields_table.setValue(fields[i].toUpperCase(),"FIELDNAME");
64         }
65
```

```
66          connection.execute(function);
67
68          JCO.Table rows = function.getTableParameterList().getTable("DATA");
69          SAPReadData[] tableRows = new SAPReadData[rows.getNumRows()];
70
71          for(int i = 0; i < rows.getNumRows(); i++) {
72              rows.setRow(i);
73              tableRows[i] = new SAPReadData(rows.getString("WA"));
74          }
75
76          return tableRows;
77      }
78  }
```

Quelltext C.5: JCo: SAPReadData.java

C.6.6. SAPReadStructure.java

```
1   package de.bizweb.sap.jco;
2
3   import com.sap.mw.jco.*;
4
5   public class SAPReadStructure extends Object {
6       private String fieldname;
7       private String offset;
8       private String length;
9       private String type;
10      private String fieldtext;
11
12      public SAPReadStructure(String fieldname, String offset, String length,
13          String type, String fieldtext) {
14          super();
15          this.fieldname = fieldname;
16          this.offset = offset;
17          this.length = length;
18          this.type = type;
19          this.fieldtext = fieldtext;
20      }
21
22      public String getFieldname() {
23          return fieldname;
24      }
25
26      public String getOffset() {
27          return offset;
28      }
29
30      public String getType() {
31          return type;
32      }
33
34      public String getFieldtext() {
35          return fieldtext;
36      }
37
38      public String getLength() {
39          return length;
40      }
41
42      public static SAPReadStructure[] getList(JCO.Client connection,
43          IRepository repository, String table, String delimiter,
44          String rowskips, String rowcount, String options, String[] fields)
```

```
45          throws Exception {
46          JCO.Function function = repository.getFunctionTemplate("ZBW_READ_TABLE")
47                                              .getFunction();
48          function.getImportParameterList().setValue(table.toUpperCase(),
49              "QUERY_TABLE");
50
51          function.getImportParameterList().setValue('X', "NO_DATA");
52
53          JCO.Table fields_table =
54              function.getTableParameterList().getTable("FIELDS");
55            for(int i=0; i < fields.length; i++) {
56              fields_table.appendRow();
57              fields_table.setValue(fields[i].toUpperCase(),"FIELDNAME");
58            }
59
60          connection.execute(function);
61
62          JCO.Table rows = function.getTableParameterList().getTable("FIELDS");
63
64          SAPReadStructure[] tableRows = new SAPReadStructure[rows.getNumRows()];
65
66          for (int i = 0; i < rows.getNumRows(); i++) {
67              rows.setRow(i);
68
69              tableRows[i] = new SAPReadStructure(rows.getString("FIELDNAME"),
70                      rows.getString("OFFSET"), rows.getString("LENGTH"),
71                      rows.getString("TYPE"), rows.getString("FIELDTEXT"));
72
73              }
74
75          return tableRows;
76      }
77  }
```

Quelltext C.6: JCo: SAPReadStructure.java

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis