Universitatea Transilvania, Braşov

Facultatea de Matematică - Informatică

Catedra de Informatică

Lucrare de diplomă

Cuprins

Prefata

Bine aţi venit în lumea Serviciilor Web! Bine aţi venit în lumea în care programarea distribuită, interoperabilitatea, independenţa de platformă şi limbaj de programare îşi spune cuvântul, îşi cere drepturile. Sper ca această lucrare să vă ajute să găsiţi reperele de bază şi să navigaţi cu plăcere printre aceste idei si concepte. Şi, de ce nu, chiar să vă aduceţi propria contribuţie la dezvoltarea unei ramuri a informaticii care a luat naştere de curând şi se află într-o continuă expansiune.

Există şi funcţionează?

În ultimul timp tehnologia serviciilor web a apărut tot mai des în atenţia presei de specialitate. Un număr mare de produse legate de programarea internet a inceput să invadeze piaţa IT. Tot mai multe companii ce îşi desfăşoară activitatea cu ajutorul aplicaţiilor Internet îşi indreaptă atenţia către această ramură a tehnologiei. Cât despre adevăraţii lideri in domeniul producerii de software, aceştia au creat companii specializate in specificarea, dezvoltarea şi folosirea acestor noi protocoale şi tehnologii. Despre toate acestea vom vorbi mai încolo. Dar din aceste observatii putem deduce că acesta este un câmp de activitate nou, dinamic si plin de viitor.

Deşi de scurt timp în activitate, serviciile web au început să prindă contur. Iar pentru a continua dezvoltarea într-o manieră cât mai organizată, a fost necesară prezenţa unor standarde adoptate de comun acord de majoritatea producătorilor ce au un cuvant de spus în acest domeniu. Din această nevoie au luat naştere diverse standarde, în mare parte specificate de consortiul W3C, dintre care vom prezenta: eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Hypertext Transfer Protocol(HTTP), Web Service Description Language (WSDL) şi Universal Description Discovery and Integration (UDDI). Aceste standarde oferă regulile de bază ale serviciilor si vor fi prezentate în amănunt pe parcursul lucrării.

Pentru cei ce nu sunt familiarizaţi cu aceste denumiri, sa aruncăm o scurtă privire peste ele:

- XML oferă utilizatorului un mod independent de platformă de a încapsula si organiza datele. În acelaşi timp permite crearea elementelor de sintaxă pentru definirea unor noi vocabulare si a unor noi structuri de date.
- SOAP este un vocabular XML ce permite computerelor să interactioneze prin intermediul reţelei. Acest protocol de organizare şi încapsulare a informaţiei este următorul pas în domeniul Remote Procedure Call (RPC).

- HTTP este protocolul de comunicare cel mai des folosit de serviciile web pentru a transmite informaţiile. Este extrem de eficient datorită posibilităţii de a trece de firewall, în acelaşi timp permiţând utilizatorului să examineze datele transmise.
- WSDL este de asemenea un vocabular XML prin care computerul descrie în limbaj 'propriu' serviciile web pe care le expune, oferind programatorului posibilitatea de a genera cod de conectare în mod automat, prin intermediul unor tooluri disponibile pe piaţă (unele free, altele comerciale).
- UDDI a apărut datorită nevoii utilizatorilor de a găsi anumite servicii care să le îndeplinească cerinţele. Bazat pe XML, este un mod de publicare a serviciilor web.

Un alt argument ce ar putea fi decisiv pentru a studia acest domeniu îl reprezintă utilizarea tehnologiei SOAP de către Microsoft in noua platforma .NET. SOAP este protocolul pe care se bazează comunicarea dintre componente cât şi un suport puternic pentru dezvoltarea serviciilor distribuite.

Despre lucrare

După ce v-am prezentat toate aceste argumente extrase din lumea ce ne inconjoara (lumea giganţilor IT), ar fi timpul să spunem câteva cuvinte despre această lucrare. Ea a fost realizată din două motive: nevoia personală de a înţelege aceste noi tehnologii şi dorinţa de a prezenta cât mai amănunţit, concis si didactic cu putinţă cunoştinţele pe care le-am acumulat în acest domeniu.

Aici veti putea gasi informaţii despre naşterea si evoluţia internet în paralel cu nevoia de a folosi metode de programare cat mai puternice (OOP, programarea paralelă, programarea distribuită). După care urmatorul mare pas va fi explicarea cât mai concisă a fiecărui protocol mentionat mai sus, punând accentul în mod firesc asupra celor trei protocoale ce formează subiectul acestei lucrări: SOAP & WSDL & UDDI. Toate aceste informaţii vor fi completate cu exemple elocvente şi cu descrierea unor produse ce oferă programatorului posibilitatea de a implementa cât mai uşor şi rapid propriile servicii web. Ultima parte a lucrării de faţă este un studiu de caz al unei aplicaţii vaste, aplicatie ce are ca scop oferirea de servicii informaţionale printr-o interfaţă WEB.

Conţinutul lucrării

Lucrarea de faţă este organizată în trei părţi, fiecare cu un rol bine definit în prezentarea informaţiilor referitoare la serviciile Web.

Prima parte defineşte conceptul de serviciu Web, avantajele şi dezavantajele folosirii lor cât şi un scurt istoric. Apoi detaliază limbajul XML, oferind informaţii de bază despre structura unui document XML, despre elementele acestuia, reguli şi tehnologii de validare şi metode de procesare. Următorul pas îl reprezintă prezentarea protocolului SOAP, cunoştinţe de bază(cum ar fi structura unui mesaj, elementele de bază, modul de codare al datelor şamd) şi

avansate(folosirea ataşamentelor, a multireferenţierii, schimbarea protocoalelor de transport şamd). Apoi urmează descrierea unui serviciu Web folosind limbajul WSDL, după care ultimul capitol prezintă informaţii de bază despre protocolul UDDI.

Partea a doua prezintă, pe marginea unui exemplu, cum se poate crea şi publica un serviciu Web folosind serverul Apache AXIS. Pentru început tratează problema instalării utilitarelor necesare (serverul Tomcat şi serverul AXIS), după care continuă prin definirea unei probleme simple, urmată de paşii de bază necesari creării serviciului Web care să îndeplinească cerinţele problemei. Această secţiune a lucrării este încheiată de explicarea modului în care programatorul poate folosi bibliotecile AXIS pentru a implementa un client pentru serviciul Web.

Partea a treia reprezintă un studiu de caz al unui serviciu Web, proiect aflat în stadiul de dezvoltare. Acesta se numeşte PITA (Personal Intelligent Travel Assistant) şi are ca scop planificarea şi supervizarea călătoriilor. Fiind un proiect mare, lucrarea va prezenta conceptele de bază ce intervin în realizarea designului. Această secţiune poate fi folosită ca un bun exemplu despre modul în care trebuie iniţiat şi organizat un serviciu Web.

Cui i se adresează această lucrare?

Această lucrare se adresează în mod special celor ce doresc să înteleagă ceea ce află în spatele ideii de serviciu WEB. Deşi extrem de vast, am încercat să modularizez acest subiect astfel încât să fie uşor de parcurs (de la un capăt la altul) cât şi usor de accesat în cazul căutării unor informaţii legate de un produs anume. Datorită dorinţei de a împărtăşi propriile cunoştinţe, această lucrare are şi un caracter didactic, oferind o parcurgere gradată a subiectului, pornind de la lucruri de bază, idei fundamentale şi ajungând la concepte avansate si tehnici mai putin folosite.

Pentru a intelege mai bine unele aspecte referitoare la organizarea, serializarea si transmiterea informatiei cu ajutorul SOAP, cititorul trebuie sa detina cunostinte de baza despre XML. O scurta prezentare a XML poate fi gasita in anexele acestei lucrari. Dar trebuie mentionat ca XML nu face subiectul central al acestei prezentari, deci este posibil ca informatia sa nu fie suficienta.

De asemenea este necesara cunoasterea limbajului java (sintaxa si nu numai) cat si idei de baza despre organizarea unui Application Server pentru partea a doua. Partea a treia necesita cunostinte avansate de java, avand in vedere ca se vor folosi urmatoarele tehnologii: parsare de xml, conectare la baza de date folosind JDBC, multithreading.

Este recomandata cunoasterea elementelor de baza ale UML (Unified Modelling Language) pentru a putea intelege cu usurinta modularizarea si structurarea pe clase a serverelor SOAP studiate in aceasta lucrare. Aceste cunostinte sunt foarte utile in parcurgerea studiului de caz oferit de partea a treia.

In concluzie, sper că acest ghid va fi uşor de utilizat, plăcut ca exprimare si concis ca şi conţinut.

Navigare placută în continuare...

Partea I

Protocolul SOAP

Capitolul 1

Introducere în lumea Serviciilor Web

După cum cu toţii ştim, ultimele decenii au adus o adevărată explozie de produse şi tehnologii noi. Industria IT a cunoscut o dezvoltare extrem de rapidă, de multe ori ducând la modificări radicale privind arhitectura şi implementarea produselor software. Un astfel de moment a sosit şi acum, când conceptul de Serviciu Web (Web Service) se apropie de maturitate.

1.1 Înainte de Servicii Web (un pic de istorie)

Serviciile Web au apărut ca o nevoie naturală de extindere a tehnologiei informaţiei. Au la bază experienţa acumulată de la începutul erei informaţionale, experienţă ce s-a concretizat, în diverse etape, prin tehnologii şi metodologii noi, aşa numitele salturi informationale. Pentru a întelege cum şi de ce au apărut Serviciile Web, trebuie să ne întoarcem privirea către trecut, pentru a analiza momentele cheie în evoluţia tehnologiei IT.

Programarea,]n primii s[i paşi, a fost dificilă şi plină de capcane ascunse, datorate în mare parte programării nestructurate. Din această cauză multe proiecte software au întâmpinat greutăţi în dezvoltarea produselor dar în mod special în încercarea de a le întreţine sau de a le îmbunătăţi. Programatorii aveau nevoie de o mai bună organizare, de aici luând naştere conceptul de programare structurată.

Următorul hop depăşit ţine mai mult de organizarea şi reprezentarea modelelor. Deşi programatorul avea la dispoziţie un model de organizare a produsului, folosirea programării structurate nu era suficientă. Îi lipsea naturaleţea organizării datelor. Astfel a apărut un nou model de programare, denumit *Programarea obiect orientată*. De ce astfel? Trebuie doar să aruncăm o privire în jur şi o să vedem că suntem înconjuraţi de obiecte: maşini, oameni, clădiri etc. Realitatea este o sumă de obiecte care interacţionează unele cu altele, de evenimente ce influenţează obiectele. Conform acestei noi metodologii se poate scrie cod într-o manieră mult mai elegantă şi mai apropiată de sistemul ce trebuie modelat.

În aceeaşi perioadă o nouă lume a început să prindă viaţă: *INTERNET-ul*. Omenirea trebuia să comunice rapid şi eficient. Telefonul nu mai era de ajuns, nu oferea suficientă informaţie şi conectarea era limitată. Utilizatorul dorea să acceseze informaţii variate fără un efort considerabil. Solutia a fost evidentă: o reţea globală de calculatoare legate permanent între ele. La început simplu proiect de cercetare, începând cu anii '80 a cunoscut o puternică

expansiune, aducând tot mai multe inovaţii. Aceste idei noi s-au dovedit a fi adevărate mine de aur, motiv pentru care marile companii nu au stat pe gânduri şi s-au implicat în propagarea acestora prin intermediul produselor proprii. Rezultatul: un flux tot mai mare de date şi informaţii a împânzit lumea virtuală.

La început informaţia a fost prezentată prin intermediul paginilor html. În scurt timp acestea nu au mai fost de ajuns. Utilizatorul dorea o interfată cât mai prietenoasă şi utilă. Avea nevoie de dinamism. Astfel a luat nastere ideea de script. Pentru început a fost doar la nivel de client (client-side scripting). Exemplele cele mai relevante sunt JavaScript şi JScript.

Dar lucrurile nu s-au oprit aici. Cantitatea crescândă de informaţie cerea foarte multă muncă de întreţinere şi administrare a paginilor deja existente. Schimbările şi actualizările erau (şi încă mai sunt) costisitoare. Aceste inconvenienţe au creat scripturile la nivelul serverului (server-side scripting). La început sub forma CGI, a evoluat treptat către tehnologii mult mai uşor de utilizat, cum ar fi ASP (Active Server Pages, soluţie propusă de Microsoft), JSP (Java Server Pages, standard propus de SUN şi adoptat de comun acord pentru scrierea de aplicaţii web în Java) şi PHP. Noile metode se sprijină tot pe vechiul HTML, dar conceptele sunt noi: bazat pe cererile HTML executate de client, serverul construieşte răspunsul tot ca pagină HTML pe baza informaţiilor pe care le deţine (în baza de date, în sesiunea utilizatorului, în fişiere de configurare aflate pe server şamd).

Datorită impulsului şi perspectivelor oferite de dezvoltarea programării orientate pe obiecte un nou fenomen a luat naştere: *programarea distribuită*. Când se crează aplicaţii mari, uneori este mai eficient sau chiar necesar ca anumite procese să fie executate pe sisteme diferite, crescând astfel performanţa soluţiei. Pentru a funcţiona corect, un astfel de sistem trebuie să permită comunicarea între componentele sale. Pentru aceasta trebuie stabilit un protocol de transmitere şi recepţionare a informaţiei. Ca urmare, fiecare mare firmă de dezvoltare software a încercat să îşi impună propria viziune, creând soluţii complexe precum: CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model) de la Microsoft, RMI/IIOP (Remote Method Invocation over Internet Inter-Orb Protocol) de la Sun. Dar această diversitate de produse ridica următoarele probleme:

- *Lipsa unor standarde oficiale.* Desi multe dintre soluţii au fost prezentate unor organizaţii abilitate în crearea şi promulgarea de standarde (cum ar fi W3C), ele nu au fost aprobate. De această problemă s-a lovit standardul CORBA, care, în faza actuală este folosit cu succes, dar versiunile diferă (mai mult sau mai puţin) de la un producător la altul.

- *Lipsa de interoperabilitate.* Fiecare soluţie are un mod diferit de a încapsula şi transmite informaţia prin reţea. De aici şi imposibilitatea de comunicare dintre ele. Totuşi există încercări de a trece peste acest obstacol, un exemplu bun fiind COM/CORBA bridge (o punte între tehnologiile COM şi CORBA). Dar apariţia oricărei schimbări în vreunul dintre protocoalele implicate duce la schimbări (uneori chiar majore) ce trebuie aduse acestei soluţii.

- *Comunicarea între diverse limbaje* este foarte dificil de obţinut dacă nu chiar imposibil. Un exemplu elocvent ar fi RMI/IIOP, implementat şi folosit doar în Java datorită modului de serializare specific cât şi protocolului de transport folosit.

1.2 De ce Serviciile Web?

Chiar dacă abia au început sa apară implementări de Servicii Web, conceptul a luat naştere cu ceva timp în urmă. Prima încercare a apartinut firmei Hewlett-Packard care în anul 1999 a lansat produsul *eSpeak*. Acesta permitea utilizatorilor să construiasca aplicaţii asemănătoare cu serviciile Web de acum. Dar această soluţie nu s-a raspândit foarte rapid datorită caracterului proprietar pe care îl are, depinzând în mare masură de echipamente şi tehnologii oferite de firma producătoare.

Următorul pas a fost făcut de Microsoft prin intermediul noii platforme pe care a lansat-o: .NET. Aceasta are ca şi componentă cheie conceptul de serviciu Web. Din acest moment toate marile companii au început o cursă de încorporare şi extindere a acestei noi tehnologii.

Si totuşi, ce este un Serviciu Web?. Un serviciu web este o interfaţă de comunicare oferită de server, prin care clienţii (programe aflate pe alte sisteme) pot solicita diverse informaţii. Clientul poate varia, poate fi prezent pe acelaşi calculator cu serverul, poate fi în aceeaşi reţea locală sau se poate afla în partea opusă a globului. Este o metodă prin care aplicaţiile pot comunica între ele prin mesaje asincrone sau prin apeluri de procedură la distanţă (RPC : Remote Procedure Call).

Dar acesta este doar un concept. Si pentru a nu sfârşi ca alte tehnologii care au pornit de la o idee bună dar s-au extins în tot mai multe direcţii fără a se focaliza către anumite probleme centrale, a fost nevoie de un standard pe baza căruia să se contruiască serviciile Web. Altfel zis un set de reguli care să fie strict respectate de produsele dedicate acestei arii. Ele se bazează pe experienţa anterioară avută cu alte produse asemănătoare cât şi pe nevoile programatorului. În consecinţă, un Serviciu Web trebuie:

- *să fie uşor de extins şi refolosit* în aplicaţii noi. Acest lucru este realizat prin adoptarea programării obiect orientate, cât şi prin folosirea modularizării. Un serviciu poate fi văzut ca un modul, un obiect. Clientul nu trebuie să stie că serverul se află pe altă maşină ci doar să apeleze o metodă a serviciului ca şi când acesta este un obiect ce aparţine propriului program.

- *să ofere interoperabilitate*, indiferent de platformă, sistem de operare şi limbaj de programare. Această problemă delicată a fost rezolvată prin decizia de a folosi XML (eXtended Markup Language) pentru a transmite datele prin reţea. Dar XML fiind un vocabular foarte general, s-a hotărât realizarea unei particularizări a acestuia, rezultând *SOAP (Simpe Object Access Protocol)*. Acesta doar impune un set de reguli de formatare a mesajului XML reprezentând informaţia transmisă.

- *să fie transmis prin cât mai multe căi posibile prin reţea*. Acest lucru este necesar deoarece multe companii doresc să folosească doar anumite protocoale de transport pentru o mai bună securitate. Aici trebuie menţionat faptul că cea mai bună solutie este folosirea protocolului HTTP (HyperText Transfer Protocol) datorită posibilităţii de a nu fi blocat de firewall.

- *să fie uşor de descris şi creat programe client*. Pentru această problemă ridicată de cei ce dezvoltă programele client ale serviciilor Web s-a găsit o soluţie ingenioasă: dezvoltarea unui nou subset XML denumit WSDL prin care, prin elemente prestabilite,

să se informeze clientul asupra structurilor (obiectelor) folosite de serviciu, asupra metodelor expuse şi asupra modului de accesare a acestora (parametrii, tipul parametrilor –intrare, ieşire, intrare şi ieşire–, protocoalele de transport admise, stilul şi tipul de codare şamd). Datorită caracterului fix pe care îl are acest limbaj, a fost posibilă realizarea unor utilitare pentru crearea suportului de comunicare (necesar unui client pentru a invoca un serviciu Web) doar pe baza fişierului WSDL ce descrie serviciul respectiv.

- *să fie uşor de găsit pe internet.* In alte cuvinte, dacă un proiect are nevoie de o anumită funcţionalitate, cei ce îl crează să poată căuta cu uşurintă pe internet serviciul respectiv şi să vadă dacă într-adevăr îndeplineşte condiţiile necesare pentru a putea fi folosit. Acest lucru este realizat prin folosirea unor regiştrii UDDI.

1.3 Avantaje şi dezavantaje ale Serviciilor Web

Şi dacă tot am arătat că Serviciile Web au luat naştere din nevoia de intercomunicare cât şi pe baza experienţei oferite de produsele anterioare, este timpul sa prezentăm avantajele folosirii serviciilor Web:

- Respectă un Standard Deschis (Open Standard), adică permit comunicarea între componente scrise în limbaje diferite sau care rulează pe platforme diferite.
- Au un caracter modular. Prin urmare companiile pot refolosi acelaşi serviciu în alte proiecte, fără a depune nici un efort.
- Sunt uşor de implementat iar costurile sunt reduse deoarece se bazează pe o infrastructură deja existentă (reţeaua de comunicare şi protocoalele folosite de aceasta).
- Reduce costul de integrare al aplicaţiilor internet cât şi comunicarea între aplicaţii B2B (Business to Business), fiind o alternativă profitabilă pentru companiile ce se folosesc de aceste tipuri de aplicaţii.
- Permit o implementare incrementală, gradată, evitând schimbări bruşte de tehnologii în interiorul unei organizaţii.

Dar nu se poate ca totul să fie perfect. Astfel că serviciile Web prezintă unele dezavantaje. Cel mai mare este faptul că nu există o specificaţie finală pentru standardele folosite, ea fiind încă în lucru. Cu toate acestea există o formă preliminară deja publicată, dar cu anumite paragrafe neclare (printre care se remarcă tratarea şi încapsularea excepţiilor de pe server către client).

1.4 Rularea serviciilor Web

Protocoalele de bază au fost stabilite. Dar acest lucru nu este suficient pentru a putea să rulăm servicii Web. Pentru aceasta mai rămân câţiva paşi importanţi despre care trebuie să vorbim.

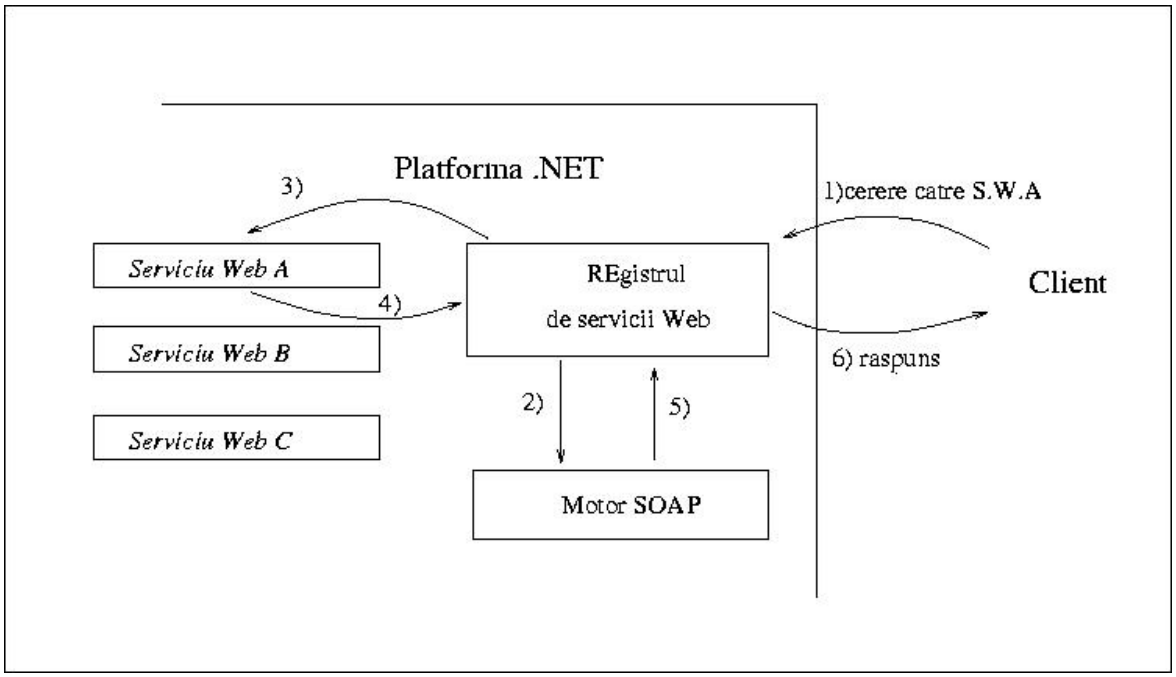

Figura 1.1: Traseul unei cereri SOAP catre un serviciu standalone pe o platforma .NET

În primul rând este necesară existenţa unei componente numită *motor SOAP (SOAP engine)*. Această componentă este responsabilă cu deserializarea cererilor de la clienţi în obiecte standard şi cu serializarea răspunsurilor pentru clienti conform specificaţiilor SOAP. Această componentă poate să fie integrată în platforma folosită (cum este cazul cu .NET) sau poate fi instalată separat, sub forma unui server (sau serviciu).

Rularea unui serviciu poate fi facută în două moduri: independent (standalone) sau în contextul unui server.

1.4.1 Rularea serviciului standalone

Acest tip de rulare este permis de platforma .NET. În realitate aceasta nu este în întregime independentă. Acest tip de serviciu este compus din două clase principale: clasa care conţine interfaţa oferită de serviciu (cât şi implementarea) şi clasa ce lansează în execuţie serviciul.

Clasa care conţine interfaţa şi funcţionalitatea serviciului este obişnuită şi trebuie să se conformeze anumitor standarde cerute de platforma .NET. Ea poate să folosească diverse alte module instalate pe platforma respectivă.

Clasa responsabilă cu lansarea serviciului reprezintă de fapt serverul şi are o formă standard. Ea anunţă platforma .NET că doreşte să înregistreze un serviciu la o anumită adresă a sistemului şi la un anumit port. Apoi trebuie să dea informaţii referitoare la serviciu: ce tip de serviciu este (singleton sau one per request), care este clasa care îi defineşte interfaţa şi implementarea cât şi metodele expuse. Apoi rămâne în aşteptare (dacă am încheia execuţia metodei principale, ar fi echivalent cu a opri serviciul Web).

In acest caz platforma .NET se comportă ca un quasiserver. Figura 1.1 prezintă traseul pe care îl urmează un mesaj SOAP de la client către server şi înapoi. Etapele sunt:

Pasul 1) Platforma .NET primeşte cererea de la client sub forma de mesaj SOAP şi îl trimite mai departe la registrul de servicii Web.

Pasul 2) Registrul deserializează mesajul prin intermediul Motorului SOAP.

Pasul 3) Pe baza mesajului decodat, registrul încarcă serviciul dorit de client şi apelează metoda cerută.

Pasul 4) Registrul primeşte rezultatul metodei cerute.

Pasul 5) Registrul serializează rezultatul metodei prin intermediul Motorului SOAP.

Pasul 6) Registrul trimite clientului rezultatul serializat în format SOAP.

1.4.2 Rularea serviciului cu server

Rularea serviciul prin intermediul unui server conferă acestuia o mai mare putere (beneficiind de mecanisme specifice unui server pentru repartizarea optimă a cererilor, optimizarea conexiunilor, folosirea de sesiuni). În acest caz este posibilă folosirea unor fişiere de configurare, unor fişiere de desfăşurare (deploy) şi de dezinstalare (undeploy) a serviciului astfel încât aceste operaţii de rutină să se desfăşoare în mod automat. Aceste fişiere speciale sunt specifice fiecărui server în parte, astfel că pentru a afla mai multe informatii trebuie consultat ghidul serverului.

Acest tip de rulare poate fi de două feluri: rularea într-un server ce suportă servicii (un exemplu îl reprezintă IIS şi serviciile .NET) sau rularea serviciului prin intermediul unei aplicaţii web instalate pe un server de aplicaţii (application server) (de ex. Apache AXIS rulând pe un server Tomcat sau Orion).

Soluţia cea mai des folosită pentru *servere ce suportă servicii* este combinaţia IIS cu servicii .NET (oferită de Microsoft) sau Apache Server cu modulul de .NET. În aceste cazuri, serverul are la dispoziţie (deobicei extern) facilităţile oferite de platforma .NET, adică Motorul SOAP nu este încapsulat în server. Spre deosebire de varianta standalone, în acest caz serverul este responsabil de gestionarea serviciilor cât şi rularea acestora, înlocuind rudimentara variantă cu un Registru De Servicii Web.

A doua soluţie este mai complexă, fiind necesară instalarea unui server de aplicaţii (de exemplu Tomcat). În interiorul acestuia se va instala o aplicatie Web care ţine locul serverului de servicii. Prin intermediul acestei aplicaţii se face gestionarea de servicii instalate şi încărcarea şi executarea acestora. În acest caz este necesară prezentarea pas cu pas a traseului pe care îl capătă o cerere SOAP:

Pasul 1) Serverul de aplicaţii primeşte o cerere de la client. Nu are relevanţă ce tip de cerere este (ar putea să fie HTTP Post de la o pagină de script sau un mesaj SOAP).

Pasul 2) Pe baza URL-ului pe care îl adresează, mesajul este trimis la aplicaţia Web ce este înregistrată la acea locaţie. Să presupunem că am primit un mesaj SOAP adresat aplicatiei Web ServerSOAP instalată pe server. În acest caz mesajul este trimis la punctul de intrare al aplicaţiei (in cazul nostru procesorul de cereri conţinut de aplicaţia ServerSOAP).

Pasul 3) Procesorul de cereri deserializează mesajul cu ajutorul Motorului SOAP conţinut de aplicaţia ServerSOAP.

Pasul 4) Procesorul de cereri localizează serviciul ce trebuie folosit, îl încarcă şi execută metoda cerută de client.

Pasul 5) Procesorul de cereri primeşte rezultatul executării metodei.

Pasul 6) Procesorul de cereri serializează rezultatul metodei folosind Motorul SOAP

Pasul 7) Procesorul de cereri trimite serverului de aplicaţii răspunsul deja serializat

Pasul 8) Serverul de aplicaţii trimite către client răspunsul.

Datorită faptului că majoritatea serverelor de aplicaţii nu suportă (înca) instalarea de servicii SOAP, este necesară folosirea unei aplicatii secundare care are ca scop implementarea funcţionalităţii cerute de specificaţiile SOAP. Această aplicatie Web (denumită în exemplul precedent ServerSOAP) este scutită de a se ocupa de nivelul de transport, acesta fiind responsabilitatea serverului.

1.5 Dezvoltarea şi viitorul serviciilor Web

1.5.1 Utilitare

După cum am menţionat mai devreme, deja există pe piaţă mai multe utilitare care au ca scop crearea şi intreţinerea cu efort minim a serviciilor Web. Acum avem prilejul să vă enumerăm unele dintre ele şi să adăugăm câteva cuvinte despre ele.

- Apache SOAP. Dezvoltat de Apache Organisation. În mare parte suportă specificaţiile SOAP 1.2. Considerat închis, versiunile ulterioare o să facă doar reparare de erori (bug-fixing), nu şi adăugare de funcţionalitate.
- Apache AXIS. Dezvoltat de Apache Organisation, este urmaşul proiectului SOAP. Oferă mult mai multă funcţionalitate, design refăcut, viteză de lucru şi folosirea memoriei mai bună.
- Platforma .NET. Dezvoltată de Microsoft. De asemenea se găseşte o implementare Ximian pentru Linux.
- Visual Net. Dezvoltat de Antarctica. Modul pentru Apache Server, printre altele oferă şi servicii .NET.
- kSOAP. Dezvoltat de Enhydra. Ofera suport doar pentru client.

O listă detaliată cu implementări SOAP poate fi studiată în anexele acestei lucrări.

1.5.2 Exemple de servicii Web

Pentru a demonstra că într-adevăr serviciile Web reprezintă o soluţie viabilă în lumea tehnologiei informaţionale, vom prezenta pe scurt câteva servicii ce au fost deja implementate şi făcute publice pentru utilizatori din orice colţ al lumii.

- Motorul de căutare Google oferă dezvoltatorilor posibilitatea de a accesa funcţionalitatea acestuia prin intermediul unui serviciu Web. Astfel pot fi dezvoltate programe ce beneficiază de capacităţile motorului Google fără a necesita interacţiunea utilizatorului prin clasica pagină de Web. Pentru mai multe informaţii consultaţi următorul URL: http://www.google.com.

- Magazinul electronic Amazon oferă, încă în stadiu experimental, un serviciu Web prin intermediul căruia se pot executa căutări produse şi de asemenea plăţi online. Pentru mai multe informaţii consultaţi următorul URL: http://www.amazon.com.

- Universitatea Berkeley din California foloşeste servicii Web pentru a integra şi îmbunătăţi sistemul de comunicaţii. Mai exact pentru a unifica emailul, voice-mailul şi mesajele fax în căsuţe poştale individuale ce pot fi accesate de proprietar prin intermediul telefonului celular, PDA-ului (Personal Digital Assistant), telefon sau client mail. În acest moment proiectul este în faza de dezvoltare. Pentru mai multe informaţii consultaţi următorul URL: http://telecom.berkeley.edu/uc/.

- Foarte multe companii aeriene, de turism, de transport, de închirieri se folosesc de servicii Web nu numai pentru a oferi clienţilor informaţii de ultimă oră cât şi pentru a integra cât mai uşor sisteme diferite. Este cazul firmelor Dollar Rent a Car şi Southwest Airlines, care au fondat un parteneriat prin care firma de transport aerian dă posibilitatea clienţilor săi să închirieze o maşină prin intermediul siteului propriu. Problema a apărut datorită diferenţei de sisteme de operare: compania aeriană foloseşte UNIX pe când cea de inchirieri Windows. Soluţia aleasă de Dollar este clară: implementarea unui serviciu Web prin intermediul căruia să se poată face rezervări pentru maşinile sale. Cu această ocazie, acest serviciu va putea fi folosit nu numai de Southwest Airlines dar şi de către alţi parteneri. Pentru mai multe informaţii consultaţi următorul URL: http://www.iflyswa.com/cgi-bin/carItinerary.

- Companiile din domeniul financiar au mereu nevoie de valori la zi legate de stocuri, de acţiuni şamd. Pentru a facilita accesul la astfel de informaţii pentru agenţii de bursă, a implementat un serviciu Web care returnează valorile curente ale cotelor. Acest lucru a dus la o mai bună organizare a informaţiilor şi astfel la un lucru mult mai eficient din partea angajaţilor firmei.

- Simplu serviciu folosit pentru traducerea unui text dintr-o limbă în alta poate fi găsit la următoarea adresă: http://babelfish.altavista.com/.

- Si daca totuşi această listă nu v-a convins atunci puteţi intra pe următorul link şi nu veţi fi dezamăgiţi. XMethods este un site cu o listă impresionantă de servicii Web dintre cele mai variate. http://www.xmethods.com/

1.5.3 Noi domenii de explorat

Serviciile Web sunt doar la început. Deşi foarte promiţătoare, ele încă mai au mult până să ajungă la deplina maturitate. Pentru aceasta există organizaţii care se ocupă de dezvoltarea de noi specificaţii, de noi protocoale şi noi metode de utilizare a acestor tehnologii.

Odată cu apariţia serviciilor Web foarte multe lucruri ce păreau greu de obţinut au devenit realizabile.

- accesarea unui stateless EJB (Enterprise Java Bean) prin intermediul SOAP este o realitate. Pentru a accesa o astfel de componentă (vezi documentaţia referitoare la Java 2 Enterprise Edition) nu mai este nevoie de complicate manevre de mapare folosind multiple tehnologii, ci doar de un simplu apel de metodă din partea clientului spre a primi rezultatul cerut. Datorită acestei realizări, aplicaţiile Web bazate pe EJB oferă acum o nouă interfaţă mult mai uşor de folosit (fiind posibil accesul chiar şi de pe telefoane mobile). Si asta fără a modifica deloc codul iniţial, doar prin realizarea unei mapări corespunzătoare în fişierele de configurare şi de desfăşurare ale aplicaţiei SOAP.
- Conectarea la baze de date folosind direct protocolul SOAP. Cu ajutorul acestei tehnologii nu va mai fi nevoie de încărcare de drivere specifice, de folosire de protocoale sau de punţi de conectare (ODBC JDBC bridge, drivere JDBC). Accesul se va face direct, executând comanda SQL cerută şi primind înapoi un rezultat deja mapat ca obiect.

1.6 EXTRA: Un pic de politică

Acest subcapitol are ca scop prezentarea unor informaţii despre stadiul în care se află specificaţiile pentru protocoalele ce fac subiectul acestui studiu cât şi despre organizaţiile care lucrează la ele. După care ne vom focaliza asupra problemei credibilităţii organizaţiei WS-I si conflictului (mai mult sau mai putin făţiş) dintre firmele mamut Microsoft şi Sun (evindent, cu referire directă la serviciile Web). După cum probabil v-aţi dat seama, decizia de a citi acest subcapitol este lăsată la latitudinea cititorului, nefiind prezentate informaţii esenţiale despre protocolul SOAP ci mai mult părerile unor programatori de pe lista oficială de discuţii a Apache AXIS şi Apache SOAP.

1.6.1 Specificaţiile

Stadiul în care se află specificaţiile este una dintre problemele cele mai stringente cu care se confruntă lumea serviciilor Web. Conceptul de serviciu Web a fost pentru prima oara aprofundat de Microsoft prin intermediul produsului .NET. În acelaşi timp o altă firmă concurentă, IBM a lucrat în stabilirea unor reguli de comunicare pentru servicii Web. Datorită efortului depus în această nouă ramură IT, aceşti doi giganţi au devenit în scurt timp autorităţi în acest domeniu. Iar ca rezultat al acestui efort si al muncii lor de dezvoltare, inţelegere şi extindere a acestui concept a luat naştere un document care este considerat baza serviciilor Web de acum: *Specificatiile SOAP 1.1.* Acelaşi grup a dezvoltat specificatiile

pentru WSDL 1.1. În schimb specificatiile pentru UDDI au fost şi sunt în continuare dezvoltate de organizaţia OASIS.

De ce aceste documente reprezintă baza serviciilor Web? Pentru că toate utilitarele care au fost implementate după apariţia lor au incercat pe cât posibil să respecte regulile enunţate în ele.

Deşi bine intenţionat, grupul de cercetători (dintre care mare parte erau angajaţi IBM şi Microsoft) se afla la început de drum. Ei trebuiau să construiască totul de la zero, să găsească formula potrivită pentru serializarea datelor cât şi modul cel mai elocvent de exprimare. Astfel că au apărut unele probleme: paragrafe nu tocmai clare, care pot duce la diverse interpretări, lipsuri în exprimare, lispuri în specificarea anumitor parametrii. Datorită acestor probleme (care par mărunte la prima vedere) a apărut un număr destul de mare de probleme de interoperabilitate între diversele implementări ce au ieşit pe piaţă.

Aceste specificaţii au fost depuse la consorţiul W3C pentru validare şi promulgare. W3C a hotărât să nu valideze aceste specificaţii, pe care le-a păstrat sub forma unor note ce pot fi găsite la următoarele adrese: `http://www.w3.org/TR/SOAP/` (SOAP 1.1) si `http://www.w3.org/TR/WSDL/wsdl.html` (WSDL 1.1). Chiar dacă nici una dintre ele nu a fost acceptată ca standard, fiind singurele specificatii în domeniul SOAP, au devenit regulile de bază pentru adăugarea de suport pentru servicii Web.

Poate că aţi observat (cu stupoare, cel putin asta a fost reacţia mea) că firma Sun (deobicei sinonim cu Java) nu s-a implicat deloc în dezvoltarea serviciilor Web prin intermediul SOAP. Chiar au afirmat că sunt sceptici în privinţa acestui nou protocol şi au hotărât să rămână la dezvoltarea de aplicaţii şi extinderea platformei Java Web Start.

În acelaşi timp Sun lucrează la un alt protocol pentru Enterprise Businnes, numit ebXML (electronic businnes using eXtensible Markup Language), reprezentând un sistem de mesage ce are la bază vocabularul XML pentru serializarea şi deserializarea datelor transmise. Specificaţiile, create de OASIS (organizaţie asemănătoare cu W3C) sunt în faza de concepţie, dar există proiecte ce implementează sau se folosesc de funcţionalitatea descrisă în draftul de specificaţie, dintre care trebuie mentionate unele dintre cele mai importante: Oracle9i Application Server, BEA WebLogic Integration, Fujitsu INTERSTAGE, IONA Orbix E2A Collaborate şi multe altele. Mai multe informaţii la siteul oficial ebXML: `http://www.ebxml.org`.

Dar să ne întoarcem la subiectul acestei lucrări: SOAP. Deşi standardul original a fost respins, acesta a fost adoptat de un număr foarte mare de dezvoltatori. Majoritatea se găsesc în USA şi Europa. Datorită numărului mare de servicii existente bazate pe acest protocol, SOAP 1.1 este considerat standardul de facto în acest domeniu. Piaţa asiatică este reluctantă în adoptarea combinaţiei SOAP&WSDL&UDDI in favoarea variantei susţinute iniţial de Sun, ebXML.

Ca urmare a acestei situaţii, s-au format două direcţii în dezvoltarea şi îmbunătăţirea SOAP. În acest subcapitol vom vorbi în mod special despre prima directie: W3C, deşi a respins propunerea venită sub forma standardului SOAP 1.1, a început dezvoltarea noii specificaţii SOAP 1.2. W3C încearcă să lămurească problemele ce au apărut în urma primei specificaţii. Acest efort se bazează pe mai multi factori:

- Studiul atent al specificaţiilor SOAP 1.1

- Răspunsul venit de la programatori în urma folosirii toolurilor implementate pe baza specificaţiilor iniţiale
- Întrebările şi sesizările ridicate de utilizatori la citirea specificaţiilor iniţiale.
- Problemele de interoperabilitate apărute în diverse situaţii şi între diferite produse.

1.6.2 SOAPBuilers şi WS-I

Aceste două organizaţii reprezintă cea de-a doua direcţie aleasă de către comunitatea programatorilor de servicii Web. Ea nu contrazice nu nimic prima direcţie ci doar încearcă să transforme interoperabilitatea dintre produsele SOAP din vis în realitate.

SOAPBuilders este o organizaţie neoficială care are ca scop transformarea visului interoperabilităţii în realitate. Deşi în linii mari interoperabilitatea este realizată, există unele mici discrepanţe între produse. Şi acesta este câmpul în care lucrează SOAPBuilders. Formată din programatori sau coordonatori ce lucrează la aproape toate produsele ce implementează specificaţiile SOAP, încearcă să stabilească principalele puncte nevralgice ale comunicării. Mai mult, se testează mereu noile versiuni ale produselor pentru a se verifica ce a fost rezolvat şi ce nu din punct de vedere al interoperabilităţii.

WS-I este o altă organizaţie oficială care are un rol asemănător: încearcă să stabilească reguli ce trebuie urmate de programatori pentru a reuşi o implementare cât mai portabilă şi cu un grad sporit de interoperabilitate cu celelalte utilitare.

Numele este o abreviere a Web Service Interoperability Organisation. Această organizaţie se caracterizează ca fiind *un efort deschis al companiilor având ca scop direct promovarea interoperabilităţii serviciilor Web, indiferent de platforme, aplicaţii şi limbaje de programare. Organizatia înglobează mare parte din liderii serviciilor Web cu scopul de a oferi utilizatorilor îndrumări, practici recomandate (recomended practices) şi resurse pentru realizarea de servicii cu grad sporit de interoperabilitate.*

Organizaţia a fost fondată în luna ianuarie a anului 2002 de patru mari companii din domeniul IT: Microsoft, IBM, BEA Systems şi Intel. Scopul declarat a fost de a oferi indicaţii şi sfaturi practice în realizarea unor servicii interoperabile. Odată cu maturizarea ei, WS-I a adunat un număr mare de firme producătoare de software şi a început o campanie activă de sprijinire a dezvoltării serviciilor web pentru o mai bună colaborare.

Spre deosebire de SOAPBuilders, WS-I se focalizează mai mult asupra codului încercând să identifice modele standard de a implementa servicii stabile. În urma acestor cercetări, au început redactarea unui document intitulat WS-I Basic Profile. La momentul redactării acestei lucrări, documentul era încă în stadiul de draft.

După cum se vede, această companie este independentă. Deşi multe alte surse au interpretat implicarea Microsoft în crearea ei ca fiind o încercare de acaparare a specificaţiilor serviciilor Web şi de promulgare a propriilor lor protocoale, acest lucru nu este adevărat. Ca dovadă, organizatia este 'condusă' de un număr de nouă oameni (la ora actuală) fiecare fiind angajat al unei companii ce s-a implicat în dezvoltarea SOAP. Microsoft nu deţine decât un singur scaun.

Adevarata problema a apărut datorită faptului că Sun nu a fost invitată să adere la această organizaţie. Acest lucru a fost apoi explicat de organizaţie prin modul indiferent cu care Sun trata protocolul SOAP, încercând pe cât de mult cu putinţă să favorizeze propriul protocol (ebXML). Acest lucru a fost realizat şi prin interventia Microsoft, care a conditionat prezenţa sa în consorţiu de excluderea participării firmei Sun. (marturie stau documentele din procesul antitrust intentat companiei Microsoft: `http://news.zdnet.co.uk/story/0,,t288-s2110205,00.html`).

Acest lucru este pe cale să fie reparat, Sun afirmând că este de acord să facă parte din membrii ce contribuie la WS-I. În acelaşi timp şi-a afirmat intenţia de a candida pentru cele două noi locuri adăugate la comitetul director. Pentru mai multe detalii: `http://www.aspnews.com/news/article/0,,4191_1488041,00.html`.

După cum se ştie, pentru a obţine interoperabilitatea este nevoie nu numai de specificaţii cât mai clare, dar şi de companii mature, care să fie în stare să treacă peste divergenţele de opinie şi peste mândrie şi să se alăture în încercarea de a realiza produse compatibile şi uşor de folosit de către programatori.

Capitolul 2

XML pe scurt

2.1 Ce este XML?

XML este acronimul de la Extensible Markup Language (limbaj de marcare extensibil). Este un set de reguli care definesc un limbaj flexibil cu ajutorul căruia pot fi descrise documente complexe şi structuri ramificare într-un mod natural. Acest limbaj nu conţine cuvinte cheie ci doar elemente de sintaxă ce definesc corectitudinea unui astfel de document.

De la apariţia sa în anul 1998, a căpătat tot mai multă importanţă, devenind repede o componentă utilă a industriei IT. Prin intermediul lui s-a dat un nou sens ideii de structurare, modelare, organizare şi descriere a informaţiei. De asemenea a devenit o unealtă centrală în aplicaţii ce se folosesc de schimburi de informaţii (aplicaţii B2B, aplicaţii distribuite – client-server sau peer2peer – şamd).

Lumea XML se află într-o continuă dezvoltare. Apar numeroase idei de aplicare şi extindere, noi tehnologii şi concepte. Iată doar câteva dintre acestea: DTD, XML Schema, XPointer, XLink, XPath, XSLT, SOAP, WSDL, UDDI, SAX, DOM, JAXM şamd. Şi lista poate continua cu un şir şi mai lung de acronime, între care să te pierzi.

Întrebarea este: câte tehnologii legate de XML trebuie cunoscute pentru a putea înţelege în profunzime serviciile Web? Răspunsul este îmbucurător: este nevoie doar de cunoştinţele de bază despre limbajul XML (elementele, sintaxa şi regulile de validitate) şi de asemenea câteva alte elemente de bază referitoare la alte tehnologii auxiliare. Majoritatea acestor informaţii vor fi prezentate pe parcursul acestui capitol, astfel:

- Sintaxa si regulile pe baza cărora se construieşte un document XML
- Spaţiile de nume, folosite pentru a identifica unic un element
- Validarea documentelor XML cu ajutorul documentelor DTD şi XML Schema
- Parsarea documentelor XML. Alegerea unei metodologii adecvate situaţiei: DOM, SAX sau o tehnică hibrid.

În perioada de început a folosirii XML, acesta era folosit cu precădere ca mecanism de reprezentare a documentelor semistructurate, cum ar fi manuale tehnice, documentele legale

sau cataloagele de produse. Aceste documente sunt destinate utilizării umane (editare şi citire de către om). O arhicunoscută extensie XML este reprezentată de vocabularul XHTML, reprezentând documentele HTML cu sintaxă XML corectă. Iată un exemplu de centrare asupra documentului:

```
<HTML>
    <HEADER><TITLE>Lista de recuzite</TITLE></HEADER>

    <BODY>
        <H2>Scurta lista<H2>
        <P>
            Aceasta lista contine informatii despre necesitatile
            unui <B>scolar</B>
        </P>
        <UL>
            <LI> Penar pentru a tine creioanele si stiloul </LI>
            <LI> Rigla pentru trasarea de linii </LI>
            <LI> Trusa de trigonometrie </LI>
        </UL>
    </BODY>
</HTML>
```

Odată cu trecerea timpului, a devenit evident faptul că XML poate aduce avantaje când este folosit pentru a marca informaţie înalt structurată, precum reprezentarea textuală a datelor dintr-o bază de date relaţională, informaţii despre structurile de date ale unui limbaj de programare şamd. Tipul de document XML centrat pe date este folosit (editat şi citit) în general de utilitare, fiind un mod preferat de organizare a datelor în vederea comunicării între mai multe aplicaţii.

Un exemplu elocvent de document centrat pe date îl reprezintă structura următoare, care conţine informaţiile despre o persoană. Acest exemplu va fi folosit pe parcursul întregului capitol, pentru a exemplifica structura unui document XML standard.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document ce organizeaza informatiile despre o persoana -->
<Persoana sex='M'>
    <Nume>Dovlecel</Nume>
    <Prenume>Alexandru Ovidiu</Prenume>
    <DataN>17.12.1980</DataN>
    <!-- Adresa persoanei este un element compus-->
    <Adresa>
        <Strada numar="3" bloc="P9">Barbu Lautaru</Strada>
        <Scara>A</Scara>
        <Apartament>10</Apartament>
        <Oras>Brasov</Oras>
        <Judet>Brasov</Judet>
        <Tara>Romania</Tara>
    </Adresa>
    <!-- O persoana poate avea mai mult de o
```

```
17      informatie de contact-->
18      <ContactInfo>
19          <Contact type="telefon">0268/330969</Contact>
20          <Contact type="email">AODovlecel@hotmail.com</Contact>
21          <Contact type="email">dovle@delsyne.ro</Contact>
22      </ContactInfo>
23  </Persoana>
```

2.2 Structura unui document XML

Acest subcapitol va folosi exemplul precedent pentru a explica mai bine elementele componente şi regulile de sintaxă necesare pentru a scrie un document XML valid. Pentru început structura standard este următoarea: o zonă de prolog care este facultativă şi conţine instrucţiuni de procesare şi zona de date, care conţine elementul rădăcină al documentului.

2.2.1 Prologul documentului

Prologul unui document XML este opţional. El este alcătuit din una sau mai multe instrucţiuni de procesare. O instrucţiune de procesare este conţinută între `<?`, respectiv `?>`. Acestea sunt folosite pentru a identifica documentul ca fiind unul XML, pentru a specifica versiunea de XML folosită şi codarea documentului.

În general aplicaţiile XML orientate spre date nu folosesc instrucţiuni de procesare, conţinând informaţiile necesare în interiorul documentului propriuzis. Dar totuşi se recomandă plasarea în prologul documentului a unei instrucţiuni de procesare speciale, prin care se declară versiunea de XML şi tipul de codare. O astfel de instrucţiune este prezentă în documentul exemplu pe linia 1.

De asemenea prologul poate conţine şi comentarii referitoare la conţinutul documentului în cauză. Un comentariu este conţinut între semnele `<!--` respectiv `-->`. Comentariile pot să apară oriunde într-un document XML. Ele nu pot fi imbricate. Sunt ignorate de aplicaţiile XML. Exemple de comentarii pot fi găsite la liniile 2, 7 şi 16–17.

2.2.2 Elemente XML

Termenul de *element XML* este o denumire tehnică pentru asocierea unei etichete de început cu o etichetă de sfârşit într-un document XML. De exemplu elementul rădăcină al documentului exemplu are eticheta de start `<Persoana sex='M'>` şi eticheta de sfârşit `</Persoana>`. Elementul XML include şi elementele imbricate, în cazul nostru fiind vorba despre elementele aflate între liniile 4 şi 22 ale documentului.

Numele unui element este valid dacă respectă următoarele reguli:

- Conţine caractere alfanumerice (0–9, A–Z, a–z)

- Conţine caracterele liniuţă de subliniere (underscore _), cratima(hyphen -) şi două puncte (colon :)
- Începe cu o literă

De remarcat că limbajul XML este sensibil la tipul literei (case sensitive).

Un element poate avea trei tipuri de conţinut: element-only (doar elemente imbricate), text-only (doar text) şi mixt. În general sunt folosite doar primele două tipuri de conţinut pentru a crea o structură cât mai uşor de înţeles şi procesat. În cazul documentului exemplu, elementul `ContactInfo` (între liniile 18 şi 22) este unul care conţine doar elemente imbricate, pe când elementele `Contact` conţin doar text (liniile 19, 20 şi 21).

Pentru a scurta scrierea elementelor vid (care nu conţin text sau alte elemente imbricate), s-a definit şi o notaţie scurtă. Astfel elementul `<Element tip="paragraf"></Element>` poate fi scris, mai scurt, `<Element tip="paragraf" />`.

2.2.3 Atributele XML

Etichetele de start pentru elemente XML pot avea, opţional, unul sau mai multe atribute. Un atribut este reprezentat de o pereche de tip `nume=valoare`. Numele unui atribut trebuie să îndeplinească aceleaşi restricţii ca şi numele unui element XML. Valoarea atributului trebuie introdusă sub formă citat, putând fi folosite semne de citat simple, cât şi duble, cu condiţia ca acestea să fie potrivite corect.

Exemple de atribute pot fi găsite şi în documentul exemplu prezentat anterior. Astfel atributul `sex` al elementului `Persoana` are valoarea M (linia 3) iar elementul `Strada` conţine două atribute care să identifice mai bine elementele legate de strada în cauză:`numar` şi `bloc` (linia 9).

2.2.4 Sumar de reguli

Pentru ca un document XML să fie valid, el trebuie să îndeplinească anumite reguli de sintaxă, prestabilite de specificaţiile XML:

- Documentul XML trebuie să conţină un singur element rădăcină.
- Orice element trebuie să fie valid. Astfel, el trebuie să aibă un nume valid, atribute valide şi trebuie să conţină o etichetă de început şi una de sfârşit (cu excepţia formei scurte pentru elemente vide).
- Elementele trebuie încapsulate corect. Astfel orice element copil trebuie închis înainte de a închide elementul părinte. De exemplu structura `<a><b></a></b>` nu este o structură XML validă.

2.2.5 Caracterele speciale

De multe ori este necesară folosirea unui caracter care nu poate apărea într-un document XML deoarece ar contrazice vreuna din regulile de validitate enunţate mai sus. De aceea limbajul XML defineşte un număr de cinci entităţi care pot fi accesate printr-o secvenţă de simulare. Iată care sunt acestea:

Caracter	Secvenţă de simulare
¡	&alt;
¿	&gt;
&	&amp;
’	&apos;
”	&quot;

Folosirea unei astfel de metode asigură introducerea unor date mai delicate, dar sporeşte dimensiunea documentului şi gradul de dificultate pentru editarea acestora. De aceea limbajul XML defineşte o altă modalitate pentru a introduce text ce ar putea strica structura documentului. Această metodă de bazează pe folosirea construcţiei CDATA. Aceasta permite introducerea unei secvenţe de caractere permise de codarea documentului, care nu include `]]>`. Iată un exemplu în care un element XML cu numele de `Exemplu` conţine un text cu mai multe caractere speciale XML:

```
<Exemplu><![CDATA[
    Acest exemplu se foloseste de < si de > fara a
    strica structura documentului.
    ]]>
</Exemplu>
```

2.3 Spaţiile de nume(namespace)

Problema coliziunii în documentele XML apare des datorită folosirii de nume de elemente des întâlnite. Astfel documentul exemplu defineşte o anumită structură pentru elementul `Persoana` dar un alt document poate defini propria formă a elementului `Persoana` astfel:
`<Persoana id="10" nume="Dovlecel" prenume="Alex" />`.
În acest caz, dacă documentul exemplu include acest nou document, aplicaţia XML responsabilă cu interpretarea informaţiei ar avea mari probleme în a recunoaşte despre ce tip de element este vorba.

Pentru a rezolva această problemă des întâlnită în lumea programării, s-a recurs la folosirea unui mecanism de spaţiu de nume. Acesta este des întâlnit, sub acelaşi nume, în limbaje precum C++ şi C#, sau sub forma pachetelor în limbajul Java. Acest mecanism presupune definirea unui nume calificat (qualified name) pentru un element astfel:
`Nume calificat = identificator spaţiu de nume + nume local .`

Spaţiile de nume XML utilizează drept identificatori aşa numiţii *identificatori de resursa uniformă* (Uniform Resource Identifiers – URI). Ei pot fi locatori (URL),

nume sau ambele. Un spaţiu de nume valid poate fi identificat de următorul URI: `http://info.unitbv.ro/AODovlecel`. De remarcat că, dacă veţi accesa adresa respectivă, nu veţi găsi nimic. Ba chiar serverul va returna un mesaj conform căruia nu poate accesa pagina respectivă. Este *important de reţinut* că un spaţiu de nume valid nu trebuie neapărat să indice un URL care să şi existe, ci doar să presupună faptul că alte persoane nu vor folosi un astfel de identificator. În acest caz probabilitatea este scăzută ca o altă persoană să folosească acest namespace.

2.3.1 Sintaxa spaţiilor de nume

Deoarece URI-urile pot fi lungi şi de obicei conţin caractere nepermise pentru numele de elemente XML, sintaxa de includere a spaţiilor de nume în documentele XML implică doi paşi:

Pasul 1 Unui identificator de spaţiu de nume i se asociază un prefix. Acest prefix este un nume care poate conţine caracterele definite de numele unui element, cu excepţia `:`.

Pasul 2 Numele calificate sunt obţinute ca o combinaţie a prefixului definit urmat de caracterul `:` şi de numele de element local. Un nume calificat arată astfel: `prefix:NumeLocal`.

Pentru a înţelege mai bine, voi prezenta un scurt document XML ce face uz de mecanismul de spaţii de nume.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Document ce organizeaza informatiile despre o persoana -->
3  <Cunoscuti
4      xmlns:p="http://info.unitbv.ro/AODovlecel/Agenda"
5      xmlns:alt="http://info.unitbv.ro/Altcineva/Persoane"
6      >
7
8      <Persoana sex='M' >
9          <persNume>Dovlecel</Nume>
10         <Prenume>Alexandru Ovidiu</Prenume>
11         <DataN>17.12.1980</DataN>
12         <!-- Adresa persoanei este un element compus-->
13         <Adresa>
14             <Strada numar="3" bloc="P9">Barbu Lautaru</Strada>
15             <Scara>A</Scara>
16             <Apartament>10</Apartament>
17             <Oras>Brasov</Oras>
18             <Judet>Brasov</Judet>
19             <Tara>Romania</Tara>
20         </Adresa>
21         <!-- O persoana poate avea mai mult de o
22         informatie de contact-->
23         <ContactInfo>
```

```
24              <Contact type="telefon">0268/330969</Contact>
25              <Contact type="email">
26                  AODovlecel@hotmail.com
27              </Contact>
28              <Contact type="email">dovle@delsyne.ro</Contact>
29          </ContactInfo>
30      </Persoana>
31
32      <alt:Persoana id="10" nume="Dovle" prenume="Alex" />
33  </Cunoscuti>
```

În cazul procesării unui astfel de document, aplicaţia XML va putea face distincţia între cele două tipuri de elemente şi va trata diferit informaţia primită.

Un prefix se asociază unui spaţiu de nume conform sintaxei `xmlns:prefix=id spatiu de nume`. Acest lucru poate fi observat la liniile 4 şi 5 ale documentului exemplu. Prefixul poate fi utilizat doar în elementul care îl defineşte.

2.3.2 Folosirea spaţiilor de nume

Documentul prezentat mai sus ca exemplu conţine informaţie redundantă referitor la spaţiile de nume. Acest lucru se datorează efectului de propagare al unui spaţiu de nume. Daca un element este declarat ca aparţinând unui anumit spaţiu de nume, atunci şi elementele conţinute de acesta aparţin aceluiaşi spaţiu de nume (în cazul în care nu specifică explicit apartenenţa la un alt spaţiu).

Pe baza acestei observaţii putem elimina prefixarea pentru toate elementele conţinute de elementul `Cunoscuti` care aparţin spaţiului de nume prefixat cu `p`. Mai exact, se poate elimina prefixul de la etichetele aflate între liniile 8 şi 30 ale documentului. Linia 32 îşi păstrează prefixul deoarece acesta este diferit de prefixul elementului părinte.

De asemenea se poate seta un spaţiu de nume implicit pentru un element folosind următoarea sintaxă:
`<NumeElement xmlns="id spatiu nume">...</NumeElement>`.
Acest lucru este echivalent cu următoarea construcţie:
`<prefix:NumeElement xmlns:prefix="id spatiu nume">...</prefix:NumeElement>`.

Pe baza acestei observaţii se poate îmbunătăţi lizibilitatea documentului XML prin transformarea spaţiului de nume prefixat `p` ca spaţiu de nume implicit pentru elementul rădăcină.

Iată cum va arăta documentul XML prezentat anterior în urma transformărilor:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Document ce organizeaza informatiile despre o persoana -->
3 <Cunoscuti
4     xmlns="http://info.unitbv.ro/AODovlecel/Agenda"
5     xmlns:alt="http://info.unitbv.ro/Altcineva/Persoane"
6     >
7
```

```
8     <Persoana sex='M' >
9         <persNume>Dovlecel</Nume>
10        <Prenume>Alexandru Ovidiu</Prenume>
11        <DataN>17.12.1980</DataN>
12        <!-- Adresa persoanei este un element compus-->
13        <Adresa>
14            <Strada numar="3" bloc="P9">Barbu Lautaru</Strada>
15            <Scara>A</Scara>
16            <Apartament>10</Apartament>
17            <Oras>Brasov</Oras>
18            <Judet>Brasov</Judet>
19            <Tara>Romania</Tara>
20        </Adresa>
21        <!-- O persoana poate avea mai mult de o
22        informatie de contact-->
23        <ContactInfo>
24            <Contact type="telefon">0268/330969</Contact>
25            <Contact type="email">AODovlecel@hotmail.com</Contact>
26            <Contact type="email">dovle@delsyne.ro</Contact>
27        </ContactInfo>
28    </Persoana>
29
30    <alt:Persoana id="10" nume="Dovle" prenume="Alex" />
31 </Cunoscuti>
```

Atributele XML pot avea de asemenea un spaţiu de nume. Deşi destul de dificil de imaginat o astfel de situaţie, o astfel de funcţionalitate poate deveni foarte utilă în procesarea anumitor documente XML. Spre exemplu vom vedea că atributul `xsi:type` este extrem de important în verificarea tipului unui element pe baza XML Schema. De asemenea acest atribut este folosit şi de codificarea SOAP.

2.4 Validarea documentului XML

XML este un limbaj flexibil, lăsând la dispoziţia utilizatorului posibilitatea de a defini elemente şi structuri noi. Dar sunt cazuri (îndeosebi cele în care documentele trebuie procesate de o aplicaţie) când este necesară verificarea structurii şi a informaţiilor introduse. Pentru a evita realizarea procesului de verificare direct de către aplicaţia XML, s-a ales o soluţie mult mai elegantă şi modulară: definirea unui limbaj pe baza căruia să se poată stabili structura unui document XML.

2.4.1 Definiţii tip de document

Primul limbaj oficial de descriere al unui fişier XML este DTD (document type definitions). Acesta presupune descrierea structurii unui document XML prin intermediul unor informaţii de procesare. Aceste informaţii pot fi plasate într-un fişier separat şi referite de către

documentul XML sau pot fi incluse în documentul XML.

Cu ajutorul limbajului DTD se pot specifica următoarele lucruri referitoare la un document XML:

- Elementele ce pot apărea în documentul XML
- Identificarea ordinii şi a relaţiilor dintre elemente
- Identificarea atributelor pentru fiecare element, a faptului că sunt opţionale sau nu şi de asemenea verificarea valorii cu ajutorul anumitor măşti.

Pe scurt, structura unui document XML poate fi validată pe baza unui document DTD (conţinutul elementelor, atributele permise şi tipurile lor de valori). Deşi un pas înainte, micşorând considerabil cantitatea de cod de verificare a aplicaţiei XML, DTD-urile au câteva deficienţe notabile:

- Ele nu utilizează marcaj XML. De aceea o aplicaţie XML care face verificarea pe baza unui document DTD trebuie să ştie să interpreteze şi acest limbaj.
- Deoarece au fost proiectate înainte de apariţia spaţiilor de nume, DTD-urile nu prezintă facilităţi mulţumitoare pentru lucrul cu acestea.
- Nu oferă capabilităţi de reutilizare
- Structurarea elementelor este de multe ori prea restrictivă (în mod special ordinea în care apar elementele şi imposibilitatea definirii numărului de repetări admise de un element)
- Nu definesc nici o noţiune despre tipuri de date.

Din aceste motive unul din principalele protocoale de servicii Web interzice în mod explicit folosirea DTD-urilor pentru definirea structurii documentului.

2.4.2 XML Schema

XML Schema este de asemenea un limbaj de descriere a unui document XML. Prin intermediul lui se pot face validări ale structurii documentului XML. În comparaţie cu DTD, XML Schema este un limbaj puternic, complex, care prezintă numeroase avantaje:

- Suport sporit pentru lucrul cu spaţii de nume
- Colecţie standard de date simple (integer, string, date şamd)
- Suport pentru definire de structuri de date complexe (elemente şi atribute)
- Schemele sunt exprimate cu ajutorul limbajului XML. O aplicaţie ce procesează date XML are posibilitatea de a interpreta o schemă folosind aceleaşi metode de parsare ca şi pentru documentul XML.

XML Schema defineşte două spaţii de nume standard prin intermediul cărora se vor putea crea documente schemă:

- *http://www.w3.org/2001/XMLSchema*, prefixat deobicei *xsd* (XML Schema Definition) defineşte elementele cu ajutorul cărora se construieşte un document schemă. De asemenea defineşte şi tipurile simple recunoscute de schema.
- *http://www.w3.org/2001/XMLSchema-instance*, prefixat deobicei *xsi* (XML Schema Instance) defineşte atribute ce sunt parte a specificaţiei schemei. Aceste atribute pot fi aplicate elementelor în documentele XML pentru a furniza informaţii suplimentare unui procesor XML.

Aceasta este lista de tipuri simple pe care XML Schema o recuoaşte: string, base64Binary, hexBinary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, decimal, boolean, time, dateTime, duration, date, Name, QName, anyURI, ID, IDREF. Deoarece protocolul SOAP (vom vedea în capitolul următor) se bazează pe XML Schema pentru a defini tipul atributelor obiectelor (folosind atributul `xsi:type`), acestea sunt şi tipurile acceptate de acesta (cu unele restricţii).

Atributul `xsi:type` este folosit pentru a transmite aplicaţiei XML informaţii referitoare la tipul datelor conţinute de elementul respectiv. Prin intermediul lui aplicaţia XML poate efectua verificări suplimentare şi de asemenea poate implementa o formă primitivă de polimorfism.

Acestea sunt doar câteva elemente ce fac din XML Schema un limbaj puternic pentru definirea structurii documentelor XML. Dacă doriţi să aflaţi mai multe informaţii referitor la acest subiect, se recomandă parcurgerea specificaţiilor oficiale W3C. Un pas înainte îl reprezintă parcurgerea unui tutorial XML Schema.

2.5 Procesarea documentului XML

Voi încheia acest scurt tur în lumea XML prin a descrie, pe scurt, tehnici deja existente privind procesarea informaţiei conţinute de un document XML.

Procesarea reprezintă procesul prin care o aplicaţie interpretează informaţiile conţinute de un document XML. Pentru a putea procesa un document, o aplicaţie trebuie mai întâi să îl poată analiza. Analiza este un proces care implică împărţirea textului unui document XML în porţiuni mici de identificatori, numite noduri. Nodurile pot fi etichete, atribute, comentarii, text şamd. Acestea sunt transmise aplicaţiei prin intermediul unui API predefinit.

Există patru modele de analiză utilizate în mod normal:

- *Pull Parsing (Analiză prin tragere).* Aplicaţia are ca rol extragerea, pe rând, a fiecărui element din documentul XML. Încă nu este definit un API standard pentru acest model de analiză.
- *Push Parsing (Analiză prin împingere).* Analizorul trimite notificări aplicaţiei despre tipurile de noduri pe care le întâlneşte în procesul de analiză. Standardul de facto pentru acest tip de analiză este denumit *SAX (Simple API for XML)* .

- *One Step Parsing (Analiză într-un pas).* Analizorul citeşte întregul document XML şi generează o structură de date (arbore de analiză) care descrie întregul lui conţinut. Standardul utilizat pentru acest model de analiză este DOM (Document Object Model). De asemenea se lucrează la particularizarea API-ului DOM pentru Java (JDOM).

- *Hybrid Parsing (Analiză hibrid).* Acest tip de analiză combină tehnologiile prezentate mai sus pentru a optimiza, pentru anumite cazuri particulare, accesul la informaţia conţinută în documentul XML.

Nu există o tehnologie optimă pentru procesare. Totul se rezumă la a alege tipul de analiză optimă în funcţie de problema ce trebuie rezolvată. Astfel, dacă este vorba de procesarea unui document în timp cât mai scurt şi cu resurse minime de memorie, API-ul SAX este recomandat. În cazul procesării complexe a unui document, necesitând posibilitatea de navigare printre noduri, este recomandată folosirea API-ului DOM.

Capitolul 3

SOAP pe scurt

Conceptul de serviciu Web a prins repede contur şi, în paralel cu acesta, protocoalele de comunicare bazate pe XML. În acest context, ca rezultat al muncii de cercetare şi dezvoltare a companiilor Microsoft şi IBM, a luat naştere un protocol care a devenit un standard de facto pentru implementarea serviciilor Web:*SOAP*.

3.1 Ce este SOAP

SOAP este acronimul pentru *Simple Object Access Protocol*. După cum îi zice şi numele, SOAP este un protocol simplu de accesare a obiectelor. Mai pompos, a fost caracterizat astfel: "O infrastructură universală de calcul distribuit, pe baza XML". O analiză sumară a acestei definiţii ne duce la următoarele concluzii:

- *Infrastructură* implică faptul că protocolul SOAP este destinat dezvoltării de aplicaţii de bază, nu doar pentru simple aplicaţii B2B şi eCom. Presupune servere capabile SOAP, care ascund nivelul de transport.
- *Universală* , prezentă peste tot. Această tehnologie se doreşte a fi un standard omniprezent, folosit pentru a uşura comunicarea dintre aplicaţii.
- *Calcul distribuit* implică faptul că SOAP poate fi utilizat pentru a realiza comunicarea dintre aplicaţii, procese. Această comunicare este independentă de platformă, sistem de operare şi limbaj de programare, activând interoperabilitatea aplicaţiilor la distanţă.
- *Pe baza XML* subliniază faptul că SOAP este un protocol contruit pe baza unor reguli impuse limbajului XML.

Din alt punct de vedere, SOAP este cel mai bun efort al industriei IT de a standardiza tehnologia de infrastructură pentru calculul distribuit multiplatoformă pe baza XML.

Deoarece SOAP este concentrat pe aspectele comune ale tuturor scenariilor de calcul distribuit, el furnizează următoarele:

- Un mecanism de definire a unităţii de comunicaţie. SOAP încapsulează orice mesaj transmis într-un plic (envelope) cu structură prestabilită.
- Un mecanism de gestionare a erorilor. Acesta trebuie să poată transmite informaţii despre natura erorii şi locul producerii acesteia.
- Un mecanism de extensibilitate. Prin intermediul lui noi funcţionalităţi pot fi adăugate pe aceeaşi structură de bază. Punctul cheie îl reprezintă versionarea prin intermediul spaţiilor de nume.
- Un mecanism flexibil pentru reprezentarea datelor, care permite schimbul de date deja formatate (text sau XML) cât şi convenţii pentru reprezentarea structurilor de date abstracte într-un limbaj de programare.
- O convenţie pentru reprezentarea RPC (Remote Procedure Calls). Acest lucru implică definirea unor structuri standard pentru un apel de procedură la distanţă şi de asemenea pentru transmiterea răspunsului.
- O abordare centrată pe document, pentru a reflecta modele de schimb de document mai naturale. Este o alternativă la RPC, pentru a evita granularizarea interfeţelor.
- Un mecanism de legătură a mesajelor SOAP la HTTP, acesta fiind cel mai utilizat protocol de comunicaţie.

3.2 Cadrul de împachetare

Cea mai importantă parte specificată de SOAP se referă la cadrul de împachetare. Deşi constă în doar câteva elemente XML, oferă structura şi mecanismele de extensibilitate care fac SOAP atât de potrivit ca fundament pentru calculul distribuit pe bază de XML. Organizarea standard a unui mesaj SOAP este simplă: mesajul este reprezentat de un element rădăcină numit plic SOAP (SOAP envelope). Acesta poate conţine opţional o zonă în care pot fi plasate unul sau mai multe antete SOAP (SOAP headers) şi un corp (body) ce cuprinde datele transmise prin intermediul mesajului. Orice număr de elemente XML pot urma elementului corp.

Pentru a înţelege mai bine, iată cum arată un apel RPC pentru metoda *cautaPersoana(Criteriu)*.

Listing 3.1: Apel RPC al metodei cautaPersoanaDupaCriteriu

```
1 <?xml version="1/0" encoding="UTF-8"?>
2
3 <SOAP-ENV:Envelope
4     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
5     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8     xmlns:myns="http://info.unitbv.ro/dovle" >
9
```

```
10    <SOAP–ENV:Header>
11       <auth:ClientAuthentication
12           xmlns:auth="http://info.unitbv.ro/AODovlecel"
13           SOAP–ENV:mustUndestand="1" >
14           <user>dovle</user>
15           <passwd>mypasswd</passwd>
16       </auth:ClientAuthentication>
17    </SOAP–ENV:Header>
18
19    <SOAP–ENV:Body>
20       <cautaPersoanaDupaCriteriu>
21          <criteriu xsi:type="myns:Criteriu" >
22             <nume xsi:type="xsd:string">Muresan</nume>
23             <prenume xsi:nil="true" />
24             <adresa xsi:type="myns:Adresa">
25                <strada xsi:nil="true" />
26                <numar xsi:nil="true" />
27                <oras xsi:type="xsd:string">Brasov</oras>
28                <judet xsi:nil="true" />
29             </adresa>
30             <telefon xsi:nil="true" />
31          </criteriu>
32       </cautaPersoanaDupaCriteriu>
33    </SOAP–ENV:Body>
34
35 </SOAP–ENV:Envelope>
```

Liniile 3–8 definesc eticheta de start a elementului plic SOAP. De asemenea sunt prezente definirile de spaţii de nume specifice SOAP ce vor fi folosite în mesaj. Liniile 10–17 definesc zona de antet, care conţine un antet simplu de autentificare. Liniile 19–33 încapsulează corpul mesajului SOAP.

3.2.1 Plic SOAP (Envelope)

Un mesaj SOAP este, din punct de vedere XML, încapsulat într-un element *envelope* conţinut de spaţiul de nume *http://schemes.xmlsoap.org/soap/envelope/*. Plicul presupune definirea spaţiilor de nume standard pentru protocolul SOAP:

- atributul *SOAP-ENV:encodingStyle* defineşte tipul de codare al mesajului. Exemplul conţine valoarea standard.
- *SOAP-ENV* reprezintă spaţiul de nume ce conţine elementele unui plic SOAP (eticheta de antet şi cea de corp).
- *xsd* defineşte spaţiul de nume pentru definiţia XML Schema. În acest caz se va folosi codificarea din schema 2001.

- *xsi* defineşte spaţiul de nume pentru atributele unui element din XML Schema.

Plicul conţine şi informaţia de versionare a mesajului. Aceasta nu este ţinută sub formă numerică, aşa cum fac mai toate produsele, ci este reprezentată de spaţiul de nume de împachetare (cel care are asociat prefixul SOAP-ENV). În cazul SOAP, serverul poate spune doar dacă mesajul are o versiune pe care o recunoaşte sau nu. Nu se poate pronunţa asupra compatibilităţii versiunii recunoscute cu cea prezentă în mesaj. Astfel se limpezeşte mult procesul de stabilire a versiunii protocolului între client şi server (ori serverul cunoaşte versiunea de împachetare a mesajului caz în care se continuă comunicarea, ori serverul nu cunoaşte versiunea în cauză, şi răspunde cu un mesaj de eroare).

3.2.2 Antet SOAP(Header)

Antetul unui mesaj SOAP reprezintă mecanismul primar de extindere a funcţionalităţii protocolului SOAP. Specificaţiile cer doar ca anteturile transmise prin intermediul zonei respective să fie elemente XML valide, neadăugând alte restricţii.

Zona de antet este reprezentată de elementul `SOAP-ENV:Header` şi este opţională. Dacă este prezent, atunci trebuie să fie primul copil al elementului `SOAP-ENV:Envelope` şi să includă orice număr de intrări (denumite *anteturi*). Deci un antet este un element XML descendent direct din elementul `SOAP-ENV:Header`.

Mai trebuie menţionat că un antet defineşte propriul spaţiu de nume de care aparţine. De asemenea este important de ştiut că un antet poate include atributul `SOAP`ENV:mustUnderstand+. Dacă acesta este prezent şi are valoarea 1 atunci serviciul este obligat ca, dacă nu poate înţelege acest antet (nu ştie să îl proceseze) să răspundă cu un mesaj de eroare (fault).

În listingul 3.1 (prezentat anterior), zona antet este plasată între liniile 10 şi 17 inclusiv şi conţine un singur antet ce aparţine de spaţiul de nume etichetat cu prefixul `auth`. De asemenea linia 13 specifică faptul că aplicaţia este obligată să proceseze acest antet (atributul mustUnderstand având valoarea 1).

Aceste anteturi sunt specifice serviciului care le interpretează şi pot fi folosite pentru transmitere de informaţie de autentificare, informaţie despre identificatorul tranzacţiei de pe server, informaţii despre identificatorul sesiunii folosite.

Deşi încă nestandard, există un mecanism de păstrare a sesiunii între apeluri (statefull session) prin folosirea unui antet SOAP care transferă de la server la client identificatorul acesteia. Clientul, de fiecare dată când execută un apel, retransmite antetul ca parte din zona de antet a apelului iar serverul interpretează antetul şi atribuie sesiunea asociată apelului.

3.2.3 Corp SOAP (Body)

Elementul `SOAP-ENV:Body` cuprinde în mod direct informaţia care stă la baza mesajelor SOAP. Toţi copiii (urmaşii) direcţi ai elementului `Body` se numesc corpuri. Corpurile pot conţine XML arbitrar. În general pe baza tipului mesajului SOAP (RPC sau message) pot guverna anumite convenţii asupra structurii corpului mesajului SOAP. În momentul de faţă,

astfel de reguli sunt specificate în legătură cu apelurile de procedură la distanţă şi vor fi prezentate într-unul din subcapitolele următoare.

3.3 Codarea datelor

Specificaţiile SOAP conţin informaţii referitoare la modul în care modelul obiectelor este mapat la limbajul SOAP XML. De asemenea prezintă algoritmi de bază pentru executarea următoarelor trei sarcini:

- Având informaţii despre organizarea unui model de obiecte, să se construiască o schemă XML pentru descrierea acestuia
- Fiind dată o listă de obiecte ce respectă modelul de mai sus (un graf instanţă al modelului de obiecte), să se genereze cod XML care să se conformeze schemei. Aceasta este operaţia de *serializare*.
- Fiind dat cod XML care se conformează schemei, să se creeze o structură de obiecte care să îndeplinească condiţiile modelului de la primul punct. Această operaţie se numeşte *deserializare*.

Deşi scopul codării pare a fi simplu (de a mapa elementele XML la obiecte specifice limbajului de programare folosit), regulile efective de realizare pot fi complicate. Această secţiune va prezenta, pe scurt, modul în care se desfăşoară această acţiune pentru trei tipuri de date: datele simple, datele complexe şi pentru şiruri.

3.3.1 Reguli generale

Codarea SOAP foloseşte un sistem de tipuri bazat pe XML Schema. Astfel majoritatea tipurilor prezente în XML Schema sunt folosite de SOAP pentru a mapa informaţia în cazul *tipurilor simple* (cunoscute şi ca tipuri scalare în limbajele de programare). Un astfel de obiect simplu este mapat în XML ca un element ce conţine text.

În afară de aceste tipuri simple mai există *tipurile compuse*, acestea fiind alcătuite din mai multe părţi. Un astfel de obect compus este mapat în XML printr-un element cu conţinut imbricat, fără text. Pornind de la tipurile compuse, se disting două categorii: *structurile* (care au elementele imbricate deosebite prin numele lor) şi *tablourile* (ale căror părţi se disting doar prin poziţia lor de ordine).

Valorile sunt instanţe de tipuri, ele fiind reprezentate ca elemente XML al căror tip este tipul valorii. Valorile simple sunt codate ca şi conţinutul elementelor care au un tip simplu, cu conţinut text (nu au nici un element copil). Valorile compuse sunt codate ca şi conţinutul elementelor care au un tip compus şi sunt alcătuite din elementele copil conţinute de acestea. Ele nu pot să conţină elemente text.

Valorile nu sunt niciodată codate ca atribute ci doar ca şi conţinutul unui element. Utilizarea atributelor este rezervată codării SOAP însăşi, după cum se va vedea mai departe.

Orice element XML creat prin serializarea unui obiect conţine un atribut `xsi:type` prin intermediul căruia se determină maparea ce va fi folosită. De asemenea acest atribut este folosit pentru a realiza polimorfismul SOAP. Codarea SOAP permite ca un subtip să apară în orice loc în care poate să apară un supertip. În acest caz, fără atributul `xsi:type` nu se poate realiza o bună deserializare a unui astfel de element.

SOAP permite şi transmiterea valorilor nule. Conform specificaţiilor, există două modalităţi prin care se poate realiza acest lucru: prin omiterea elementului care are valoare nulă (dacă este un atribut al unui obiect, lipsa acestuia presupune valoarea nulă) sau prin folosirea atributului `xsi:nil` (acesta trebuie setat cu valoarea "true" în caz că atributul este nul).

3.3.2 Codarea datelor simple

Datele simple se codează pe baza tipurilor din XML Schema. Pentru a înţelege cel mai bine acest lucru analizaţi câteva elemente prezente în exemplul de mesaj SOAP prezentat în listingul 3.1. Astfel de date simple sunt serializate în liniile 22, 23, 25–28. Unele dintre ele sunt marcate ca nul. În acest caz specificaţiile prezenţa atributului `xsi:type` nu este obligatorie.

După cum se observă, valoarea asociată acestor tipuri este trecută ca element text conţinut de elementul ce defineşte tipul respectiv. Maparea valorilor se face conform XML Schema, care defineşte cum trebuie să arate o valoare pentru fiecare din tipurile admise.

3.3.3 Codarea structurilor

Structurile sunt codate prin intermediul elementelor XML ce conţin alte elemente copii. Acesti copii ai elementului iniţial trebuie să aibă nume diferit pe baza căruia se vor face legăturile dintre atributele obiectului ce trebuie deserializat si elementele copil prezente. Prezenţa atributului `xsi:type` este obligatorie pentru a permite maparea dintre tipul XML şi tipul din limbajul de programare aferent. Din această cauză se poate folosi şi polimorfismul prin intermediul SOAP.

Exemplul din listingul 3.1 ilustrează modul în care se execută codarea structurilor prin elementele `criteriu` (începând cu linia 21) şi `adresa` (începând cu linia 24).

3.3.4 Codarea şirurilor

Şirurile, ca şi structurile, sunt tot elemente compuse. Ele sunt reprezentate tot printr-un element ce conţine unul sau mai multe elemente copil, dar de data aceasta aceste elemente au acelaşi nume, dar contează ordinea în care apar. În unele cazuri este folosit şi un atribut care determină poziţia la care se află fiecare element (prin poziţie absolută `SOAP-ENC:position` sau poziţie relativă la ultimul indice `SOAP-ENC:offset`), permiţând codarea optimă a şirurilor rare.

Codarea unor astfel de elemente se face un pic diferit faţă de elementele de tip structură, fiind necesară folosirea unor atribute speciale care să determine tipul şirului şi alte elemente.

Pentru a înţelege mai bine procesul de serializare şi deserializare al unui şir se recomandă analizarea răspunsului SOAP primit în urma cererii precedente. Acesta arată conform listingului 3.2.

Listing 3.2: Raspuns la apelul RPC al metodei cautaPersoanaDupaCriteriu

```
<?xml version="1/0" encoding="UTF-8"?>

<SOAP–ENV:Envelope
   SOAP–ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:SOAP–ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:myns="http://info.unitbv.ro/dovle" >

   <SOAP–ENV:Body>
      <cautaPersoanaDupaCriteriuResponse>
         <cautaPersoanaDupaCriteriuResult xsi:type="Array"
            xsi:arrayType="myns:Persoana[2]" >
            <item xsi:type="myns:Persoana" >
               <nume xsi:type="xsd:string">Muresan</nume>
               <prenume xsi:type="xsd:string">Vasile</nume>
               <adresa xsi:nil="true" />
               <telefon xsi:nil="true" />
            </item>
            <item xsi:type="myns:Persoana" >
               <nume xsi:type="xsd:string">Muresan</nume>
               <prenume xsi:type="xsd:string">Alina</nume>
               <adresa xsi:nil="true" />
               <telefon xsi:nil="true" />
            </item>
         </cautaPersoanaDupaCriteriuResult>
      </cautaPersoanaDupaCriteriuResponse>
   </SOAP–ENV:Body>

</SOAP–ENV:Envelope>
```

Prezenţa atributului `xsi:type` nu este obligatorie în acest caz. Dacă apare, atunci are o valoare predefinită (Array). Prezenţa atributului `xsi:arrayType` este obligatorie şi specifică tipul de bază al şirului. După cum se vede, sintaxa este elocventă.

3.4 Convenţii RPC

Până acum, în acest capitol am prezentat două exemple, un apel RPC şi un răspuns la apelul respectiv, codate cu protocolul SOAP. De asemenea am menţionat că un mesaj SOAP de tip RPC trebuie să îndeplinească anumite condiţii din punctul de vedere al organizării elementului `SOAP-ENV:Body`. Acum este momentul să prezint aceste reguli.

3.4.1 Apelul RPC

Mesajul din listingul 3.1 reprezintă codarea unui apel de procedură la distanţă folosind protocolul SOAP. Acesta solicită rularea procedurii `cautaPersoanaDupaCriteriu( Criteriu criteriu )` pe server, cu parametrul `criteriu` serializat conform mesajului. Apelul efectiv se face prin trimiterea acestui mesaj, prin intermediul protocolului HTTP către un URI la care este conectat serverul, URI ce identifică pentru acel server serviciul ce trebuie accesat.

Cererea RPC este modelată ca o structură încapsulată în corpul mesajului SOAP (elementul `SOAP-ENV:Body`) astfel:

- Numele structurii (primului element conţinut de corpul mesajului) este identic cu numele procedurii de executat pe server.
- Fiecare parametru in şi in-out este modelat ca un element al structurii mesajului (primului element din corpul SOAP).
- Numele elementelor reprezentând parametrii este identic cu numele parametrilor din metoda RPC. De asemenea tipurile trebuie să coincidă (făcând abstracţie de polimorfism)
- Elementele reprezentând parametrii apar în aceeaşi ordine ca şi parametrii din semnătura metodei.

3.4.2 Răspunsul la un apel RPC

Răspunsul la un apel RPC de asemenea trebuie să îndeplinească anumite reguli pentru a fi valid şi pentru a putea fi interpretat corect de o aplicaţie SOAP. În primul rând răspunsul este şi el modelat ca o structură. Aceasta trebuie să aibă numele egal cu numele metodei apelate şi precedat de cuvântul *Response*. După cum se vede şi în listingul 3.2, numele structurii principale este *cautaPersoanaDupaCriteriuResponse*.

De asemenea în această structură există elemente copil pentru tipul de retur al metodei şi de asemenea pentru fiecare parametru in-out şi out al metodei. Primul dintre aceste elemente este întotdeauna rezultatul apelului metodei, următoarele elemente, identificabile după nume reprezentând parametrii de intrare-ieşire şi ieşire. Prin convenţie numele elementului rezultat este acelaşi cu numele operaţiei, cu *Result* adăugat la sfârşit.

Mai multe despre intermediari şi actori, multireferenţiere, tipurile de parametrii (in, in-out şi out), SOAP cu ataşamente şi adaptarea nivelului de transport în capitolul următor ce prezintă elemente SOAP de nivel avansat.

Capitolul 4

Elemente avansate de SOAP

Capitolul precedent a prezentat informaţii de bază legate de protocolul SOAP, cum ar fi apariţia şi avantajele folosirii acestuia, cadrul de împachetare al unui mesaj şi codarea datelor. Acest capitol are ca rol prezentarea unor funcţionalităţi mai delicate ale protocolului SOAP.

Pentru început voi prezenta anumite aspecte referitoare la codarea datelor: multireferenţierea şi tipurile de parametrii admişi. Apoi voi prezenta foarte pe scurt conceptul de transmitere de mesaje SOAP cu ataşamente folosind formatul MIME. Ultima secţiune are ca rol prezentarea alternativelor la folosirea protocolului de transport HTTP pentru transmiterea mesajelor SOAP.

4.1 Multireferenţiere

Modelele abstracte de date permit ca acelaşi obiect să fie referit din mai multe locaţii distincte. Altfel spus, în momentul serializării graful de obiecte să conţină un obiect care să fie referit din mai multe locaţii. De exemplu obiectul `Obj` poate fi referenţiat de trei locaţii diferite ale unui şir de obiecte ce trebuie serializat.

În mod normal, serializatorul va prelua şirul de obiecte şi, rând pe rând va genera cod XML SOAP pentru fiecare dintre obiectele conţinute de şir. Astfel că la sfârşit obiectul `Obj` va fi serializat de trei ori, ca fiind un obiect diferit. Un alt dezavantaj, şi mai important, este că şirul, la deserializare, va avea o altă structură decât cea din momentul serializării, conţinând trei obiecte distincte (ce-i drept egale ca valoare cu obiectul iniţial `Obj`).

Problema pe care trebuie să o rezolve specificaţiile SOAP este codarea obiectului o singură dată şi transmiterea referinţei către acesta de fiecare dată când este accesat pentru serializare. Acest mecanism de referenţiere se bazează pe mecanismul de referenţiere deja existent în limbajul XML, care conţine un două tipuri speciale de atribute ID(identificator) şi HREF (referinţă). Atributele ID sunt folosite pentru a identifica unic elementul ce le conţine. Atributele HREF sunt folosite pentru a referenţia un atribut după ID-ul său.

Iată cum vor fi tratate aceste obiecte multireferenţiate. Fiecărui obiect de acest tip i se atribuie un ID unic, prin intermediul căruia va fi accesat. În momentul în care serializorul trebuie să serializeze un astfel de element, doar introduce un atribut HREF conţinând ID-ul

obiectului, după care trece mai departe. Odată terminată serializarea elementului corp SOAP, se adaugă în urma sa noi elemente, reprezentând forma serializatîî XML SOAP a obiectelor multireferenţiate. În plus, aceste elemente mai conţin un câmp special ID.

Pentru a fi mai clar, vă recomand analizarea mesajului din listingul 4.1.Acest mesaj SOAP conţine un şir de trei persoane, dar fiecare element din şir referenţiind acelaşi obiect.

Listing 4.1: Multireferentierea

```
<?xml version="1/0" encoding="UTF-8"?>

<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:myns="http://info.unitbv.ro/dovle" >

   <SOAP-ENV:Body>
      <cautaPersoanaDupaCriteriuResponse>
         <cautaPersoanaDupaCriteriuResult xsi:type="Array"
            xsi:arrayType="myns:Persoana[2]" >
            <item href="#001" />
                <item href="#001" />
                <item href="#001" />
         </cautaPersoanaDupaCriteriuResult>
      </cautaPersoanaDupaCriteriuResponse>
   </SOAP-ENV:Body>

    <multiref id="#001" xsi:type="myns:Persoana" >
       <nume xsi:type="xsd:string">Muresan</nume>
      <prenume xsi:type="xsd:string">Alina</nume>
      <adresa xsi:nil="true" />
      <telefon xsi:nil="true" />
   </multiref>

</SOAP-ENV:Envelope>
```

Multireferenţierea prezintă anumite avantaje în momentul în care este necesară serializarea unui graf complex de obiecte care sunt de multe ori multireferenţiate, ducând la o reprezentare corectă a informaţiei cât şi la micşorarea cantităţii de date transmisă.

În majoritatea cazurilor multireferenţierea nu este o soluţie viabilă, deoarece necesită mai mult efort de creare a unui astfel de mesaj (memorie, timp de procesare) şi de asemenea necesită ca partea cu care se face comunicarea să suporte această facilitate. De asemenea aplicaţia destinaţie va avea nevoie de o putere de procesare mai mare pentru a putea interpreta astfel de mesaje.

Deşi o componentă puternică, ea trebuie folosită doar atunci când într-adevăr poate aduce avantaje. Totul depinde de problema ce trebuie rezolvată.

4.2 Tipul parametrilor

SOAP defineşte, în afară de tipul de *parametrii de intrare*, alte două tipuri: *parametrii de intrare/ieşire* şi *parametrii de ieşire*. Parametrii de intrare sunt echivalentul transmiterii prin valoare a parametrilor pentru apelul unei operaţii. Parametrii de intrate-ieşire sunt echivalentul parametrilor referinţă transmişi la apelul unei operaţii. Parametrii de ieşire nu au echivalent pentru programarea standard, ei fiind asemănători ca idee cu valoarea returnată de operaţie. O operaţie care are parametrii de ieşire este echivalentă cu o operaţie care returnează mai multe rezultate deodată.

Serviciile Web şi SOAP se deosebesc major de alte produse RPC cum ar fi RMI-IIOP sau CORBA. În acestea transmiterea prin referinţă a unui parametru se făcea efectiv transmiţând o referinţă remote prin intermediul căreia se putea modifica obiectul referenţiat. Din punctul de vedere al SOAP, această idee nu poate fi executată, deoarece se bazează pe o cerere şi pe un răspuns, deci nu pe un interschimb continuu de date.

Modul prin care SOAP rezolvă această problemă (din punct de vedere al specificaţiilor) este elegant. Parametrii de tip intrare-ieşire şi ieşire sunt adăugaţi la elementul structură răspuns ca elemente imediat următoare elementului rezultat. Dacă vă mai aduceţi aminte, rezultatul la un apel RPC era încapsulat astfel: primul element al corpului SOAP îl reprezintă structura răspuns. Primul element al structurii răspuns conţine structura rezultat (codarea rezultatului efectiv al operaţiei). Elementele următoare reprezintă codarea parametrilor intrare-ieşire şi ieşire.

Astfel SOAP delegă responsabilitatea rezolvării acestor probleme de simulare a transmiterii prin referinţă bibliotecii client care trebuie să preia valorile de ieşire şi să seteze valoarea variabilelor referinţă conform mesajului SOAP.

Pentru produsele Java, acest tip de parametrii a reprezentat o problemă destul de complexă deoarece acest limbaj de programare nu permite transmiterea parametrilor prin referinţă şi cu atât mai puţin existenţa mai multor parametrii de ieşire. Această problemă a fost rezolvată prin intermediul unor clase de tip Holder, care conţin o referinţă către valoarea obiectului. Acest procedeu nu este standardizat şi de aceea o parte din aplicaţiile SOAP scrise în Java nu oferă astfel de capabilităţi. Astfel că pentru mai multe informaţii adresaţi-vă documentaţiei oferite de produsul SOAP pe care îl folosiţi.

4.3 SOAP cu ataşamente (SOAP with attachments)

Există diverse situaţii în care mesajele SOAP necesită posibilitatea transmiterii unor ataşamente suplimentare. Protocolul SOAP standard oferă suport pentru transmiterea datelor binare de mari dimensiuni (cum ar fi imagini şamd) prin intermediul codării `SOAP-ENC:base64`. Din păcate această metodă oferă o eficienţă scăzută, necesitând timp de procesare (verifică fiecare caracter spre a nu fi unul specific XML, în acest caz schimbând valoarea acestuia cu o constantă predefinită) şi de asemenea rezultatul ocupând aproximativ cu un sfert mai mult decât dimensiunea iniţială. Deci în cazul transmiterii unor cantităţi mari de informaţie, această soluţie nu este fezabilă.

Din această cauză specialiştii în servicii Web au pus la punct o nouă specificaţie, numită

SOAP with attachments prin intermediul căreia se permite transmiterea mesajului SOAP prin intermediul unei structuri MIME cu mai multe părţi. În această structură, mesajul SOAP propriuzis devine rădăcina mesajului MIME.

Important de specificat faptul că într-un mesaj MIME ataşamentele îşi păstrează formatul nativ, care în majoritatea cazurilor este un format binar.

Formatul MIME a fost dezvoltat iniţial pentru transmiterea mesajelor de mail conţinând ataşamente. La ora actuală este element de bază folosit de SOAP pentru transmiterea cantităţilor mai mari de date.

4.4 Schimbarea nivelului de transport

Protocolul SOAP cuprinde, în specificaţiile sale, mai multe reguli prin intermediul cărora se face o integrare naturală a acestuia cu protocolul de comunicare HTTP. Dar SOAP nu este legat de acest protocol. Din contră, specificaţiile menţionează faptul că programatorul poate să utilizeze o varietate mare de protocoale, printre care SMTP, FTP, HTTPS şamd astfel încât să poată obţine rezultatele dorite.

4.4.1 SOAP pe SMTP

E-mailul este larg răspândit pe Internet. Protocoalele importante în legătură cu e-mailul sunt *SMTP (Simple Mail Transfer Protocol)*, *POP (Post Office Protocol)* şi *IMAP (Internet Message Access Protocol)*.

E-mailul este o unealtă extrem de puternică pentru transferul mesajelor SOAP, atunci când nu este cerută rularea sincronă a acestora. Răspândirea sa, existenţa anteturilor, mecanisme de ataşamente, buffere şi cozi pentru mesaje sunt doar câteva dintre motivele pentru care SMTP (şi tehnologia mail) sunt o alternativă viabilă a protocolului HTTP.

4.4.2 SOAP pe HTTP/S

SOAP specifică faptul că protocolul de transport preferat este HTTP. Dar în multe situaţii când este necesar un nivel mai ridicat de securitate se impune folosirea unor tehnologii auxiliare mai puternice, care să permită criptarea şi securizarea unui canal de comunicaţii. De aceea o menţiune foarte importantă o reprezintă posibilitatea de a configura bibliotecile SOAP (client şi server) pentru a folosi versiunea securizată a protocolului HTTP, numită HTTP/S.

Prin intermediul aceste versiuni, clientului i se vor cere informaţii suplimentare de autentificare (nume utilizator şi parolă) pentru a i se permite accesul la URI-ul ce reprezintă serviciul Web. Aceste informaţii pot fi uşor transferate chiar prin intermediul SOAP, folosind zona antet (headers).

4.4.3 Alte protocoale

În funcţie de natura problemei ce trebuie rezolvată, s-a ajuns la unele soluţii extrem de interesante. De exemplu protocolul FTP (File Transfer Protocol) se dovedeşte de asemenea o soluţie fiabilă, simulând transferul de mesaje SOAP asincrone prin intermediul funcţiei de transfer de fişiere.

De asemenea infrastructuri de transfer de mesaje sofisticate, precum IBM MQSeries, Microsoft Message Queue (MSMQ) şi Java Messaging Service (JMS) sunt potrivite pentru transportul mesajelor SOAP.

Capitolul 5

WSDL

5.1 Interoperabilitate si WSDL

Unul dintre cele mai importante avantaje ale protocolului SOAP îl reprezintă nivelul ridicat de interoperabilitate. Deşi un mit, acest lucru a început să devină tot mai probabil odată cu adoptarea pe scară largă a protocoalelor bazate pe XML. Odată cu publicarea specificaţiilor oficiale, aceste protocoale sunt cu un pas mai aproape de a-şi atinge scopurile.

Interoperabilitatea reprezintă capacitatea aplicaţiilor alfate pe platforme diferite, pe sisteme de operare diferite şi programate în limbaje diferite de a interschimba informaţii. Ea este îngreunată deobicei de probleme precum nepotrivirea codării caracterelor de la o limbă la alta, reprezentarea diferită a unor tipuri standard (de ex. pe unele platforme `int` este reprezentat pe 16 biţi, pe alte platforme pe 32 de biţi) şamd. Din cele enunţate înainte o codare binară standard este destul de greu de definit pentru a reuşi să reunească toate caracteristicile mai multor limbaje de programare diferite. De aceea singura soluţie viabilă este folosirea limbajului XML.

Deşi specificaţiile, implementările corecte de biblioteci SOAP precum şi alţi factori independenţi de programator influenţează caracterul de interoperabilitate, important de ştiut este că şi programatorul joacă un anume rol în sporirea gradului de interoperabilitate al unui serviciu Web. El poate urma anumite procedee standard de implementare şi elemente recomandate de organizaţii internaţionale (precum WS-I). De asemenea păstrarea cât mai generală a implementării, fără a folosi tipuri speciale ale motorului SOAP folosit, cât şi modularizarea şi documentarea serviciului pot face din acesta o soluţie viabilă pentru mulţi potenţiali clienţi.

De ce este necesară documentarea serviciului? Deoarece serviciul este oferit spre uz întregii comunităţi Internet (sau unui subset). Există programatori care vor să se folosească de funcţionalitatea oferită de serviciul Web în cauză. Dar sarcina lor este foarte dificilă în cazul lipsei documentaţiei referitoare la interfaţa oferită de serviciu şi la mapările ce trebuie folosite pentru a putea transmite corect obiecte şi alte informaţii.

Din această cauză au luat naştere mai multe limbaje de documentare a unui serviciu Web. Unele dintre ele se axează pe organizarea şi accesarea serviciilor, altele pe topografia unui serviciu Web mai complex iar altele pur şi simplu descriu ce face fiecare metodă şi cum

ar trebui apelată. Dintre toate aceste limbaje se remarcă, datorită folosirii sale pe scară largă, limbajul *WSDL (Web Services Description Language)*. Submis pentru aprobare de către IBM, Microsoft şi alţii în septembrie 2000, acest protocol are ca rol descrierea tehnică elementară a interfeţei serviciului. O descriere WSDL prezintă trei proprietăţi fundamentale ale unui serviciu Web:

- *Ceea ce face* serviciul – operaţiile pe care serviciul le furnizează.
- *Cum este accesat* serviciul – detalii ale formatelor de date şi protocoale folosite pentru accesarea operaţiilor.
- *Unde este localizat* serviciul – detalii ale adresei de reţea specifice protocolului (un URL).

Trebuie să menţionez aici că un document WSDL ce descrie un serviciu este destul de impresionant, ca dimensiuni cât şi ca varietate de etichete, spaţii de nume şamd. Deşi este bazat tot pe limbajul XML, citirea lui poate fi destul de greoaie. De aceea următorul subcapitol va prezenta organizarea unui astfel de document şi însemnătatea fiecărei etichete. Având în vedere nivelul înalt de normalizare al unui astfel de document, odată ce aţi înţeles elementele componente, va fi uşor de urmărit, pas cu pas, funcţionalitatea oferită de un serviciu Web descris cu ajutorul acestui limbaj.

5.2 Structura unui document WSDL

Un document WSDL este compus din următoarele elemente: PortType, Operation, Message, Type, Binding, Port, Service. Pentru a înţelege mai bine organizarea efectivă şi ce reprezintă fiecare din componentele acestei lungi liste, voi prezenta un exemplu de document WSDL. Acest document conţine informaţii referitoare la un serviciu simplu, cu o singură clasă şi o operaţie definite. Acesta se va numi *Telesoap*. Clasa se va numi *Persoana* şi va încapsula informaţii despre o persoană (nume, prenume, număr de telefon, adresă). Operaţia expusă de serviciu are următoarea semnătură: `Persoana cautaPersoana( String nume, String prenume )`. Iată cum arată documentul WSDL:

Listing 5.1: Exemplu de fisier WSDL

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3    targetNamespace="http://localhost:8080/axis/services/Telesoap"
4    xmlns="http://schemas.xmlsoap.org/wsdl/"
5    xmlns:intf="http://localhost:8080/axis/services/Telesoap"
6    xmlns:tns1="http://info.unitbv.ro/AODovlecel"
7    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
8    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
9    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
10
11    <!-- Descrierea tipurilor -->
```

```
12    <wsdl:types>
13       <schema targetNamespace="http://info.unitbv.ro/sample"
14          xmlns="http://www.w3.org/2001/XMLSchema">
15
16          <complexType name="Persoana">
17             <sequence>
18                <element name="nume" type="xsd:string"/>
19                <element name="prenume" type="xsd:string"/>
20                <element name="adresa" type="xsd:string"/>
21                <element name="telefon" type="xsd:integer"/>
22             </sequence>
23          </complexType>
24       </schema>
25    </wsdl:types>
26
27    <!-- descrierea mesajelor -->
28    <wsdl:message name="cautaTelefonRequest">
29       <wsdl:part name="nume" type="xsd:string"/>
30       <wsdl:part name="prenume" type="xsd:string"/>
31    </wsdl:message>
32
33    <wsdl:message name="cautaTelefonResponse">
34       <wsdl:part name="cautaTelefonReturn" type="tns1:Persoana"/>
35    </wsdl:message>
36
37    <!-- Definitii de tip Port -->
38    <wsdl:portType name="TeleSoapWS">
39       <wsdl:operation name="cautaTelefon"
40          parameterOrder="nume,prenume">
41          <wsdl:input message="intf:cautaTelefonRequest"
42             name="cautaTelefonRequest"/>
43          <wsdl:output message="intf:cautaTelefonResponse"
44             name="cautaTelefonResponse"/>
45       </wsdl:operation>
46    </wsdl:portType>
47
48    <!-- Definitii de legatura -->
49    <wsdl:binding name="TelesoapSoapBinding" type="intf:TeleSoapWS">
50       <soap:binding style="rpc"
51          transport="http://schemas.xmlsoap.org/soap/http"/>
52
53       <wsdl:operation name="cautaTelefon">
54          <soap:operation soapAction=""/>
55          <wsdl:input name="cautaTelefonRequest">
56             <soap:body
57             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
58             namespace="http://localhost:8080/axis/services/Telesoap"
```

```
59              use="encoded"/>
60          </wsdl:input>
61          <wsdl:output name="cautaTelefonResponse">
62              <soap:body
63              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
64              namespace="http://localhost:8080/axis/services/Telesoap"
65              use="encoded"/>
66          </wsdl:output>
67      </wsdl:operation>
68  </wsdl:binding>
69
70  <!-- Definitii de serivicii -->
71  <wsdl:service name="TeleSoapWSService">
72      <wsdl:port binding="intf:TelesoapSoapBinding" name="Telesoap">
73          <soap:address
74              location="http://localhost:8080/axis/services/Telesoap"/>
75      </wsdl:port>
76  </wsdl:service>
77
78 </wsdl:definitions>
```

După cum se vede, chiar şi pentru un serviciu de dimensiuni reduse un astfel de fişier de descriere are dimensiuni impresionante. Următoarele secţiuni vor prezenta pe scurt ce reprezintă fiecare parte a documentului WSDL şi le va exemplifica, pe baza exemplului din listingul B.1. Ordinea în care vor fi prezentate elementele nu este cea de apariţie ci una prin care să prezinte informaţia de la definirea interfeţei (`portType`) până la legarea de implementare (`binding`) şi localizarea serviciului (`service`).

5.2.1 Elementele PortType

Cel mai bun punct de pornire pentru înţelegerea unui serviciu Web descris printr-un document WSDL este elementul `portType`. Acesta descrie interfaţa serviciului . Aceasta este cea mai concisă descriere a *ceea ce face* serviciul. Restul elementelor din definiţia WSDL reprezintă detalii de care depinde `portType` şi vor fi examinate în secţiunile următoare.

Iată cum arată acest element din documentul prezentat în listingul B.1 între liniile 38 şi 46:

Listing 5.2: Elementul portType

```
1 <wsdl:portType name="TeleSoapWS">
2     <wsdl:operation name="cautaTelefon"
3         parameterOrder="nume,prenume">
4         <wsdl:input message="intf:cautaTelefonRequest"
5             name="cautaTelefonRequest"/>
6         <wsdl:output message="intf:cautaTelefonResponse"
7             name="cautaTelefonResponse"/>
8     </wsdl:operation>
9 </wsdl:portType>
```

Un document WSDL poate conţine zero sau mai multe definiţii `portType`. Prezenţa mai multor elemente este echivalentă cu faptul că serviciul Web exportă o funcţionalitate exprimată de mai multe interfeţe.

Un element `portType` conţine un singur atribut `name`. Acesta defineşte numele prin care va fi referită interfaţa în acest document. De aceea el trebuie să fie diferit de numele altor elemente `portType` aflate în WSDL. Numele acestor elemente respectă în general un şablon: *numeServiciuWeb*`WS` sau *numeServiciuWeb*`PortType`.

Elementul `PortType` conţine mai multe elemente `operation` prin intermediul cărora se defineşte inferfaţa portului respectiv.

5.2.2 Elementele Operation

Un element `operation` în WSDL este echivalent cu semnătura unei metode în Java. O operaţie defineşte o metodă pe un serviciu Web, inclusiv numele metodei şi parametrii de intrare, precum şi tipul de returnare al metodei şi, dacă este necesar, erorile (fault) ce pot fi generate.

Conform listingului 5.2, interfaţa `TeleSoapWS` defineşte o singură operaţie (metodă) denumită `cautaTelefon`. Aceasta este de tip cerere-răspuns deoarece primul element copil este de tip `input` şi este urmat de un element de tip `output`. Prin intermediul acestor elemente pot fi definite patru tipuri de apel:

- operaţie cerere-răspuns(primul element este de intrare, al doilea de ieşire), echivalentul unui apel de metodă
- operaţie cu sens unic (doar elementul de intrare), echivalentul unei operaţii care transmite date şi nu necesită răspuns
- operaţie de notificare (doar elementul de ieşire), echivalentul notificării clientului de către server că un eveniment a avut loc
- operaţie de solicitare răspuns (elementul de ieşire urmat de cel de intrare) echivalentul cererii serverului de a primi un răspuns de la client.

În starea curentă a uneltelor WSDL se pune cu adevărat accent pe mesajele cu sens unic şi cerere-răspuns. Mesajele de tip notificare şi solicitare răspuns cer un anumit tip de corelare, fiind necesare cunoaşterea în prealabil a anumitor elemente pentru ca serverul să poată iniţia comunicarea cu clientul (adresa şi portul la care acesta poate fi găsit şamd).

Cele două elemente copil ale elementului `operation` nu definesc efectiv semnătura intrării sau ieşirii ci referă anumite mesaje care reprezintă, după cum vom vedea, semnătura de intrare şi cea de ieşire a metodei.

5.2.3 Elementele Message

Un mesaj reprezintă un concept foarte simplu. Un mesaj este o colecţie de părţi. Un document WSDL poate conţine zero sau mai multe elemente `message`, acestea fiind plasate

ca şi copii ai elementului rădăcină. În cazul documentului prezentat în listingul B.1, sunt definite două mesaje, unul referit de intrarea operaţiei `cautaTelefon`, altul referit de aceeaşi operaţie prin intermediul elementului de ieşire. Acestea sunt definite între liniile 28 şi 35.

Să luăm cazul elementului `cautaTelefonRequest`. El este definit astfel:

Listing 5.3: Elementul message

```
1 <wsdl:message name="cautaTelefonRequest">
2     <wsdl:part name="nume" type="xsd:string"/>
3    <wsdl:part name="prenume" type="xsd:string"/>
4 </wsdl:message>
```

Un element `message` conţine obligatoriu un atribut `name`. În acest caz el are valoarea `cautaTelefonRequest`. Numele fiecărui element `message` trebuie să fie unic în interiorul documentului WSDL deoarece el va fi identificatorul prin care va fi referit mesajul.

Acest mesaj conţine două părţi. Prima are numele `nume` şi este de tip `string`. A doua parte are numele `prenume` şi este de asemenea de tip `string`. O parte (part) este formată din două proprietăţi: numele părţii şi tipul acesteia. Atributul `name` trebuie să fie unic între toate elementele copil ale mesajului. Proprietatea de tip este definită ca un atribut `type`, acesta luând valori specifice XML Schema sau valori specifice aplicaţiei definite în secţiunea de tipuri.

5.2.4 Elementele Type

Cu ajutorul secţiunii `types` se definesc tipuri noi ce vor fi folosite de serviciul Web. În cazul nostru este vorba despre entitatea `Persoana`. Structura acestor tipuri este definită cu ajutorul limbajului XML Schema şi este introdusă cu ajutorul elementului `complexType`. Un exemplu de astfel de definire poate fi studiat în listingul B.1, între liniile 16 şi 22.

Pentru a înţelege modul de definire, se recomandă parcurgerea documentaţiei referitoare la limbajul de definire şi validare XML Schema.

5.2.5 Elementele Binding (legătură)

Serviciul are definite interfeţele pe care le expune (cu ajutorul elementelor `portType`), dar nu specifică pe moment nimic despre modul în care aplicaţia client poate să acceseze aceste metode. Acesta este scopul elementului legătură (binding): la o interfaţă ataşează informaţii referitoare la protocolul de transport (HTTP, FTP, SMTP şamd), stilul apelului (rpc sau document) şi specifică, pe rând, pentru fiecare operaţie conţinută de portul legat, cum se face codarea (encoded), spaţiul de nume ce trebuie folosit şi modul de codare (encodingStyle).

Un element de tip legătură conţine obligatoriu un atribut `name`. Acesta deobicei este de forma `<portName><protocol>Binding` şi trebuie să fie unic în documentul WSDL. În cazul exemplului din listingul B.1, numele elementului legătură este `TelesoapSoapBinding`. Acest lucru ne spune faptul că accesarea tipului de port legat p(referit prin intermediul atributului obligatoriu `type`) se face folosind protocolul SOAP. Acest lucru poate fi confirmat şi de prefixul elementului `soap:binding` conţinut de elementul legătură (linia 50).

Prin intermediul elementului `soap:binding` se specifică nivelul de transport folosit (în cazul nostru este vorba de HTTP, referit prin folosirea atributului `transport`) şi modul în care se face apelul (definit prin intermediul atributului `style`, care poate avea valorile `rpc` sau `document`).

Urmează ca elementul legătură să precizeze informaţii de acces pentru fiecare din operaţiile definite de portul legat. În acest caz fiind vorba despre o singură operaţie, conţine doar un element `wsdl:operation` (linia 53).

Acum, în câteva cuvinte, ce specifică elementul `wsdl:operation`. În primul rând defineşte valoarea atributului SOAPAction (linia 54) pentru a descrie intenţia clientului în apelul metodei respective. Apoi se specifică formatarea mesajului de intrare pentru operaţie (liniile 55–60) şi de asemenea formatarea mesajului de ieşire (liniile 61–66).

5.2.6 Elementele Port

Elementul `port` are ca scop specificarea adresei de reţea a obiectivului ce găzduieşte serviciul Web. Mai exact asociază o adresă unică (specifică protocolului folosit) la un element `binding`. Elementele `port` sunt denumite şi trebuie să fie unice între toate porturile definite de documentul XML.

Elementul `port` pentru listingul B.1 poate fi analizat între liniile 72 şi 75.

5.2.7 Elementele Service

Scopul elementului `service` este de a conţine un set de elemente `port` înrudite. Nimic mai mult. Pare totuşi un mare deranj să risipeşti toate aceste elemente pentru a descrie un grup de porturi. De ce ar grupa un designer câteva porturi în cadrul unul element `service`?

Un motiv este de a grupa porturile legate de interfaţa aceluiaşi serviciu (portType) dar exprimate prin protocoale diferite (prin intermediul legăturilor). De exemplu o interfaţă poate fi legată de folosirea protocolului HTTP prin intermediul unui element binding, şi aceeaşi interfaţă să fie legată de folosirea protocolului SMTP de un alt element binding. În acest caz aceste două legări pot fi grupate într-un singur element serviciu.

Aceasta a fost mulţimea elementelor de bază ce apar într-un document WSDL. Deşi destul de bogată, ea este organizată natural, prin prisma relaţiilor dintre componentele ei. Astfel, odată înţeles modul de organizare, programatorul poate înţelege fără prea multe probleme ideile de bază ale unui serviciu Web (interfaţa de comunicare, codarea folosită, clasele de date suplimentare şamd).

5.3 Folosirea documentului WSDL

Un document WSDL descrie amănunţit interfaţa oferită de un serviciu Web şi modul în care aplicaţia client ar trebui să acceseze serviciul. Pentru început el poate fi folosit de programatorul aplicaţiei client spre a înţelege interfaţa client şi clasele de date folosite.

Caracterul normalizat al unui astfel de document şi faptul că respectă anumite reguli publice (specificaţiile WSDL) au favorizat dezvoltarea unor utilitare extrem de practice, care sunt capabile să genereze clasele de comunicare client(*stub*) şi server(*skeleton*) pe baza acestor informaţii.

Spre exemplu AXIS oferă programatorilor aplicaţia *WSDL2Java* care are ca rol preluarea unui document WSDL ce descrie un serviciu Web şi crearea claselor de date şi a claselor de comunicare prin intermediul cărora aplicaţia client va efectua apelurile la distanţă. Aceste clase de comunicare publică interfaţa serviciului Web şi ascund nivelul de transport (serializarea elementelor, efectuarea apelului prin protocolul de transport, primirea răspunsului şamd). Astfel programatorul va avea senzaţia că serviciul Web este ca un obiect ce aparţine de aplicaţia sa.

Astfel de utilitare sunt tot mai des întâlnite în lumea IT. Ele sunt oferite (deobicei) ca accesorii auxiliare pe lângă motorul SOAP. După cum am menţionat, Apache AXIS dispune de un astfel de utilitar. De asemenea mediul integrat de dezvoltare Visual Studio .NET oferă această funcţionalitate. În ultimul timp puterea acestor tipuri de programe a crescut, astfel că generează cod optim în timp foarte scurt, eliminând perioada de dezvoltare a nivelului de comunicare de către programatori. Deci, sfatul meu este să folosiţi aceste utilitare oricând se iveşte ocazia deoarece scurtează semnificativ timpul de dezvoltare al unei aplicaţii client SOAP.

Capitolul 6

UDDI

Până acum am văzut, pe scurt, cum se poate organiza un serviciu Web şi cum pot fi transmise datele între parteneri de comunicare (client şi server, sau de la egal la egal) prin intermediul protocolului SOAP. De asemenea am prezentat şi modul de descriere al unui serviciu Web prin intermediul fişierelor WSDL, cu ajutorul elementelor de SOAP şi XML Schema. Se poate afirma că aceste elemente sunt arhisuficiente unui programator pentru a dezvolta un serviciu Web uşor accesibil. Acest lucru este complet adevărat.

Dar din păcate nu trăim într-o lume ideală. Oricât de bun ar fi serviciul pe care îl oferim, acesta poate rămâne în anonimat dacă nu este prezentat într-un mod adecvat spre folosire. Altfel zis, în limbaj de marketing: publicitatea este cheia. Pentru a fi folosit, serviciul trebuie să fie cunoscut, iar pentru asta este nevoie de o tehnologie de prezentare: regiştrii UDDI.

6.1 Scurt istoric

Problema care se pune este cum se poate publica un serviciu Web, cum poate fi făcut cunoscut. Soluţiile sunt variate. Iată câteva dintre acestea:

- Producătorul serviciului contactează clientul direct (mail, fax) şi transmite informaţiile referitoare la funcţionalitatea oferită de produsul implementat. Aceasta este cea mai simplă abordare, dar presupune ca producătorul să aibă cunoştinţă de toti clienţii interesaţi de folosirea serviciului. Din păcate, lucru greu de realizat în cele mai multe cazuri.

- Publicarea şi promovarea descrierilor de servicii Web pe un site Web specializat în acest proces. În acest caz, site-ul Web acţionează ca un registru de cservicii iar operaţia de publicare este doar un act de prezentare Web ce poate fi accesat cu un simplu browser de HTML. Un pas înainte este făcut de produsele *IBM ADS* sau *Microsoft .NET DISCO*. Acestea oferă îmbunătăţiri în această direcţie.

- Adăugarea documentului de descriere într-un deposit de documente WSDL. Această soluţie este asemănătoare cu cea precedentă, doar că oferă posibilitatea ca registrul să anunţe clienţii în eventualitatea modificării descrierii serviciului Web.

- Folosirea *UDDI (Universal Description Discovery and Integration).* Aceasta este cea mai sofisticată abordare a registrelor de servicii. UDDI include mult mai multă informaţie decât descrierea de serviciu, incluzând metadate de afaceri. UDDI nu numai că răspunde întrebării 'unde este localizat serviciul' ci abordează şi întrebări de genul 'Cum ştie solicitantul ce afacere furnizează serviciul'.

Odată cu dezvoltarea programării orientate pe obiecte şi pe componente, a devenit evidentă necesitatea unor regiştrii care să păstreze informaţii despre aceste elemente, despre modul cum pot fi găsite şi utilizate. La începutul anului 2000, pe măsură ce ideea de serviciu Web a început să prindă contur, a devenit clar că regiştrii de servicii Web urmau să fie esenţiali pentru ca tot acest concept să devină practic. Având în vedere natura publică (standarde deschise) ce ţine de tehnologia serviciilor Web, era imposibil ca acest protocol să fie dezvoltat altfel.

Astfel, ca urmare a colaborării dintre reprezentanţi ai firmelor Ariba, IBM şi Microsoft, a apărut şi a fost anunţată formal dezvoltarea acestui standard pe data de 6 septembrie 2000. Suportul pentru această tehnologie s-a extins dincolo de cele trei companii fondatoare, la momentul actual implicând o comunitate de peste 310 companii.

Obiectivul UDDI este de a facilita descoperirea de servicii Web atât pentru proiectare cât şi în mod dinamic, la execuţie. În consecinţă proiectul UDDI menţine un registru de afacere publică online (împreună cu serviciile corespondente), denumit *UDDI Business Registry.* Acesta a intrat în funcţiune pe data de 2 mai 2001 şi de fapt conţine doi regiştri replicaţi ce sunt găzduiţi de două companii: IBM şi Microsoft (denumiţi operatori UDDI). Este posibil ca mai alţi operatori (precum Hewlet-Packard) să fi lansat proprii regiştrii până în momentul de faţă.

6.2 Ce este UDDI

UDDI este mai mult decât un registru de servicii şi de afaceri. El defineşte şi un se de structuri de date şi o specificaţie API pentru a înregistra şi a găsi afaceri, servicii, legături şi tipuri de servicii, prin program.

Specificaţia API pentru UDDI oferă un set de API-uri pentru publicare pentru a înregistra servicii şi API-uri de interogare pentru a găsi servicii.

În plus faţă de furnizarea unui API programabil, operatorii de registru UDDI oferă, fiecare, o interfaţă de utilizator bazată pe pagini Web prin intermediul căreia se pot executa acţiuni standard de înregistrare, gestionare şi găsire de afaceri şi servicii.

6.3 Folosirea UDDI

După cum am menţionat în paragrafele anterioare, funcţionalitatea UDDI poate fi accesată prin intermediul unei interfeţe Web oferite de operatorii UDDI. Un exemplu bun îl constituie site-urile `http://www.ibm.com/services/uddi` şi `http://uddi.microsoft.com`.

Pentru a putea lucra optim cu aceşti regiştri, trebuie ştiut faptul că ei prezintă câteva cerinţe:

- Cerinţe de bază
  - Un set de specificaţii de structură a datelor, pentru ca metadatele să fie stocate în registru.
  - Un set de specificaţii ale operaţiilor Create, Read, Update, Delete, pentru stocare, ştergere şi interogare a datelor din registru.
- Cerinţe comune pentru metadate
  - drept de proprietate şi conţinere.
  - categorisire. Datele pot fi plasate în una sau mai multe categorii, facilitând operaţia de căutare şi interogare.
- Cerinţe comune pentru operaţii
  - autentificare pentru operaţii de schimbare a informaţiei şi pentru regiştrii publici.
  - acces deschis pentru operaţiile de citire şi interogare.

Partea II

Apache AXIS

Capitolul 1

Despre AXIS

1.1 Scurt istoric

Organizaţia Apache este unul dintre cei mai importanţi producători de soft gratuit. Ea a devenit foarte cunoscută datorită produsului lor forte, serverul de web Apache. Incurajată de acest succes, a continuat dezvoltarea în mai multe direcţii, astfel materializându-se şi proiectul XML. Acesta are ca scop implementarea de componente java care să ajute la integrarea limbajului XML în lumea programării distribuite.

Odată cu apariţia specificaţiilor SOAP referitoare la serviciile Web, proiectului XML i s-a mai adăugat o nouă componentă, denumită *Apache Soap*. Este vorba despre primul server SOAP scris in Java, implementat sub forma unei aplicatii web.

Cu trecerea timpului însă au început să apară tot felul de probleme, datorate în mod special schimbării frecvente a specificaţiilor cât şi unei proiectări defectuase. Din această cauză adaugarea de noi facilităţi necesita multă muncă de programare, incluzând schimbări de structură în arhitectura serverului. Din acest motiv s-a renunţat la dezvoltarea de noi funcţionalităţi pentru acest proiect şi s-a creat o nouă echipă ce avea să înceapă proiectarea unui server de SOAP mai performant: *Apache AXIS*.

În momentul de faţă, serverul Apache AXIS se află la începuturi, versiunea 1.0. Acest document va face referire la această versiune în încercarea de a descrie modul de funcţionare, organizarea şi, pe cât posibil, detaliile tehnice şi alte elemente din bucătăria internă.

1.2 Avantajele folosirii serverului AXIS

AXIS este urmaşul de drept al lui Apache SOAP. Echipa de programatori care a contribuit la dezvoltarea AXIS s-a folosit de experienţa acumulată de la proiectul anterior. AXIS aduce următoarele funcţionalităţi de bază, care îl fac unul dintre cele mai puternice servere de SOAP cu suport de Java:

- *Viteza*. Spre deosebire de predecesorul său, AXIS foloseste tehnologia SAX pentru a parsa mesajele, rezultând o viteză mai mare de procesare a documentelor XML primite

cât şi consum redus de memorie.

- *Flexibilitate.* Acest lucru este realizat printr-un design care oferă puncte de extensie pentru programator. Astfel se pot adăuga foarte uşor module de transport noi, serializori şi deserializori noi şamd.

- *Stabilitate.* AXIS defineşte un set de interfeţe pe care programatorul se poate baza, acestea schimbându-se relativ lent (în comparaţie cu ritmul în care evoluează specificaţiile SOAP).

- *Obiect orientarea componentelor.* AXIS introduce un nou concept pentru serverele SOAP: handlerele. Acestea sunt clase speciale, care pot fi scrise de programator şi care execută anumite acţiuni asupra câmpurilor unui mesaj (antetul, corpul şamd) înainte de a fi trimis la serviciul Web pentru procesare. Aceste componente sunt oriect orientate şi extind o anumită interfaţă, acest lucru ducând la o organizare mai bună a funcţionalităţii cât şi la reutilizarea şi distribuirea acestor obiecte în proiecte noi.

- *Modulul de transport.* Acesta este realizat cât mai simplu cu putinţă, permiţând adăugarea unor noi tipuri de transport la cele existente într-o manieră naturală. În acest moment AXIS vine cu o colecţie stabilă de nivele de transport care includ comunicarea prin: SMTP, FTP, HTTP.

- *Nucleul motorului de procesare* este independent de modulul de transport, fiind responsabil doar cu serializarea obiectelor în format SOAP-XML şi invers.

- *Suport pentru WSDL.* AXIS suportă WSDL 1.1 permitând crearea unor componente client pentru comunicarea cu servicii externe cât şi posibilitatea de a genera fişiere wsdl pentru a descrie serviciile Web proprii.

În momentul de faţă AXIS se apropie cu paşi repezi de maturitate. Dar el nu este doar un motor SOAP. El mai vine şi cu alte componente importante pentru un programator de servicii Web:

- *Un server independent.* Cu AXIS nu mai este nevoie de instalarea unui servlet container (cum este jakarta-tomcat, produs tot de Apache), serviciile putând rula direct pe serverul inclus în distribuţia de AXIS. Această soluţie este folosită îndeosebi în faza de testare şi dezvoltare a unui serviciu Web şi de asemenea în dezvoltarea de noi funcţionalităţi pentru serverul de SOAP.

- *Un server care se conectează la un server de servleturi.* Acesta se instalează ca o aplicaţie Web pe un server (servlet container) cum ar fi Tomcat.

- *Suport sporit pentru WSDL.* Include generarea automată a documentului wsdl pentru orice serviciu instalat pe axis. De asemenea un utilitar foarte folosit este wsdl2java care preia un document wsdl şi creează o structura de clase care pot fi folosite pentru a apela uşor serviciul ataşat documentului.

- *Exemple.* Acestea arată cum trebuie realizat un serviciu şi de asemenea prezintă anumite tehnologii care pot fi folosite în procesul dezvoltării unui serviciu Web.

- *Utilitar de supraveghere a pachetelor TCP/IP.* Acesta poate fi folosit de un programator de servicii Web pentru a verifica o cerere către serviciu şi răspunsul acestuia. Este *foarte util* în rezolvarea problemelor de interoperabilitate, în faza de dezvoltare a serviciului Web.
- *Documentaţie.* Acesta este un punct forte în comparaţie cu predecesorul său produs tot de Apache. Include un ghid de utilizare, ghid pentru extinderea funcţionalităţii, ghid de administrare şi de integrare în diverse servere şi multe altele.

Şi acum aceasta este gama de facilităţi cu care vine această versiune de Apache AXIS:

- Motor de SOAP compatibil cu specificaţiile SOAP 1.1 şi de asemenea cu o parte din funcţionalitatea cerută de specificaţiile SOAP 1.2
- Sistem flexibil de configurare şi instalare a serviciilor Web
- Suport pentru serializarea şi deserializarea tipurilor de bază şi a colecţiilor (prin trecerea în şiruri)
- Suport pentru serializarea şi deserializarea automată a claselor de tip JavaBean.
- Suport pentru definire de noi serializori şi deserializori.
- Generare automată a fişierului wsdl pentru serviciile instalate.
- Utilitarul *WSDL2Java* pentru a construi nivelul de transport al aplicaţiei client, având la dispoziţie documentul wsdl al serviciului Web.
- Utilitarul *Java2WSDL* pentru a construi documentul wsdl ataşat unui serviciu
- Puncte de extensie pentru tehnologii de securitate
- Suport pentru Conexiuni orientate pe sesiune, folosind cookies în headerul HTTP sau headere SOAP (independente de nivelul de transport).
- Suport incipient pentru SOAP with Attachments.
- Un EJB Provider pentru a accesa EJB session beans ca pe nişte servicii Web.
- Transport HTTP pentru SOAP

După cum am menţionat în partea I, scopul principal al serviciilor Web este de a obţine o cât mai bună interoperabilitate între diverse tipuri de servere şi clienti, motiv pentru care s-a elaborat protocolul SOAP, care se bazează pe XML. De exemplu următoarea configuraţie este viabilă: serverul SOAP să fie AXIS. În acest caz serviciul Web este implementat folosind java. Clientul, în schimb, poate fi implementat în orice limbaj, atât timp cât foloseşte o bibliotecă SOAP. Astfel putem să implementăm clientul folosind kSOAP, wSOAP, Apache SOAP (dacă dorim să rămânem la java), cu ajutorul .NET (care oferă o multitudine de limbaje), cu librării pentru Phyton, PHP şi lista poate continua.

În majoritatea cazurilor este de preferat ca programul client să se folosească de biblioteca folosită de programul server pentru a evita probleme de comunicare datorate implementărilor diferite. Astfel de probleme apar îndeosebi la transmiterea colecţiilor de diverse tipuri, fiecare limbaj având clasele sale particulare. De asemenea multireferenţierea poate cauza probleme, nefiind suportată de toate tipurile de biblioteci SOAP client. Astfel produsul Apache AXIS oferă suport pentru a fi folosit şi cu rol de client în efectuarea apelurilor de proceduri la distanţă.

Această parte a lucrării va începe prin a detalia instalarea şi folosirea programului AXIS ca server. Apoi va prezenta modul de implementare şi desfăşurare a unui serviciu Web. Pentru a înţelege mai bine modul de utilizare, explicaţiile vor fi însoţite de un exemplu. Acesta va fi dezvoltat pas cu pas, pentru a înţelege raţionamentele şi modul în care sunt duse la îndeplinire. Următorul pas este reprezentat de implementarea clientului. Acesta va fi implementat tot cu ajutorul librăriei oferite de AXIS, asigurând în acest fel interoperabilitatea.

Capitolul 2

Instalarea şi testarea serverului AXIS

Aceasta este prima etapă ce trebuie îndeplinită pentru a putea începe dezvoltarea de servicii Web: instalarea unui server de SOAP împreună cu toate componentele necesare pentru o bună rulare şi de asemenea testarea de bază a acestuia.

Acest capitol începe prin descrierea instalării serverului jakarta tomcat, versiunea 4.1.17, folosit pe parcursul acestei lucrări pentru rularea şi testarea serverului de SOAP Apache AXIS. Pasul următor îl reprezintă instalarea propriuzisă a serverului AXIS ca aplicatie Web a serverului Tomcat. Testarea va face subiectul următorului subcapitol, realizată cu ajutorul unui serviciu Web simplu. Ultimul subcapitol va prezenta o listă de probleme ce ar putea să apară la instalarea acestor produse şi sfaturi referitoare la plasarea şi încărcarea programelor.

2.1 Instalarea serverului Jakarta Tomcat

După cum am menţionat în paragrafele precedente, această lucrare a necesitat folosirea serverului Tomcat produs de Apache pentru a putea să ruleze AXIS. Aceasta nu este o cerintă obligatorie, AXIS putând rula pe orice servlet container care respectă specificatiile *servlet 2.2*.

Înainte de a instala serverul tomcat, trebuie să avem instalat pe sistem un mediu de rulare (runtime environment) java. De asemenea este recomandată setarea variabilei sistem JAVA_HOME.

Pentru a instala serverul tomcat, trebuie urmaţi următorii paşi:

Pasul 1) Obţinerea unei distributii binare a produsului. Aceasta poate fi descărcată de la urmatoarea adresă de Internet: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/ Se poate alege între diversele versiuni publicate. Pentru această lucrare s-a folosit versiunii 4.1.17, descărcând fişierul cu numele: *tomcat-4.1.17.tar.gz*.

Pasul 2) Dezarhivarea distribuţiei binare în directorul dorit de utilizator. În continuarea acestei lucrări, acesta va fi adresat astfel: TOMCAT_HOME.

Pasul 3) Setarea variabilei sistem *TOMCAT_HOME*, care să indice directorul în care a fost dezarhivat serverul tomcat. Acest pas este opţional, dar recomandat.

Următorul pas este reprezentat de configurarea portului pe care va rula serverul. El vine deja configurat pentru a rula pe portul 8080. Dacă utilizatorul doreşte modificarea acestei setări (în caz că deja există un serviciu care rulează pe acest port sau din alte motive), atunci trebuie să caute în fisierul *TOMCAT_HOME\conf\server.xml* locul în care este specificat portul (cauta valoarea 8080) şi să modifice această valoare conform preferintelor. Valoarea portului trebuie să fie mai mare de 1024 şi să nu fie folosită de un alt serviciu.

Serverul trebuie lansat în execuţie. Acest lucru se realizează prin pozitionarea în directorul TOMCAT_HOME\bin şi lansarea fişierului startup.bat (în cazul platformelor UNIX, startup.sh). Pentru oprire se foloseşte executabilul shutdown.bat (în cazul platformelor UNIX shutdown.sh).

Ultimul pas îl reprezintă verificarea. Utilizatorul trebuie să deschidă un browser şi să acceseze pagina de la adresa *http://localhost:8080*. Dacă s-a ales rularea pe alt port, această modificare trebuie facută şi în adresa introdusă în browser. În acest moment ar trebui ca utilizatorul să aibă acces la pagina de prezentare a serverului Tomcat. Pentru testarea completă, se recomandă accesarea linkului de exemple şi rularea unora dintre acestea.

2.2 Instalarea serverului Apache AXIS

Acestea sunt etapele care trebuie urmate pentru a instala serverul Apache AXIS:

Pasul 1) Obţinerea unei distribuţii binare a produsului. Ultima versiune oficială poate fi gasită întotdeauna la site-ul oficial al acestui produs, întreţinut de organizaţia Apache, producătoarea lui: http://xml.apache.org/axis. Pentru a o descărca se urmează linkul de 'downloads', şi de la pagina respectivă se alege versiunea dorită. Această lucrare foloseşte ultima versiune stabilă apărută la momentul scrierii ei, versiunea 1.0 (fişierul xml-axis-10.tar.gz).

Pasul 2) Dezarhivarea distribuţiei binare în directorul dorit de utilizator. În continuarea acestei lucrări, acesta va fi adresat astfel: AXIS_DIR.

Pasul 3) Instalarea AXIS ca aplicaţie web pentru Tomcat (sau oricare alt container de servleturi). Orice servlet container are un director în care se instalează aplicaţiile Web, deobicei denumit *webapps*. În cazul serverului Tomcat, acest director este *TOMCAT_HOME\webapps*. În acest director se copiază directorul *AXIS_DIR\webapps\axis*, director ce se găseşte în distribuţia de AXIS.

Pasul 4) Instalarea dependentelor. Pentru a rula, AXIS are nevoie de mai multe biblioteci java. Majoritatea acestora vin înglobate cu distribuţia de AXIS şi, după pasul anterior, le putem găsi în directorul *TOMCAT_HOME\webapps\axis\WEB-INF\lib*. Pentru a completa acest pas, în acest director mai trebuie copiată o librarie responsabilă cu parsarea documentelor XML. Există două variante mai des întâlnite: fişierul xerces.jar (care se obţine din

distribuţia produsului Xerces) sau crimson.jar şi jaxp.jar. Acestea pot fi obţinute de pe Internet sau chiar din interiorul tomcat (în loc de xerces.jar, se găseşte librăria xercesImpl.jar, în directorul TOMCAT_HOME\common\endorsed).

Pasul 5) Pornirea serverului Web (tomcat în cazul nostru). Dacă serverul era pornit la instalarea AXIS, atunci trebuie repornit pentru a putea identifica noua aplicaţie Web.

Pasul 6) Verificarea instalării. Pentru a verifica dacă instalarea a fost efectuată cu succes trebuie urmaţi paşii

- Se verifică serverul de Web este pornit dacă este pornit(în cazul nostru Tomcat).
- Se navighează la pagina de start a AXIS: *http://localhost:8080/axis*. Ar trebui să fie afişată pagina de început de la Apache AXIS.
- Se validează configuratia locală a instalării prin urmarea linkului : 'Validate the local installation configuration'. Aici se va rula un test în care AXIS verifică prezenţa librăriilor necesare, după care afisează diagnosticul: instalarea are toate componentele, sau, în caz negativ, sunt listate componentele care lipsesc. Acest pas nu asigură în intregime faptul că AXIS a fost instalat cu succes, dar garantează prezenţa unor librării obligatorii.
- Listarea serviciilor deja instalate. Din pagina de start a AXIS se urmează linkul *'View the list of deployed Web services'*. Rezultatul ar trebui să fie o pagină care conţine serviciile cu care AXIS este distribuit. Dând clic pe unul din linkurile 'wsdl', ar trebui să se afişeze un document wsdl ataşat serviciului respectiv.

Pasul 7) Ultimul pas îl reprezintă instalarea unui serviciu Web simplu şi rularea unui client care să îl acceseze. Acest pas va fi prezentat în subcapitolul următor deoarece introduce notiuni noi despre cum se crează şi instalează un serviciu, cât şi modul de a scrie un client pentru serviciul respectiv.

De menţionat faptul că toate aceste informaţii referitoare la instalarea şi rularea produselor Tomcat şi AXIS se găsesc şi în documentaţia cu care acestea sunt distribuite: TOMCAT_HOME\RUNNING.txt (pentru tomcat) şi AXIS_DIR\docs\install.html. În aceste documente se pot găsi informaţii vaste în legătură cu problemele ce pot apărea la instalare cât şi anumite recomandări.

2.3 Primul serviciu Web

Acest subcapitol va demonstra cum se crează un serviciu Web simplu şi cum se instalează acesta pe AXIS. De mentionat că modul de instalare este valabil doar pentru servicii simple, care nu necesită organizare pe directoare şi folosirea altor clase declarate în afara fişierului serviciu. Alt mod de instalare al serviciilor (cel mai des folosit şi recomandat de producătorii AXIS) va fi prezentat în capitolele următoare. Pentru moment acesta este modul ideal de a testa faptul că AXIS functionează corect.

2.3.1 Crearea şi instalarea serviciului "Calculator"

Serviciul creat trebuie să ofere două metode simple:

- *aduna* : primeste ca parametrii două valori întregi şi returnează suma acestora.
- *scade* : primeste ca parametrii două valori întregi şi returnează diferenţa acestora.

Pentru aceasta este necesară o clasă java care să ofere funcţionalitatea cerută. Ea va fi salvată într-un fişier Calculator.java şi va arăta astfel:

Listing 2.1: Exemplu de fisier WSDL

```
1 public class Calculator
2 {
3     public int aduna( int a , int b )
4     {
5         return a + b ;
6     }
7
8     public int scade( int a , int b )
9     {
10        return a - b ;
11    }
12 }
```

Acesta va fi serviciul *Calculator*. Pentru a-l instala pe AXIS, fisierul Calculator.java trebuie redenumit în Calculator.jws şi copiat în directorul TOMCAT_HOME\webapps\axis. Asta este tot. De acum acest serviciu poate fi accesat la URL-ul : http://localhost:8080/axis/Calculator.jws.

2.3.2 Crearea şi explicarea clientului pentru "Calculator"

Pentru a testa dacă într-adevăr serviciul "Calculator" funcţionează trebuie implementat un client. Acesta trebuie să cunoască locaţia la care se găseşte serviciul şi numele metodelor pe care le apelează. Acest lucru este uşor în acest caz deoarece serviciul a fost realizat de acelaşi programator.

Clientul este reprezentat în acest caz printr-o singură clasă, cu o metodă `main` care se lansează în execuţie. Ea va incărca cei doi parametrii, fiecare din ei fiind un întreg, după care va apela la început metoda de adunare a serviciului, iar apoi metoda de scădere a serviciului pe aceşti parametrii.

Pentru a păstra codul modularizat, va fi creată o procedură statică pentru apelarea unei metode cu un anumit nume şi cu doi parametrii de tip int. Iată cum arată clasa CalcClient:

Listing 2.2: Codul sursa al clasei CalcClient

```
1 import org.apache.axis.client.Call;
2 import org.apache.axis.client.Service;
```

```
3 import org.apache.axis.encoding.XMLType;
4 import org.apache.axis.utils.Options;
5 import javax.xml.rpc.ParameterMode;
6
7 public class CalcClient
8 {
9     public static final String SERVICE_URL =
10         "http://localhost:8080/axis/Calculator.jws" ;
11
12     public static void main( String[] args )
13     {
14         Integer a, b ;
15         if( args.length < 2 )
16         {
17             System.out.println("Nu sunt suficiente argumente." ) ;
18             System.exit( 1 ) ;
19         }
20         try
21         {
22             a = new Integer( args[0] ) ;
23             b = new Integer( args[1] ) ;
24
25             // apeleaza adunarea remote
26             System.out.print("Apelare adunare : " + a +
27                 " + " + b + " ... " ) ;
28             System.out.println( " = " + apeleaza( "aduna", a, b ) ) ;
29
30         // apeleaza scaderea remote
31             System.out.print("Apelare scadere : " + a +
32                 " - " + b + " ... " ) ;
33             System.out.println( " = " + apeleaza( "scade", a, b ) ) ;
34         }
35         catch( NumberFormatException nfe )
36         {
37             System.out.println("Argumentele nu sunt de tip intreg." ) ;
38             System.exit( 1 ) ;
39         }
40         catch( Exception e )
41         {
42             System.out.println("Exceptie aruncata : " ) ;
43             e.printStackTrace() ;
44         }
45
46     }
47
48
49     public static int apeleaza( String metoda, Integer a, Integer b )
```

```
50          throws Exception
51      {
52          Service service = new Service();
53          Call call = (Call) service.createCall();
54
55          call.setTargetEndpointAddress(new java.net.URL(SERVICE_URL));
56          call.setOperationName( metoda );
57          call.addParameter( "a", XMLType.XSD_INT, ParameterMode.IN );
58          call.addParameter( "b", XMLType.XSD_INT, ParameterMode.IN );
59          call.setReturnType( XMLType.XSD_INT );
60
61          Integer ret = (Integer) call.invoke( new Object[]{a,b});
62          return ret.intValue() ;
63      }
64
65  }
```

Clasele folosite îndeosebi în interiorul metodei *apeleaza* sunt definite în biblioteca axis.jar şi sunt folosite pentru a executa apeluri la distanţă. Denumirile claselor, semnătura metodelor şi sintaxa sunt clare, astfel că se învată uşor modul de utilizare. Pentru moment este destul. Mai multe detalii vor fi prezentate în capitolele care trateaza crearea aplicaţiei client folosindu-se motorul AXIS.

2.3.3 Compilarea şi rularea programului client

Deşi clasa principală a clientului este gata, ea mai trebuie compilată. Din păcate acest proces nu este chiar aşa de uşor, deoarece acest program trebuie să includă bibliotecile necesare pentru lucrul cu AXIS ca şi client. Acestea sunt aceleaşi ca şi în cazul utilizării AXIS ca server. Singura deosebire este că clientul trebuie să le adauge manual în CLASSPATH, pe cand în cazul serverului, acestea sunt încărcate automat de către serverul Web în care a fost instalat.

Deci ce trebuie să facem pentru a compila şi rula programul? Trebuie să setăm variabila sistem CLASSPATH să conţină următoarele biblioteci:

- *axis.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *commons-logging.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *jaxrpc.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *wsdl4j.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *saaj.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *log4j-1.2.4.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *axis-ant.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)

- *commons-discovery.jar* (TOMCAT_HOME\webapps\axis\WEB-INF\lib)
- *xercesImpl.jar*(TOMCAT_HOME\common\endorsed)
- *servlet.jar*(TOMCAT_HOME\common\lib)

După cum se vede, este o listă impresionantă. şi în general nu este recomandat să fie introduse bibliotecile unui program în variabila sistem CLASSPATH, deoarece, în cazul altor programe pot apărea conflicte. De exemplu am realizat un program care are nevoie de un anumit parser de java (xerces, versiunea 2.x.x). Dacă clientul nostru are nevoie de altă versiune deoarece nu poate rula cu o versiune aşa de veche, adăugând la CLASSPATH (la sfârşit) noua versiune de xerces, aceasta nu va fi luată în considerare deoarece clase cu acelaşi nume apar în prealabil. Dacă o introducem în faţă, atunci aplicatia mai veche nu va mai funcţiona deoarece există anumite diferenţe notabile între cele două versiuni de Xerces.

De obicei se recomandă ca, dacă o aplicatie are nevoie de anumite biblioteci externe, acestea să fie adaugate la CLASSPATH în momentul rulării sau compilării, folosind optiunea `-cp` a comenzii `javac`. Aceeaşi opţiune există şi pentru comanda java. Având în vedere numărul mare de biblioteci care trebuie folosite, nici această soluţie nu este tocmai bună, deoarece pentru fiecare rulare s-ar introduce un număr foarte mare de caractere.

O solutie mai bună ar fi ca, pentru fiecare proiect, programatorul să creeze un script executabil (pentru Windows un fişier cu extensia bat, pentru UNIX, să se seteze atributul de fişier executabil). Acest fişier să salveze CLASSPATH-ul iniţial, să adauge la acesta bibliotecile necesare proiectului, să ruleze programul, după care să seteze CLASSPATH-ul la vechea valoare.

Important Pentru a putea rula aplicaţia client, variabila sistem CLASSPATH trebuie să conţină şi biblioteca *xmlParserAPIs.jar*, aceasta conţinând implementarea efectivă a metodelor de parsare XML. În cazul acestui program, fişierul compileClient.bat va arăta astfel:

```
echo off

echo Salveaza vechiul CLASSPATH
SET OLDCP=%CLASSPATH%

echo Creeaza noul CLASSPATH
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\axis.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\commons-discovery.jar;%CLASSPATH
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\commons-logging.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\jaxrpc.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\log4j-1.2.4.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\saaj.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\wsdl4j.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\lib\servlet.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\endorsed\xmlParserAPIs.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\endorsed\xercesImpl.jar;%CLASSPATH%
```

```
echo Compileaza clientul ...
javac CalcClient

echo Reface CLASSPATH-ul
SET CLASSPATH=%OLDCP%

echo on
```

Pentru a rula programul, se înlocuieşte linia care conţine comanda de compilare cu comanda de rulare java. De asemenea se mai adaugă şi eventualele argumente de care are nevoie aplicaţia. Aşa arată fişierul runClient.bat :

```
echo off

echo Salveaza vechiul CLASSPATH
SET OLDCP=%CLASSPATH%

echo Creeaza noul CLASSPATH
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\axis.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\commons-discovery.jar;%CLASSPATH
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\commons-logging.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\jaxrpc.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\log4j-1.2.4.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\saaj.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\webapps\axis\WEB-INF\lib\wsdl4j.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\lib\servlet.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\endorsed\xmlParserAPIs.jar;%CLASSPATH%
SET CLASSPATH=%TOMCAT_HOME%\common\endorsed\xercesImpl.jar;%CLASSPATH%

echo Ruleaza clientul ...
java CalcClient 10 20

echo Reface CLASSPATH-ul
SET CLASSPATH=%OLDCP%

echo on
```

Presupunând că utilizatorul nu a avut nici o problemă în a compila sursa clasei `Calculator`, rezultatul rulării fişierului `run.bat` este următorul (de mentionat că la compilare nu este necesar ca serverul să fie pornit, dar la rularea programului acest lucru este obligatoriu):

```
Salveaza vechiul CLASSPATH
Creeaza noul CLASSPATH
Ruleaza clientul ...
Apelare adunare : 10 + 20 ...  = 30
Apelare scadere : 10 - 20 ...  = -10
Reface CLASSPATH-ul
```

Pentru a schimba datele de test, utilizatorul trebuie doar să modifice apelul comenzii java din fişierul `runClient.bat`, introducând valorile dorinte în locul celor afişate acum.

După cum se vede, nu este extrem de complicat. Dar mai există o altă variantă, mai elegantă, care poate ajuta utilizatorul în procesul de dezvoltare al unui serviciu Web, partea de client cât şi partea de server: folosirea utilitarului *ant*. Acesta primeşte ca parametru un fişier de intrare în care se specifică o listă de comenzi şi le procesează într-o ordine prestabilită. Pentru mai multe informaţii referitoare la folosirea acestui produs, a se consulta anexa.

Pentru a înţelege mai bine, voi exemplifica procesul de compilare cât şi procesul de rulare cu ajutorul acestui program. Pentru început trebuie creat un fişier text cu numele *build.xml* care să conţină comenzile *ant*. Acest fişier este în format XML.

Listing 2.3: Listingul fişierului *build.xml*

```
1 <project name="TeleSoap" default='run' >
2
3 <property name="tomcat.dir" value="d:/dovle/Diploma/tomcat/" />
4 <property name="axis.webdir"
5     value="${tomcat.dir}webapps/axis/WEB-INF/" />
6
7 <path id="compile.cp" >
8    <pathelement path="${axis.webdir}lib/commons-discovery.jar" />
9    <pathelement path="${axis.webdir}lib/jaxrpc.jar" />
10   <pathelement path="${axis.webdir}lib/saaj.jar" />
11   <pathelement path="${axis.webdir}lib/xerces.jar" />
12   <pathelement path="${axis.webdir}lib/axis.jar" />
13   <pathelement path="${axis.webdir}lib/commons-logging.jar" />
14   <pathelement path="${axis.webdir}lib/log4j-1.2.4.jar" />
15   <pathelement path="${axis.webdir}lib/wsdl4j.jar" />
16   <pathelement path="${tomcat.dir}common/lib/servlet.jar" />
17   <pathelement path="${classpath}"/>
18 </path>
19
20 <path id="run.cp" >
21     <pathelement path="." />
22     <path refid="compile.cp" />
23 </path>
24
25 <target name="compile">
26    <javac srcdir="." destdir="." classpathref="compile.cp"
27       includes="CalcClient.java" />
28 </target>
29
30 <target name="run" depends="compile">
31    <java classname="CalcClient" classpathref="run.cp" fork="true" >
32         <arg value="10" />
33         <arg value="20" />
34    </java>
```

```
35 </target>
36
37 </project>
```

Şi acum câteva cuvinte despre acest fişier şi comenzile folosite. Pentru a lansa în execuţie acest fişier utilizatorul va folosi programul ant. Dacă va dori rularea programului va tasta comanda `ant`. Parametrul run nu este necesar, deoarece acţiunea de rulare a programului este implicită. Pentru compilare, utilizatorul va lansa comanda `ant␣compile`.

Fişierul `build.xml` conţine următoarele informaţii:

- Linia 1 conţine numele proiectului şi acţiunea implicită. În acest caz este vorba despre rularea programului
- Liniile 3 — 5 setează proprietăţile reprezentând locaţia serverului Tomcat şi a serverului Apache AXIS. Deşi acest lucru nu este obligatoriu, se recomandă folosirea proprietăţilor pentru a organiza fişierele de comenzi ant. Astfel, în cazul în care se schimbă locaţia serverului de Tomcat, trebuie modificată doar valoarea proprietăţii respective.
- Liniile 7 — 18 definesc calea către bibliotecile cerute pentru compilarea programului.
- Liniile 20 — 23 definesc calea către fişierele compilate ale programului şi de asemenea către bibliotecile necesare pentru rulare. Acestea din urmă sunt cele folosite pentru compilare, astfel că această cale include şi pe cea definită anterior.
- Liniile 25 — 28 definesc acţiunea *compile*. Numele comenzilor şi a parametrilor folosiţi sunt elocvente.
- Liniile 30 — 35 definesc acţiunea *run*. Atributul `depends` indică faptul că această acţiune se va executa numai după ce compilarea a fost executată cu succes.

Având în vedere că, din acest moment, exemplele vor deveni tot mai complexe, această lucrare va prezenta pentru compilare şi rulare doar fişiere de comenzi `ant`. Pentru o înţelegere mai bună a produsului *Jakarta Ant* se recomandă parcurgerea anexei aferente cât şi a documentaţiei distribuită cu acesta.

2.4 Probleme şi sfaturi

Este posibilă apariţia unor probleme la instalarea unuia din produsele menţionate anterior, cum ar fi faptul că serverul respectiv nu funcţionează, sau la primirea unei cereri aruncă o anumită eroare şamd.

Într-o astfel de situaţie cel mai bine este să se verifice corectitudinea paşilor de instalare, după care să se studieze (din nou, dacă este cazul) documentaţia venită cu distribuţia produsului. Aceasta deobicei include un ghid detaliat de instalare, un manual de utilizare şi de asemenea un document cu întrebările frecvente sau cu erorile care pot să apară.

În caz că problema nu este rezolvată, se mai poate apela şi la o altă resursă: Internetul. Se poate căuta cu ajutorul unui motor de căutare, utilizatorul folosindu-se de cuvintele cheie care identifică problema (dacă este cazul chiar şi o secvenţă din mesajul de eroare).

Dacă nici această soluţie nu a dat roade, utilizatorul poate apela la listele de mailuri (mailing lists) care deobicei însoţesc produsele de acest tip. Se poate obţine accesul la acestea din site-ul oficial al produsului, urmărind linkul "Mailing list". Aceste liste de mailuri conţin mesaje transmise de utilizatori, întrebări, soluţii la anumite probleme şamd. Utilizatorul poate consulta arhivele listei şi să caute printre mesajele deja trimise sau chiar să trimită un mesaj prin care să explice problema pe care o are şi să spere să primească un răspuns de la utilizatorii care cunosc aceste probleme.

Referitor la organizarea fişierelor în faza de dezvoltare a serviciului Web sunt valabile aceleaşi consideraţii ca şi în cazul oricărui proiect de dimensiuni mari:

- Păstrarea fişierelor într-o structură ierarhică prestabilită.
- Păstrarea surselor separat de fişierele binare obţinute prin compilare şi de fişierele resursă ale proiectului
- Se recomandă folosirea unui utilitar care să ajute la compilarea surselor cât şi la alte actiuni auxiliare, cum ar fi instalarea serviciului, repornirea serverului şamd. Pentru majoritatea proiectelor din această lucrare s-a folosit utilitarul *Ant*. Pentru mai multe informaţii, a se consulta Apendicele referitor la utilizarea acestuia.

Capitolul 3

Dezvoltarea unui serviciu Web pentru AXIS

Următoarele capitole prezintă rezolvarea unei probleme folosind servicii Web. Pentru început aceasta va fi enunţată, apoi se va da organizarea claselor şi structura serviciului, după care se va arăta cum se implementează aceste clase şi modul de desfăşurare. Următoarea parte a lucrării prezintă implementarea unui client, folosind diverse opţiuni oferite de AXIS.

Problema aleasă spre rezolvare este practică şi lasă imaginaţia să se desfăşoare. Soluţia propusă de această lucrare, deşi funcţională, are mai mult scopuri didactice. Din această cauză se va renunţa la optimizarea codului în favoarea lizibilităţii. De asemenea funcţionalitatea serviciului va fi limitată, fiind foarte uşor de adăugat noi metode pe parcurs, în funcţie de nevoile proiectului.

3.1 Cerinţele proiectului

După cum ştim, serviciul de informaţii telefonice este deficitar. Pentru a afla un număr de telefon sau pentru a descoperi unde stă o persoană, pe baza numărului său de telefon, utilizatorul trebuie să sune la numărul de informaţii, să comunice (de multe ori destul de greoi) cu funcţionara RomTelecom şi să afle informaţia dorită. Pentru a rezolva această problemă se propune realizarea unui serviciu Web care să ofere o interfaţă de comunicare simplă pentru accesarea functionalităţii cerute. Acest proiect trebuie să ţntrunească următoarele cerinţe:

- Să permită căutarea unei persoane după numărul de telefon.
- Să permită căutarea unui număr de telefon pe baza unor informaţii despre posesorul acestuia (nume, adresă, oraş, judeţ şamd).

Acestea sunt cerinţele minimale pentru un sistem care să ajute utilizatorul să afle informaţii despre un număr de telefon sau despre o persoană. Fiind un sistem public, nu va necesita autentificare, ci doar va oferi servicii tuturor care le solicită.

3.2 Descrierea soluţiei

Ca rezultat al cerinţelor enunţate mai sus, am realizat aplicaţia *TeleSOAP*. Structura acesteia este simplă şi poate fi folosită pentru dezvoltarea oricărui serviciu Web. Modularizarea oferă posibilitatea extinderii acestei implementări, adăugând noi metode şi criterii de căutare.

Elementul central al aplicaţiei îl reprezintă clasa serviciu Web. Aceasta trebuie să expună două metode, una pentru fiecare din funcţionalităţile cerute. Având în vedere că foloseşte date ce trebuie stocate, clasa serviciu se va folosi de o clasă ce intermediază accesul la informaţii. Şi dacă tot am pomenit de informaţii, mai trebuie create clasele de date şi clasa criteriu de căutare. Pentru a înţelege mai bine relaţiile dintre aceste clase se recomandă studierea diagramei UML de clase din figura 3.1.

Iată câteva detalii despre fiecare clasă în parte.

În primul rÂnd trebuie definită şi organizată informaţia folosită de acest serviciu Web. Având în vedere că el trebuie să conţină informaţii despre persoane şi numerele lor de telefon, este evidentă prezenţa unei clase care să păstreze informaţiile despre un abonat: *Persoana*. Datele păstrate sunt următoarele :

- *Nume* păstrează numele persoanei în cauză,
- *Prenume* păstrează prenumele persoanei,
- *Telefon* păstrează numărul de telefon,
- *Adresa* păstrează adresa abonatului.

Primele două atribute ale clasei sunt de tip `String`. Pentru a complica o idee solutia, câmpul `Adresa` va fi tot un obiect complex, definit de următoarele atribute:

- *Strada*
- *Numar*
- *Oras*
- *Judet*

De asemenea câmpul `Telefon` este unul complex, alcătuit din două atribute de tip `String`: *Prefix* şi *Numar*. A fost ales acest tip de dată (String) deoarece se pot executa mult mai uşor operaţii de căutare de substringuri (în caz că utilizatorul doreşte un număr de telefon care include o anumită secvenţă).

Clasa care organizează datele şi permite accesul la ele se va numi *DataHolder*. Deoarece acest proiect are doar rol didactic, ea va păstra în memorie un număr prestabilit de persoane. Acestea vor fi încărcate de mână la instanţierea acesteia. Clasa va oferi metode variate de acces pentru a putea obţine informaţii despre persoanele înregistrate. În cazul implementării acestui proiect pentru lumea reală, această clasă va conţine informaţii despre localizarea datelor şi, în loc de a extrage obiectele din proprii membri, va accesa baze de date.

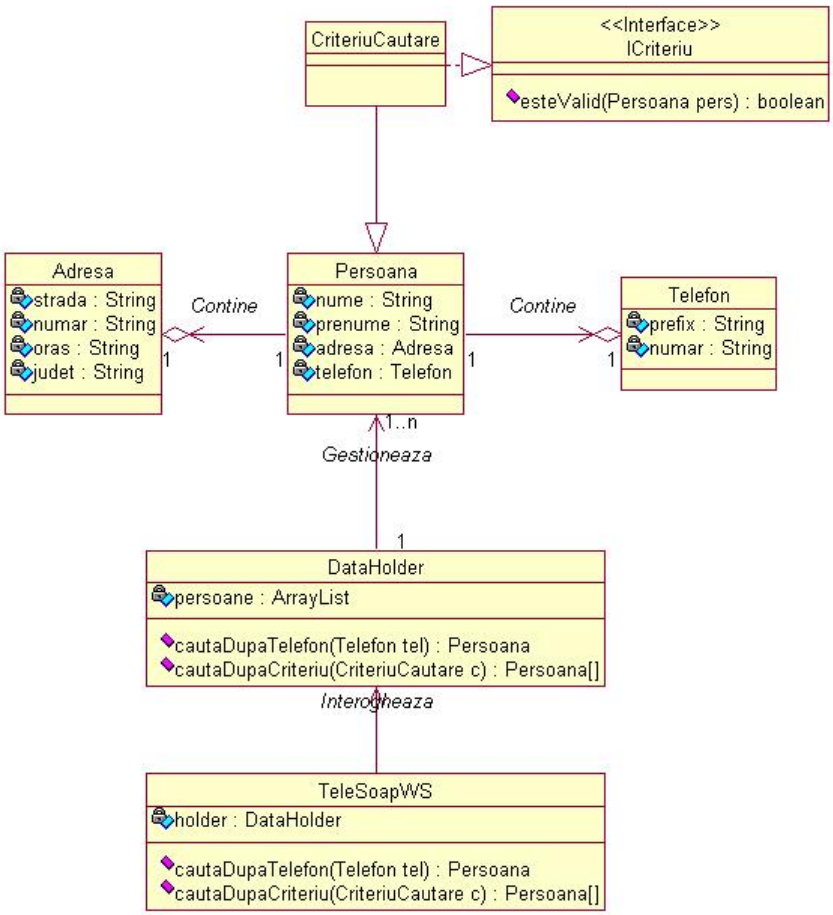

Figura 3.1: Diagrama UML de clase pentru TeleSoap

Pentru a permite o căutare cât mai flexibilă, va fi introdusă şi o clasă care să execute căutarea avansată. Aceasta clasă se numeşte *CriteriuCautare* şi are următoarele proprietăţi:

- Contine atribute care definesc criteriul de căutare printe persoane
- Defineşte o metodă care primeşte o persoană ca parametru şi returnează adevărat dacă acea persoană întruneşte criteriile cerute.
- Atributele acestei clase pot fi serializate şi deserializate în format SOAP pentru a putea fi primită de la client.

Şi acum clasa care va implementa funcţionalitatea cerută de proiect: *TeleSoap*. Aceasta va fi expusă direct, ca serviciu Web, prin intermediul serverului AXIS. Structura ei este următoarea:

- *Persoana cautaPersoana(Telefon tel)* . Această metodă va căuta o persoană după un număr de telefon.
- *Persoana[] cautaPersoane(CriteriuCautare crit)* . Aceasta metoda aplică criteriul de căutare pentru toate persoanele aflate în evidenţa programului şi le returnează pe cele care îndeplinesc condiţiile cerute.

Mai trebuie făcute câteva precizări în legătură cu clasele ce definesc obiectele care vor fi primite de la client sau transmise acestuia. Aceste precizări vor fi explicate mai în detaliu când se va instala aplicaţia, fiind legate în mare măsură de fişierul responsabil cu configurarea serviciului. Clasele `Persoana`, `Telefon`, `Adresa` şi `CriteriuCautare` trebuie să îndeplinească specificaţiile pentru clase de tip JavaBeans. Ele trebuie să aibă un constructor fără paramentrii şi de asemenea metode pentru obţinerea fiecărui atribut şi pentru setarea acestuia. Acest lucru este necesar deoarece Apache AXIS oferă o componentă specială pentru serializarea şi deserializarea unor astfel de obiecte. Dacă nu sunt satisfăcute aceste specificaţii, atunci utilizatorul trebuie să îşi implementeze propriul serializator şi deserializator.

3.3 Implementarea claselor de date

Obiectele de acest tip vor fi transmise ca parametrii la apeluri de proceduri la distanţă sau vor fi returnate de acestea. Pentru a uşura munca programatorului, produsul Apache AXIS vine cu o gamă largă de serializori şi deserializori. Serializorul `org.apache.axis.encoding.ser.BeanSerializer` permite încapsularea unui obiect de tip JavaBean în XML fără a necesita vreun efort din partea programatorului. Datorită acestor considerente este recomandată folosirea acestei clase în cazul în care nu trebuie executate acţiuni speciale asupra obiectului la serializare şi deserializare.

O clasă de tip JavaBean are următoarele caracteristici:

- Este o clasă complexă, adică este alcătuită din mai multe atribute.

- Defineşte un constructor implicit, (un constructor fara nici un parametru). Acest lucru nu împiedică clasa să definească şi alţi constructori, pentru a uşura lucrul programatorului, dar aceştia nu vor fi folositi de către AXIS.
- Fiecare atribut care poate fi trimis la distanţă are asociată o metodă publică de citire. Presupunând că atributul este de forma `<tip> myAtr` atunci metoda va avea următoarea semnătură: `public <tip> getMyAtr( )`.
- Fiecare atribut care poate fi primit de la distanţă are asociată o metodă de setare. Presupunând că atributul este de forma `<tip> myAtr` atunci metoda va avea următoarea semnătură: `setMyAtr( <tip> val )`.

3.3.1 Implementarea clasei *Persoana*.

Având în vedere cele spune mai înainte, clasa `Persoana` va arăta astfel:

Listing 3.1: Codul sursa al clasei Persoana

```
1  package com.telesoap.data;
2
3
4  public class Persoana
5  {
6      protected String nume = null ;
7      protected String prenume = null ;
8      protected Adresa adresa = null ;
9      protected Telefon telefon = null ;
10
11     public Persoana()
12     {
13     }
14
15     public Persoana( String nume, String prenume, Adresa adresa,
16         Telefon telefon )
17     {
18         this.nume = nume ;
19         this.prenume = prenume ;
20         this.adresa = adresa ;
21         this.telefon = telefon ;
22     }
23
24     public Adresa getAdresa()
25     {
26         return adresa;
27     }
28
29     public String getNume()
30     {
```

```
31          return nume;
32      }
33
34      public String getPrenume()
35      {
36          return prenume;
37      }
38
39      public Telefon getTelefon()
40      {
41          return telefon;
42      }
43
44      public void setAdresa(Adresa adresa)
45      {
46          this.adresa = adresa;
47      }
48
49      public void setNume(String nume)
50      {
51          this.nume = nume;
52      }
53
54      public void setPrenume(String prenume)
55      {
56          this.prenume = prenume;
57      }
58
59      public void setTelefon(Telefon telefon)
60      {
61          this.telefon = telefon;
62      }
63
64  }
```

În acest caz, implementarea grăieşte de la sine. Condiţiile pentru a fi o clasă de tip JavaBean sunt îndeplinite.

3.3.2 Implementarea claselor Adresa şi Telefon.

Aceste clase au parte de acelaşi tratament. Altfel spus, şi ele trebuie să fie de tip JavaBean. Având în vedere simplitatea codului sursă, listarea va fi partială, incluzând doar definirea atributelor.

Listing 3.2: Codul sursa al clasei Persoana

```
1 package com.telesoap.data;
2
```

```
3 public class Adresa
4 {
5     private String strada = null ;
6     private String numar = null;
7     private String oras = null;
8     private String judet = null;
9       .
10      .
11      .
12 }
```

Listing 3.3: Codul sursa al clasei Persoana

```
1 package com.telesoap.data;
2
3 public class Telefon
4 {
5     public String prefix = null ;
6     public String numar = null ;
7       .
8       .
9       .
10 }
```

3.3.3 Implementarea clasei criteriu

Clasa `CriteriuCautare` trebuie să conţină informaţii despre ceea ce utilizatorul doreşte să caute. Deoarece utilizatorul va căuta persoane, fiecare câmp al clasei `Persoana` va trebui să aibă unul sau mai multe câmpuri corespondente în clasa `Criteriu`. Acestea vor reprezenta criteriul de căutare asupra câmpului respectiv.

În acest caz am hotărât ca structura clasei criteriu să fie întru totul asemănătoare cu cea a clasei `Persoana`. Diferenţa o constituie rolul pe care îl are fiecare atribut Criteriu, ce înseamnă el pentru atributul corespondent din clasa Persoana. Dacă atributul `X` al criteriului este `null` sau un `String` vid, atunci criteriul nu este interesat de câmpul +X+ al persoanelor. Dacă atributul nu este vid atunci o persoană indeplineşte criteriul dacă şi numai dacă şirul respectiv este conţinut de atributul persoanei.

Având în vedere faptul că criteriul va conţine aceleaşi câmpuri ca şi persoana, concluzia logică este ca va extinde clasa `Persoana`.

În afară de aceste valori, clasa Criteriu trebuie să ofere o metodă care să verifice dacă o persoană (transmisă ca parametru) îndeplineşte criteriul sau nu.

Listing 3.4: Codul sursa al clasei CriteriuCautare

```
1 package com.telesoap;
2
3 import com.telesoap.data.Persoana;
4
```

```
5  public class CriteriuCautare
6     extends Persoana
7  {
8
9     public CriteriuCautare()
10    {
11    }
12
13
14    public boolean esteValid( Persoana pers )
15    {
16       if( ! esteCampValid(pers.getNume(), nume ) )
17          return false ;
18
19       if( ! esteCampValid(pers.getPrenume(), prenume ) )
20          return false ;
21
22       if( telefon != null )
23       {
24          if( ! esteCampValid(pers.getTelefon().getPrefix(),
25             telefon.getPrefix() ) )
26             return false ;
27          if( ! esteCampValid(pers.getTelefon().getNumar(),
28             telefon.getNumar() ) )
29             return false ;
30       }
31
32       if( adresa != null )
33       {
34          if( ! esteCampValid(pers.getAdresa().getStrada(),
35             adresa.getStrada() ) )
36             return false ;
37          if( ! esteCampValid(pers.getAdresa().getNumar(),
38             adresa.getNumar() ) )
39             return false ;
40          if( ! esteCampValid(pers.getAdresa().getOras(),
41             adresa.getOras() ) )
42             return false ;
43          if( ! esteCampValid(pers.getAdresa().getJudet(),
44             adresa.getJudet() ) )
45             return false ;
46       }
47
48       return true ;
49    }
50
51    private boolean esteCampValid( String text, String mask )
```

```
52      {
53          if( ( mask != null ) && ( mask.length() > 0 ) )
54              return text.indexOf( mask ) >= 0 ;
55          return true ;
56      }
57  }
```

Dupa cum am precizat şi mai înainte, clasa Criteriu va extinde clasa `Persoana` pentru a nu redefini aceleaşi atribute şi funcţionalitate (metodele de setare şi de citire a valorilor). Acest lucru este reflectat de linia 6.

Punctul cheie al acestei clase îl reprezintă metoda de verificare a apartenentei unei persoane la criteriu. Aceasta este definită între liniile 14 şi 43. Pe scurt, verifică fiecare câmp în parte dacă este valid. Începe prin validarea numelui şi a prenumelui. Dacă `Adresa` nu este setată în criteriu, atunci nu încearcă să facă verificarea şi pe acest câmp. La fel procedează şi cu câmpul `Telefon`.

Metoda care face compararea a două câmpuri, unul din criteriu si altul din obiectul Persoana este definită începând cu linia 45. Ea returnează `true` dacă atributul criteriului conţine informaţie şi dacă acesta este conţinut de atributul persoanei. În caz contrar metoda returnează `false`.

3.4 Implementarea clasei DataHolder

Pana acum am definit clasele care conţin informaţia cu care lucrează serviciul TeleSOAP şi clasa care execută verificări de bază asupra unei persoane. Această sectiune defineste clasa responsabilă cu păstrarea şi accesarea informaţiei referitoare la persoane.

Având în vedere că acest serviciu a fost creat mai mult cu scop didactic, această clasa va păstra o listă predefinită de persoane, cu care va fi iniţializată la început.

De asemenea clasa va permite interogări de tipul : 'returnează persoana cu un anumit număr de telefon' şi 'returnează persoanele care îndeplinesc un anumit criteriu'. Implementarea acestora este clară şi nu necesită explicaţii suplimentare.

Listing 3.5: Codul sursa al clasei DataHolder

```
1  package com.telesoap;
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5
6  import com.telesoap.data.Adresa;
7  import com.telesoap.data.Persoana;
8  import com.telesoap.data.Telefon;
9
10 public class DataHolder
11 {
12     private ArrayList persoane ;
```

```
13
14    public DataHolder()
15    {
16       persoane = new ArrayList() ;
17
18       persoane.add( creazaPersoana("Muresan", "Domnica",
19          "Alexandru Ioan Cuza", "2", "Codlea", "Brasov",
20          "0268", "250175" ) ) ;
21       persoane.add( creazaPersoana("Muresan", "Marian",
22          "Mircea Eliade", "", "Fagaras", "Brasov",
23          "0268", "214520" ) ) ;
24       persoane.add( creazaPersoana("Muresan", "Alexandru",
25          "Lamaitei", "62", "Brasov", "Brasov",
26          "0268", "320773" ) ) ;
27       persoane.add( creazaPersoana("Muresan", "Carolina",
28          "Grivitei", "56", "Brasov", "Brasov",
29          "0268", "161084" ) ) ;
30       persoane.add( creazaPersoana("Marian", "Nicolae",
31          "Branduselor", "56", "Brasov", "Brasov",
32          "0268", "184148" ) ) ;
33       persoane.add( creazaPersoana("Dragan", "Mihaela",
34          "Margaretei", "P12", "Brasov", "Brasov",
35          "0268", "132418" ) ) ;
36       persoane.add( creazaPersoana("Acvila", "George",
37          "13 Decembrie", "69", "Brasov", "Brasov",
38          "0268", "165572" ) ) ;
39       persoane.add( creazaPersoana("Simbad", "Marinarul",
40          "Carpatilor", "19", "Brasov", "Brasov",
41          "0268", "310339" ) ) ;
42    }
43
44    public Persoana cautaDupaTelefon( Telefon telefon )
45       throws PersoanaNotFoundException
46    {
47       Persoana p ;
48       for( Iterator it = persoane.iterator() ; it.hasNext() ; )
49       {
50          p = (Persoana) it.next() ;
51          if( p.getTelefon().equals( telefon ) )
52             return p ;
53       }
54
55       throw new PersoanaNotFoundException(
56          "Nu exista persoana cu numarul de telefon : " + telefon ) ;
57    }
58
59    public ArrayList cautaDupaCriteriu( CriteriuCautare c )
```

```
60      {
61          ArrayList rez = new ArrayList() ;
62          Persoana p ;
63          for( Iterator it = persoane.iterator() ; it.hasNext() ; )
64          {
65              p = (Persoana) it.next() ;
66              if( c.esteValid( p ) )
67                  rez.add( p ) ;
68          }
69
70          return rez ;
71      }
72
73      private Persoana creazaPersoana(
74          String nume, String prenume,
75          String strada, String numar, String oras, String judet,
76          String prefix, String tel )
77      {
78          return new Persoana( nume, prenume,
79              new Adresa(strada, numar, oras, judet),
80              new Telefon( prefix, tel )
81          ) ;
82      }
83
84  }
```

In cazul implementării acestui serviciu Web pentru lumea reală, această clasă nu ar păstra informaţia direct, ci doar ar accesa-o, pe baza anumitor informaţii auxiliare, cum ar fi locaţia bazei de date, numele si parola utilizatorului şamd.

3.5 Implementarea clasei serviciu Web

Până acum, nimic special. Surprinzător, dar nici clasa `TeleSoapWS` nu are o structură specială. Este o clasă oarecare, cu metode publice, care exportă funcţionalitatea cerută. Iată cum arată codul sursă:

Listing 3.6: Codul sursa al clasei TeleSoapWS

```
1  package com.telesoap;
2
3  import java.util.ArrayList;
4
5  import com.telesoap.data.Persoana;
6  import com.telesoap.data.Telefon;
7
8  public class TeleSoapWS
9  {
```

```
10      private static DataHolder data = new DataHolder() ;
11
12      public Persoana cautaDupaTelefon( Telefon telefon )
13          throws PersoanaNotFoundException
14      {
15          return data.cautaDupaTelefon( telefon ) ;
16      }
17
18      public Persoana[] cautaDupaCriteriu( CriteriuCautare c )
19      {
20          ArrayList l = data.cautaDupaCriteriu( c ) ;
21
22          Persoana[] rez = new Persoana[ l.size() ] ;
23          for( int i = 0, size = l.size() ; i < size ; i++ )
24              rez[ i ] = (Persoana) l.get( i ) ;
25
26          return rez ;
27      }
28
29  }
```

Este necesară o singură precizare: de ce câmpul `data` a fost marcat cu specificatorul `static`? Răspunsul nu este foarte complicat. Serverul de SOAP, când primeşte o cerere pentru un anumit serviciu, în mod implicit instanţiază un nou obiect de tipul clasei serviciu (în cazul nostru un nou obiect de tipul `com.telesoap.TeleSoapWS`), după care apelează metoda cerută a obiectului respectiv. Dacă nu am fi setat atributul `data` ca fiind static, atunci ar fi trebuit să fie iniţializat în constructorul clasei serviciu, lucru care nu are nici un sens, având în vedere că dorim accesarea aceloraşi informaţii. Astfel că acest atribut, fiind static, este instanţiat o singură dată, la încărcarea serviciului.

Capitolul 4

Configurarea şi instalarea unui serviciu Web

4.1 Fisierele WSDD

În AXIS, cu excepţia serviciilor foarte simple, care pot fi instalate sub forma unor fişiere *.jws*, singura modalitate de a instala un serviciu este de a-l descrie printr-un fisier WSDD (Web Service Deployment Descriptor) şi apoi de a-l înregistra pe server prin intermediul fişierului de configurare wsdd.

Deci acest fişier wsdd are ca rol păstrarea, organizarea şi transmiterea informaţiilor referitoare la un serviciu către server, în vederea desfăşurării. Formatul intern de organizare este XML, datorită flexibilităţii şi a nivelului de organizare a datelor. Fişierul conţine următoarele informaţii:

- Tipul de fişier : de desfăşurare a seviciului, de ştergere a serviciului, de listare de informaţii despre serviciile instalate.
- Serviciile care le instalează. Fiecare serviciu este reprezentat printr-un nod XML. Fiecare astfel de nod care defineşte un serviciu Web conţine următoarele informaţii:
  - Numele serviciului
  - Tipul serviciului. Poate fi `java:RPC` (Remote Procedure Call: presupune o cerere şi un răspuns, tratând cererile sincron) sau `java:MSG` (Messaging: presupune doar o cerere care nu are nevoie de răspuns, tratarea lor se face asincron).
  - Numele clasei care reprezintă serviciul (nume calificat).
  - Scopul serviciului. Acesta poate avea următoarele valori:
    * `Application`. Clasa serviciu este instanţiată o singură dată şi este folosită mereu aceeaşi instanţă pentru a servi toate cererile
    * `Request` Pentru fiecare cerere se instanţiază o nouă clasă serviciu, căreia i se transmite cererea, după care este eliberată.

    - `Session` O clasă serviciu este creată în momentul în care un client începe o sesiune, după care doar clientul respectiv are acces la valorile atributelor ei. Mecanismul de menţinere al sesiunii înca nu este standardizat.
  - Metodele expuse de serviciu. Aici se pot seta toate metodele cu ajutorul semnului * sau doar o listă de metode, separate prin virgulă.
  - Handlere pe fluxul de intrare(componente de procesare a cererii). Acestea procesează anumite câmpuri ale cererii şi le modifică pentru a putea fi folosite de serverul AXIS. Un astfel de handler este cel care face organizarea de sesiuni. De asemenea utilizatorul poate să îşi definească propriile handlere, de exemplu pentru a implementa anumite politici de securitate.
  - Handlere pe fluxul de iesire(componente de procesare a răspunsului). Odată ce răspunsul a fost împachetat în format SOAP de către serverul AXIS, acesta poate fi îmbogăţit sau modificat. De exemplu se poate adăuga informaţie despre identificatorul sesiunii folosite.

- Mapările claselor de tip JavaBean folosite de serviciu. Acest lucru se realizează prin intermediul tagului XML `beanMapping`. Acesta conţine următoarele informaţii :
  - Numele XML (qname : qualified name). Prin intermediul lui se poate obţine obiectul java din documentul XML
  - Numele clasei.
- Mapările generale ale claselor folosite de serviciu. Acest lucru se realizează prin intermediul tagului XML `typeMapping`. Acesta conţine următoarele informaţii :
  - Numele XML (qname : qualified name). Prin intermediul lui se poate obţine obiectul java din documentul XML
  - Numele clasei.
  - Numele clasei serializor.
  - Numele clasei deserializor.

Acest fişier este editat de programator. Nu este foarte greu de scris, dar pentru uşurinţă se recomandă copierea unui fişier de desfăşurare al unor exemple soap din distribuţia oficială şi apoi modificarea acestuia, conform necesităţilor.

4.2 Configurarea TeleSoap cu WSDD

Deşi fişierul wsdd de desfăşurare al aplicaţiei TeleSoap nu este foarte complex, listarea acestuia va ajuta la înţelegerea structurii de bază a unui astfel de document.

Listing 4.1: Fisierul WSDD de desfasurare al TeleSoap

```
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3
```

```
4    <service name="Telesoap" provider="java:RPC" >
5       <parameter name="className" value="com.telesoap.TeleSoapWS"/>
6       <parameter name="scope" value="Application" />
7       <parameter name="allowedMethods" value="*"/>
8    </service>
9
10   <beanMapping
11      qname="sample:Persoana"
12      xmlns:sample="http://info.unitbv.ro/sample"
13         languageSpecificType="java:com.telesoap.data.Persoana"/>
14
15   <beanMapping
16      qname="sample:Adresa"
17      xmlns:sample="http://info.unitbv.ro/sample"
18         languageSpecificType="java:com.telesoap.data.Adresa"/>
19
20   <beanMapping
21      qname="sample:Telefon"
22      xmlns:sample="http://info.unitbv.ro/sample"
23            languageSpecificType="java:com.telesoap.data.Telefon"/>
24
25   <beanMapping
26      qname="sample:Criteriu"
27      xmlns:sample="http://info.unitbv.ro/sample"
28            languageSpecificType="java:com.telesoap.CriteriuCautare"/>
29
30 </deployment>
```

După cum se vede din prima linie, acesta este un fişier de desfăşurare. Se declară spaţiul de nume specific fişierelor wsdd de la Apache şi de asemenea spaţiul de nume pentru elementele java.

Începând de la linia 4 şi până la linia 8, se defineşte serviciul TeleSoap astfel: numele său este TeleSoap si este de tip RPC în Java. Clasa care implementează funcţionalitatea sa este `com.telesoap.TeleSoapWS`. Scopul este `Aplication`. Expune toate metodele publice ale clasei.

Următoarele taguri definesc mapările pentru clasele JavaBean pe care serviciul le foloseşte.

4.3 Instalarea

Instalarea unui serviciu Web pe Apache AXIS se realizează în mai multi paşi.

Pasul 1) Pornirea serverului de aplicaţii. În cazul nostru este vorba de Tomcat.

Pasul 2) Compilarea fişierelor serviciului Web folosind un compilator de Java.

Pasul 3) Copierea fişierelor respective în directorul TOMCAT_HOME/webapps/axis/WEB-INF/classes
Pentru o mai bună organizare se recomandă împachetarea claselor ce reprezintă serviciul într-o bibliotecă jar şi copierea acesteia în directorul TOMCAT_DIR/webapps/axis/WEB-INF/lib. Urmând această recomandare, este mult mai uşor de dezinstalat un serviciu Web.

Pasul 4) Înregistrarea serviciului prin intermediul fişierului wsdd.

Pasul 5) Repornirea serverului de aplicaţii (sau, dacă este posibil, doar repornirea serverului de SOAP).

Toţi paşii sunt clari cu exceptia celui de-al patrulea: înregistrarea serviciului Web. Pentru a putea încheia cu succes acest pas trebuie să avem la dispoziţie fişierul wsdd al serviciului Web (în cazul nostru este vorba despre fişierul deployTeleSoap.wsdd). De asemenea utilizatorul trebuie să aibă în calea de căutare a bibliotecilor Java bibliotecile enumerate în primul capitol necesare compilării unui program client.

Pentru înregistrare, utilizatorul trebuie să ruleze următoarea comandă:
`java org.apache.axis.client.AdminClient [-p <port>] <deploy_file>`
unde ¡port¿ reprezintă numărul portului pe care rulează serverul. Daca nu este dat, se consideră valoarea implicită, adică 8080.

În cazul serviciului TeleSoap, comanda arată astfel:
`java org.apache.axis.client.AdminClient deployTeleSoap.wsdd`
Dacă variabila CLASSPATH conţine toate bibliotecile necesare, dacă serverul este pornit şi dacă nu apare nici o problemă, rezultatul ar trebui să arate astfel:

```
<Admin>Done processing</Admin>
```

O soluţie viabilă pentru acest al patrulea pas este realizarea unui fişier de comenzi care să seteze variabila CLASSPATH să conţină toate bibliotecile necesare, după care să ruleze comanda mai sus prezentată.

Acest proces nu terbuie reluat de fiecare dată când serverul de aplicaţii este pornit. Odată executată această comandă, serverul păstrează informaţiile despre serviciul respectiv şi le reîncarcă la repornire.

Dar totuşi există anumite momente când reîncărcarea sau chiar redesfăşurarea serviciului Web sunt necesare. Aceste actiuni apar mai ales în faza de dezvoltare a unui serviciu Web şi la faza de actualizare:

- Dacă autorul modifică clasele de care se foloseste serviciul Web, atunci trebuie să le recompileze, după care să le copieze peste vechile clase aflate in directorul unde le-a copiat initial (lib sau classes). După acest pas trebuie repornit serverul de SOAP.
- Dacă se aduc modificări la interfaţa expusă, la mapările de obiecte (se adaugă, se scot sau se modifică parametrii din fişierul wsdd) atunci serviciul Web trebuie redesfăşurat. Asta înseamnă că, în afară de copierea fişierelor noi peste cele vechi, mai trebuie rulată şi comanda de instalare. Evident, pentru a ţine cont de modificări, serverul de SOAP trebuie repornit.

4.4 Folosirea ant pentru desfasurare

După cum se observă, procedeul de desfăşurare este destul de complicat şi necesită efectuarea mai multor paşi. Dacă aplicaţia s-ar instala o singură dată nu ar fi foarte complicat, dar din păcate în timpul procesului de dezvoltare al unui serviciu Web este nevoie de multe testări, instalări, reinstalări şamd. De aceea acest proces poate deveni extrem de agasant.

O soluţie binevenită o reprezintă folosirea utilitarului *ant* pentru a efectua toate aceste acţiuni. În prima fază se depune efort pentru realizarea unui fişier adecvat de comenzi ant, dar apoi totul decurge simplu, fără nici o problemă, fiecare redesfăşurare fiind chemată printr-o comandă scurtă.

În continuare voi prezenta un fişier build.xml prin intermediul căruia se poate face partea de copiere şi desfăşurare a serviciului TeleSoap. De asemenea, dacă se foloseste serverul Tomcat, decomentând anumite comenzi se poate face şi repornirea serverului de SOAP.

Listing 4.2: Fisierul build.xml pentru instalarea serviciului TeleSoap

```
1 <project name="TeleSoap" default='refresh' >
2
3 <!-- Axis values -->
4 <property name="axis.dir" value="d:/dovle/Diploma/tomcat/webapps/axis/" />
5 <property name="axis.webdir" value="${axis.dir}WEB-INF/" />
6 <property name="axis.web.path" value="/axis" />
7
8 <property name="deploy.wsdd" value="deployTeleSoap.wsdd" />
9 <property name="undeploy.wsdd" value="undeployTeleSoap.wsdd" />
10
11 <!-- TOMCAT : config proprietati pentru folosirea Application Manager >
12 <property name="tomcat.url"      value="http://localhost:8080/manager"/>
13 <property name="tomcat.username" value="dovle"/>
14 <property name="tomcat.password" value=""/ -->
15
16
17 <!-- Configure the custom Ant tasks for the Manager application >
18 <taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask"/ -->
19
20 <path id="axiscp" >
21    <pathelement path="${classpath}"/>
22    <pathelement path="${axis.webdir}lib/commons-discovery.jar" />
23    <pathelement path="${axis.webdir}lib/jaxrpc.jar" />
24    <pathelement path="${axis.webdir}lib/saaj.jar" />
25    <pathelement path="${axis.webdir}lib/xerces.jar" />
26    <pathelement path="${axis.webdir}lib/axis.jar" />
27    <pathelement path="${axis.webdir}lib/commons-logging.jar" />
28    <pathelement path="${axis.webdir}lib/log4j-1.2.4.jar" />
29    <pathelement path="${axis.webdir}lib/wsdl4j.jar" />
30    <pathelement path="d:/dovle/Diploma/tomcat/common/lib/servlet.jar" />
31 </path>
```

```
32
33 <!-- INSTALL TARGETS -->
34 <target name="compile" >
35    <javac srcdir="src/" destdir="bin/" />
36 </target>
37
38 <target name="refresh" depends="compile" >
39    <delete file="${ant.webdir}lib/telesoap.jar" />
40
41    <jar destfile="${axis.webdir}lib/telesoap.jar"
42       basedir="bin" />
43
44    <!-- reload url="${tomcat.url}"  username="${tomcat.username}"
45       password="${tomcat.password}" path="${axis.web.path}"/ -->
46 </target>
47
48 <target name="deploy" depends="refresh">
49    <!-- Call the axis AdminClient on the deployment file wsdd -->
50    <java classname="org.apache.axis.client.AdminClient"
51       classpathref="axiscp" fork="true" >
52       <arg value="${deploy.wsdd}" />
53    </java>
54 </target>
55
56
57 <!-- UNINSTALL TARGET -->
58 <target name="undeploy" >
59    <java classname="org.apache.axis.client.AdminClient"
60       classpathref="axiscp" fork="true" >
61       <arg value="${undeploy.wsdd}" />
62    </java>
63
64    <delete file="${ant.webdir}lib/telesoap.jar" />
65
66    <!-- reload
67       url="${tomcat.url}"
68       username="${tomcat.username}"
69       password="${tomcat.password}"
70       path="${axis.web.path}"/ -->
71 </target>
72
73 </project>
```

Se setează ca acţiune implicită `refresh` deoarece, în perioada de dezvoltare, aceasta este cea mai des întâlnită situaţie. Următoarele linii setează directoarele în care se va face instalarea (directoarele axis).

Liniile 8 si 9 definesc fişierele cu ajutorul cărora se va face desfăsurarea sau stergerea

serviciului.

De asemenea mai este nevoie de locaţia bibliotecilor necesare pentru compilarea şi rularea aplicaţiei de desfăşurare. Acest lucru se realizează prin definirea componentei `axiscp`, între liniile 20 şi 31.

Urmează definirea acţiunilor de desfăşurare :

- *compile* compilează serverul şi plasează fişierele class în directorul bin.
- *refresh* reîncarcă serviciul. În primul rând aşteaptă execuţia acţiunii de compilare, după care şterge arhiva precedentă şi o crează din nou în directorul lib al AXIS. Dacă se lucrează cu tomcat, decomentând ultimele linii ale acţiunii, se restartează serviciul AXIS.
- *deploy* desfăşoară serviciul. Întâi aşteaptă rularea acţiunii de `refresh` după care lansează comanda de înregistrare a serviciului. Dacă acesta a fost deja înregistrat, informaţia este suprascrisă cu cea nouă.

Ultima acţiune este pentru dezinstalarea serviciului Web. La început lansează comanda de dezinstalare, după care şterge clasele din directorul `lib` al AXIS pentru a nu apărea probleme. De asemenea, dacă se foloseste Tomcat, decomentând ultimele linii se execută repornirea serverului AXIS.

4.4.1 Configurarea Tomcat pentru a putea fi accesat de Ant

Acest subcapitol prezintă, pas cu pas, acţiunile ce trebuie întreprinse pentru a permite utilitarului Ant să se conecteze la un server Tomcat şi să execute anumite acţiuni, cum ar fi restartarea, reîncărcarea şi ştergerea unei aplicaţii Web. Aceste informaţii sunt valabile numai în cazul folosirii serverului Jakarta Tomcat, versiunea 4.x.x (Catalina). În cazul altor servere sau altor versiuni, se recomandă citirea documentaţiei.

Pasul 1) Se copiază fişierul *catalina-ant.jar* din directorul TOMCAT_DIR/server/lib în directorul ANT_DIR/lib.

Pasul 2) În fişierul *TOMCAT_DIR/conf/tomcat-users.xml* se adaugă următoarele intrări:
```
<role rolename="manager"/>
```
şi
```
<user username="<myuser>" password="<mypwd>" roles="manager"/>
```

Pasul 3) Se reporneşte Tomcat şi se testează dacă utilizatorul are acces la locaţia http://¡tomcaturl¿:¡tomcatport¿/manager.

Pasul 4) Pentru a folosi aceste facilităţi trebuie decomentate intrările din build.xml referitoare la Tomcat.

Se recomandă consultarea documentaţiei serverului Tomcat referitoare la acest subiect. În mod special se recomandă citirea informaţiilor despre aplicaţia `Manager`, activarea acesteia putând duce la probleme de securitate.

Capitolul 5

Implementarea clientului TeleSoap manual

Până acum avem implementată aplicaţia de server. Pentru a vedea dacă a fost desfăşurată corect pe server, ne conectăm cu un browser la adresa *http://localhost:8080/axis/services/Telesoap?wsdl*. În caz de eroare, trebuie reinstalat serviciul TeleSoap conform paşilor prezentaţi anterior.

Să ştim că serviciul este desfăşurat corect este un pas înainte, dar nu este suficient. Cum putem şti că funcţionează. Singura modalitate de a realiza acest lucru este de a implementa o aplicaţie client care să se folosească de serviciul creat anterior.

Un client este realizat din mai multe module: modulul de interfaţă cu utilizatorul, modulul de persistenţă, modulul de organizare a datelor în memorie, modulul de comunicare cu serverul. Dintre toate aceste module, AXIS intervine doar în ultimele două. La organizarea datelor în memorie trebuie îndeplinite anumite cerinte (clasele ce trebuie transmise să fie JavaBean). Modulul de comunicare cu serverul este simplificat prin folosirea AXIS, nemaifiind nevoie de definirea unui protocol propriu şi de implementarea la nivel jos (low level) a acestuia.

Înainte de a începe implementarea modulelor clientului, este important de ştiut ce oferă AXIS programatorului pentru realizarea celor două module. El oferă două posibilităţi:

- *Implementarea manuală.* Apelarea la distanţă a procedurii se face manual. Adică programatorul are la dispozitie clasele de nivel înalt oferite de AXIS dar trebuie să completeze parametrii de apelare prin intermediul limbajului de programare, manual. Printre aceşti parametrii se numără: adresa Web a serviciului, numele metodei, maparea claselor ce vor fi transmise şi returnate. De asemenea parametrii de apel ai procedurii se setează manual.
  Această metodă poate fi folosită cu succes în cazul în care programatorul aplicaţiei client este si realizatorul serverului sau cel puţin cunoaşte bine structura aplicaţiei server. În acest caz programatorul are mai mult control asupra modului de apelare, dar volumul de lucru creşte simţitor.

- *Generarea automată a modulelor cu WSDL2Java.* Distribuţia AXIS oferă suport pentru tehnologia WSDL. Pentru a ajuta programatorul în dezvoltarea serviciilor Web, AXIS vine integrat cu două utilitare: *Java2WSDL* şi *WSDL2Java*. Cel din

urmă primeşte ca parametru un fişier de tip wsdl si generează cod Java care să faciliteze accesul la serviciul Web: clasele de date (JavaBean), interfeţele de comunicare implementarea acestora (stub). Astfel, programatorul nu mai trebuie să completeze setările pentru apelarea metodei la distanţă ci doar să instanţieze clasa de comunicare şi să execute apelul ca şi cum serviciul ar fi doar o clasă aflata pe acelaţi calculator.

În cele ce urmează voi prezenta implementarea modulelor de date şi de comunicaţie manual, folosindu-mă de clasele oferite de AXIS. Un punct foarte important de reţinut este organizarea funcţionalităţii. Se recomandă plasarea codului de comunicare intr-o singură clasă, care să exporte metodele respective. De asemenea clasele de date să fie plasate, pe cât cu putinţă, în acelaşi pachet, grupate pe tipul de informaţie pe care îl transmit.

5.1 Implementarea claselor de date

Aceste clase nu prezintă dificultate în implementare. Programatorul trebuie doar să respecte cerinţele JavaBean. În caz de nevoie poate îmbogăţi funcţionalitatea claselor, de exemplu adăugând metode de obţinere a valorii obiectului ca String HTML (toHTMLString() : String) sau chiar să suprascrie metoda toString.

În cazul TeleSoap, clasele `Persoana`, `Telefon` şi `Adresa` rămân la fel, putând să copiem codul acestora din implementarea aplicaţiei server.

În schimb în clasa `CriteriuCautare` survin unele schimbări. Pe client nu mai este necesară prezenţa metodei de validare a unei persoane, deoarece această acţiune este realizată de către server. În principiu această clasă ar putea să arate astfel :

Listing 5.1: Codul sursa al clasei client CriteriuCautare

```
1 package com.telesoap;
2
3 import com.telesoap.data.Persoana ;
4
5 public class CriteriuCautare extends Persoana
6 {
7     public CriteriuCautare()
8     {
9         // constructor implicit cerut de specificatiile JavaBeans
10    }
11 }
```

Asta este tot, după cum am zis şi la implementarea clasei pentru server. Restul funcţionalităţii este preluată de la clasa `Persoana` (câmpurile si metodele de încărcare şi obţinere a valorilor).

Pentru a demonstra că se poate face extindere de funcţionalitate, voi mai introduce un constructor, de data aceasta cu parametrii, pe baza cărora să se iniţializeze valorile atributelor. Constructorul arată astfel:

Listing 5.2: Nou constructor al clasei client CriteriuCautare

```
1     public CriteriuCautare( String nume, String prenume,
2         String strada, String numar, String oras, String judet,
3         String prefix, String telefon )
4     {
5         this.nume = nume ;
6         this.prenume = prenume ;
7 
8         this.adresa = new Adresa() ;
9         this.adresa.setStrada( strada ) ;
10        this.adresa.setNumar( numar ) ;
11        this.adresa.setOras( oras ) ;
12        this.adresa.setJudet( judet ) ;
13 
14        this.telefon = new Telefon() ;
15        this.telefon.setPrefix( prefix ) ;
16        this.telefon.setNumar( telefon ) ;
17    }
```

Acelaşi lucru se poate realiza şi cu celelalte clase de date. Atunci constructorul prezentat mai sus ar deveni mult mai elegant deoarece s-ar folosi de noii constructori ai claselor `Telefon` şi `Adresa`.

O ultimă precizare la adresa claselor de date: nu este neaparat nevoie ca clasa sa se numească la fel cu cea de pe server. De fapt în majoritatea cazurilor, programatorul nici nu ştie cum se numeşte clasa pe server (dacă nu este el cel ce a implementat aplicaţia server). Doar maparea clasei la numele calificat XML să fie făcută corect. Şi dacă tot am pomenit despre maparea clasei la un nume XML, să arăt cum se face implementarea comunicaţiei cu serverul.

5.2 Implementarea modulului de comunicare

În acest caz modulul de comunicare se rezumă doar la o clasă `TeleSoapRemote`. În cazul unor proiecte mai mari, acest modul poate cuprinde mai multe clase unele pentru conectare la diverse servicii Web, altele pentru organizarea şi refolosirea conexiunilor şamd.

Această clasă `TeleSoapRemote` expune două metode statice, fiecare din acestea având corespondentă o metodă expusă de serviciul Web. Aceste metode statice au aceeaşi semnătură cu cele de pe server, pentru a da impresia programatorului că de fapt lucrează pe acelaşi calculator. Acest tip de implementare nu este obligatoriu, dar este recomandat, datorită organizării codului si funcţionalităţii.

Listing 5.3: Codul sursă al clasei TeleSoapRemote

```
1 package com.telesoap;
2 
3 import java.util.Vector;
4 
5 import javax.xml.namespace.QName;
6 import javax.xml.rpc.ParameterMode;
```

```
7  import javax.xml.rpc.ServiceException;
8
9  import org.apache.axis.client.Call;
10 import org.apache.axis.client.Service;
11 import org.apache.axis.encoding.ser.BeanSerializerFactory;
12 import org.apache.axis.encoding.ser.BeanDeserializerFactory;
13
14 import com.telesoap.data.Adresa;
15 import com.telesoap.data.Persoana;
16 import com.telesoap.data.Telefon;
17
18 public class TeleSOAPRemote
19 {
20    public static final String SERVICE_URL =
21       "http://localhost:8000/axis/services/Telesoap" ;
22
23    public static final String SERVICE_NS =
24       "http://info.unitbv.ro/sample" ;
25
26    public static Persoana cautaDupaTelefon( Telefon telefon )
27       throws Exception
28    {
29       Call call = createAxisCall( "cautaDupaTelefon") ;
30       mapeazaObiectele( call ) ;
31
32       QName qn = new QName( SERVICE_NS, "Telefon" );
33       call.addParameter("telefon", qn, ParameterMode.IN);
34       call.setReturnClass( Persoana.class ) ;
35
36       return (Persoana) call.invoke( new Object[] { telefon } ) ;
37    }
38
39    public static Vector cautaDupaCriteriu( CriteriuCautare criteriu )
40       throws Exception
41    {
42       Call call = createAxisCall( "cautaDupaCriteriu") ;
43       mapeazaObiectele( call ) ;
44
45       QName qn = new QName( SERVICE_NS, "Criteriu" );
46       call.addParameter("criteriu", qn, ParameterMode.IN);
47       call.setReturnClass( Vector.class ) ;
48
49       return (Vector) call.invoke( new Object[] { criteriu } ) ;
50    }
51
52
53    private static Call createAxisCall( String method )
```

```
54          throws ServiceException
55      {
56          Service  service = new Service();
57          Call call = (Call) service.createCall();
58          call.setTargetEndpointAddress( SERVICE_URL ) ;
59          call.setOperationName( method ) ;
60          return call ;
61      }
62
63      private static void mapeazaObiectele( Call call )
64      {
65          QName qn = new QName( SERVICE_NS, "Persoana" );
66          call.registerTypeMapping(
67              Persoana.class, qn,
68              new BeanSerializerFactory(Persoana.class, qn),
69              new BeanDeserializerFactory(Persoana.class, qn)
70          );
71
72          qn = new QName( SERVICE_NS, "Adresa" );
73          call.registerTypeMapping(
74              Adresa.class, qn,
75              new BeanSerializerFactory(Adresa.class, qn),
76              new BeanDeserializerFactory(Adresa.class, qn)
77          );
78
79          qn = new QName( SERVICE_NS, "Telefon" );
80          call.registerTypeMapping(
81              Telefon.class, qn,
82              new BeanSerializerFactory(Telefon.class, qn),
83              new BeanDeserializerFactory(Telefon.class, qn)
84          );
85
86          qn = new QName( SERVICE_NS, "Criteriu" );
87          call.registerTypeMapping(
88              CriteriuCautare.class, qn,
89              new BeanSerializerFactory(CriteriuCautare.class, qn),
90              new BeanDeserializerFactory(CriteriuCautare.class, qn)
91          );
92
93      }
94
95  }
```

Pentru a realiza un apel de procedură la distanţă, AXIS oferă clase ajutătoare, care să ascundă elementele de nivel jos ale modulului de comunicaţii: protocolul de transport, protocolul de serializare şi deserializare şamd. În continuare voi detalia câteva dintre aceste clase.

5.2.1 Clasa org.apache.axis.client.Service

Clasa `Service` implementează funcţionalitatea cerută de interfaţa `Service` (pachetul `javax.xml.rpc` din biblioteca `jaxrpc.jar`). Prin intermediul ei se facilitează accesul programatorului la serviciul aflat la distanta.

- Poate fi folosită ca un intermediar intre aplicaţia client si serviciul Web.
- Obţine informatii despre un serviciu Web pe baza parametrilor trimişi în constructor.
- Setează parametrii necesari pentru conectarea la serviciul Web prin intermediul obiectelor de tip `Call`.

În cazul clasei `TeleSoapRemote`, obiectul de tip `Service` este folosit doar pentru creerea obiectelor de tip `Call`, cu ajutorul cărora se face apelul propriuzis. Acest lucru este realizat la liniile 56 (unde este creat un obiect de tip `Service`) şi 57 (unde cu ajutorul obiectului `service` programatorul obţine un obiect de tip apel).

Pentru a îmbunătăţi funcţionalitatea clasei `TeleSoapRemote`, se poate declara un atribut static de tip serviciu, care să fie initializat chiar la început. În acest caz nu ar mai fi nevoie de a crea, pentru fiecare apel, un nou obiect serviciu. Pentru a duce lucrurile un pic mai departe, se poate transmite acestui atribut, în momentul contructiei, adresa la care se află fişierul wsdl al serviciului. Avantajul ar fi că locaţia, anumite mapări cât şi încărcarea metodelor (obţinerea listei de metode de pe server, cu tot cu lista de parametrii) este făcută la început, o singură dată.

5.2.2 Clasa org.apache.axis.client.Call

Aceasta este una dintre cele mai importante clase oferite de biblioteca AXIS. Prin intermediul ei se realizează apelul propriuzis de metodă, într-un mod transparent pentru programator. Oferă trei constructori :

- **public** Call(Service service) crează un obiect apel şi completează valorile anumitor parametrii în funcţie de câtă informaţie este oferită de obiectul serviciu. Aceasta este versiunea cea mai des întâlnită. De asemenea obiectul apel mai poate fi obţinut cu ajutorul metodelor clasei serviciu (după cum am văzut în exemplul anterior).
- **public** Call(String url) crează un obiect apel şi setează locaţia la care să se facă apelul. Toate celelalte atribute trebuie setate de programator, în cod.
- **public** Call(java.net.URL url) crează un obiect apel şi setează locaţia la care să se facă apelul. Funcţionalitatea este aceeaşi cu a constructorului precedent, diferind doar tipul parametrului.

Un obiect de tip apel are ca rol organizarea tuturor informaţiilor necesare pentru a executa un singur apel de metodă la distanţă, cât şi executarea propriuzisă a apelului. Ciclul de viaţă pentru un astfel de obiect este următorul:

Pasul 1 Instanţierea obiectului. Se realizează cu ajutorul constructorilor sau a unui obiect de tip serviciu. (linia 57)

Pasul 2 Setarea atributelor referitoare la serviciul apelat. Aici intră setarea locaţiei serviciului (linia 58) cât şi informaţii de autentificare (contul utilizatorului şi parola), de transport (ce tip de transport să se folosească : setTransport) şamd.

Pasul 3 Realizarea mapărilor intre obiecte şi numele XML asociate acestora. Acest lucru este realizat de comanda

Listing 5.4:

```
1 public void registerTypeMapping( Class javaType, QName xmlType,
2 SerializerFactory sf, DeserializerFactory df )
```

Această metodă merită mai multă atenţie:

- Parametrul *javaType* specifică ce clasă va fi mapată
- Parametrul *xmlType* specifică la ce nume si namespace va fi mapată clasa javaType. De exemplu, clasa `Adresa` este mapată la numele Adresa şi namespaceul 'http://info.unitbv.ro/sample'.
- Parametrul *sf* specifică ce fabrică de serializori va fi folosită când este nevoie de trecerea din obiect în XML.
- Parametrul *df* specifică ce fabrică de deserializori va fi folosită când este nevoie de trecerea de la XML la obiect java.

Pentru a înţelege mai bine, iată cum foloseşte AXIS aceste informaţii de mapare: clientul are un parametru de tip Adresa. AXIS găseşte maparea respectivă şi obţine un serializor cu ajutorul fabricii de serializori specificată la mapare, după care trimite serializorului obiectul iar acesta îl transformă în XML. În momentul primirii unui tag care are specificat un anumit tip şi un anumit namespace, AXIS ca şi client localizează maparea, obţine un deserializor şi, prin intermediul acestuia obţine valoarea obiectului primit de la server.

Grăitoare în acest sens este metoda statică de mapare a tuturor obiectelor folosite la un apel. Aceasta se găseşte între liniile 63 şi 93.

Pasul 4 Specificarea numelui metodei de apelat cât şi a semnăturii acesteia, adăugând o listă de parametrii. Parametrii, în acest moment sunt daţi doar prin tipul lor, nu prin valoare. Acest lucru se realizează cu ajutorul metodelor Call.setOperationName, Call.addParameter şi Call.setReturnType (sau în unele cazuri se va folosi Call.setReturnClass). Numele şi modul în care sunt folosite în programul demonstrativ sunt elocvente, astfel că nu mai este nevoie de nici un fel de comentarii. Acest lucru se poate vedea la liniile 59, 32—34 şi 46—47.

Pasul 5 Executarea apelului se face folosind metoda
public Object invoke(Object[] params)
Aceasta execută apelul procedurii remote cu parametrii transmişi prin şirul de obiecte. Rezultatul este de tip `Object` astfel că trebuie transformat la primire. Liniile 36 şi 49 sunt elocvente în această privinţă.

5.2.3 Programul client

Programul client mai conţine şi module de grafică. Dar acesta nefăcând subiectul lucrării de faţă nu va fi prezentat. Pentru mai multe informaţii referitor la proiect, se recomandă parcurgerea etapelor de instalare pe server si rularea clientului. De asemenea, pentru a observa modul de transmitere a datelor se poate folosi un utilitar de ascultare a transmisiunilor prin intermediul TCP/IP.

Modulul de comunicare fiind complet separat de modulul de interfaţă, utilizatorul îşi poate contrui propriul program care să ofere interfaţa dorită şi de asemenea să se folosească de serviciul dezvoltat anterior.

Capitolul 6

Clientul TeleSoap cu ajutorul utilitarului WSDL2Java

În capitolul precedent am prezentat un mod de implementare a clientului. Acesta este recomandat atunci când programatorul are suficiente informaţii la dispoziţie privind structura serviciului cât şi modul de lucru al acestuia. Dar în majoritatea cazurilor, programatorul nu va avea la dispoziţie toate aceste informaţii decât prin prisma documentului wsdl al serviciului Web. În cazul programului TeleSoap, serviciul Web este de dimensiuni reduse astfel că programatorul poate descoperi structura sa prin simpla parcurgere a fisierului de descriere wsdl. Dar în cazul unor servicii mult mai mari (cum ar fi şi cazul aplicaţiei PITA, prezentată în partea următoare), programatorul aplicaţiei client ar întâmpina greutăţi foarte mari în descifrarea wsdl.

Adresându-se acestei probleme, echipa AXIS a inclus în distribuţia produsului două utilitare: *WSDL2Java* şi *Java2WSDL*. Primul dintre ele are ca rol crearea unei structuri de clase stub şi de date pe baza unui fişier WSDL, iar al doilea generează un fişier WSDL pe baza unor clase Java. Acest capitol va arăta cum poate fi folosit WSDL2Java pentru a realiza un client al serviciului TeleSoap.

6.1 WSDL2Java pe scurt

Utilitarul WSDL2Java este de fapt o clasă Java, conţinută în biblioteca `axis.jar`. Numele său calificat este: `org.apache.axis.wsdl.WSDL2Java`. Sintaxa de lansare în execuţie este următoarea:

```
java org.apache.axis.wsdl.WSDL2Java [options] WSDL-URI
```

Trebuie menţionat că acest program are nevoie de aceleaşi biblioteci cerute de AXIS ca şi client pentru a putea rula. De aceea variabila CLASSPATH trebuie să arate la fel ca în cazul execuţiei unui program client care foloseşte AXIS. O modalitate simplă de a realiza acest lucru o reprezintă modificarea fişierului `run.bat` sau adăugarea unui nou element `<target name="...">` în fişierul `build.xml`. Voi exemplifica aceste lucruri în subcapitolul următor.

Iată câteva opţiuni ale utilitarului WSDL2Java:

- *-h, –help.* Afişează ecranul de ajutor, conţinând lista completă de opţiuni şi explicaţii suplimentare pentru fiecare dintre ele.
- *-N, –NStoPkg ¡argument¿* . Conţine maparea între un spaţiu de nume şi un pachet. În caz că nu există o astfel de mapare pentru un spaţiu de nume, AXIS generează automat un pachet pe baza spaţiului de nume (spaţiul `http://x.y.com` va deveni pachetul `com.y.x`. Toate clasele aferente elementelor mapate la spaţiul de nume respectiv vor fi create în pachetul asociat. Argumentul are următoarea formă: `<spaţiu de nume>=<nume pachet>`.
- *-f, –fileNStoPkg ¡argument¿* . În cazul serviciilor mari poate exista un număr destul de mare de spaţii de nume, astfel că maparea prin linie de comandă devine greoaie. De aceea WSDL2Java oferă posibilitatea încărcării mapărilor dintr+un fişier separat. Această comandă specifică numele fişierului ca argument. Dacă această opţiune nu este prezentă, se va căuta fişierul implicit, numit `NStoPkg.properties` aflat în directorul curent. Intrările din acest fişier arată astfel: `<namespace>[\:<element>]=<nume pachet>`.
- *-p, –package ¡argument¿.* Specifică în ce pachet vor fi plasate toate clasele create. Opţiunile precedente nu trebuie să apară (-N şi -f).
- *-a, –all.* WSDL2Java va genera toate mapările existente în fişierul WSDL. Dacă această opţiune nu este prezentă, WSDL2Java generează doar clasele de date care apar ca parametrii sau sunt returnate de metodele serviciului.
- *-o, –output ¡argument¿.* Specifică directorul de bază în care vor fi plasate clasele şi pachetele generate. De exemplu opţiunea `--output src` va genera toate clasele în directorul `src`.

Pentru a obţine o listă completă cu opţiunile acestui utilitar se recomandă folosirea opţiunii `--help` sau, pentru mai multe informaţii, consultarea documentaţiei referitoare la acest produs (fişierul `AXIS_HOME\docs\reference.html`).

6.2 WSDL2Java pentru TeleSoap

Folosirea utilitarului WSDL2Java reprezintă primul pas în construirea aplicaţiei client, realizând nivelul de transfer de date pe marginea căruia se va dezvolta întreaga aplicaţie.

Pentru început voi crea un director `TeleSoapWSDLClient`. Acesta va conţine programul client realizat cu ajutorul utilitarului prezentat în paragrafele precedente. În acest director voi mai crea un director, numit `src`, care va conţine fişierele sursă ale programului, şi de asemenea un director `bin` care va conţine clasele compilate.

Următorul pas îl reprezintă realizarea unui fişier `build.xml` pentru compilarea şi rularea programului. Acesta arată la fel cu cel prezentat la crearea manuala a programului client, doar că va mai conţine un element suplimentar care va apela WSDL2Java pentru crearea

nivelului de comunicaţie pentru aplicaţia client. Având în vedere că restul fişierului arată la fel, voi prezenta doar acest nou element:

Listing 6.1: Elementele adăugate fişierului build.xml

```
1 <property name="NSTeleSoap"
2     value="http://info.unitbv.ro/sample" />
3 <property name="NSService"
4     value="http://localhost:8080/axis/services/Telesoap" />
5 <property name="WSuri"
6     value="http://localhost:8080/axis/services/Telesoap?wsdl" />
7
8 <target name="wsdl2java">
9    <java classname="org.apache.axis.wsdl.WSDL2Java"
10          classpathref="run.cp" fork="true" >
11       <arg value="--output" />
12       <arg value="src" />
13
14       <arg value="--NStoPkg" />
15       <arg value='"${NSTeleSoap}"="com.telesoap.remote"' />
16
17       <arg value="--NStoPkg" />
18         <arg value='"${NSService}"="com.telesoap.service"' />
19
20       <arg value="${WSuri}" />
21    </java>
22 </target>
```

Primele şase linii definesc proprietăţile: `NSTeleSoap` (spaţiul de nume al mapărilor claselor de date), `NSService` (spaţiul de nume al serviciului Web, se găseşte la începutul fişierului wsdl) şi `WSuri` (locaţia fişierului wsdl).

Acţiunea `wsdl2java` este definită între liniile 8 şi 22 şi constă în rularea utilitarului `WSDL2Java` cu următorii parametrii:

- *–output*. Liniile 11, 12. Specifică directorul de ieşire.
- *–NStoPkg*. Liniile 14, 15. Toate elementele aflate în spaţiul de nume pentru date vor fi plasate (ca şi clase) în pachetul cu numele `com.telesoap.remote`.
- *–NStoPkg*. Liniile 17, 18. Toate elementele aflate în spaţiul de nume al serviciului Web vor fi plasate (ca şi clase) în pachetul cu numele `com.telesoap.service`. Vom vedea în subcapitolul următor că acestea vor fi clasele folosite pentru a realiza apelul la distanţă.
- *Locaţia fişierului wsdl* (de descriere a serviciului).

În urma rulării acestei acţiuni vom obţine următoarea structură de directoare şi fişiere conţinute în directorul `src`:

```
+ com
    + telesoap
        + remote {pachetul com.telesoap.remote}
            Adresa.java
            Criteriu.java
            Persoana.java
            Telefon.java

        + service {pachetul com.telesoap.service}
            TelesoapSoapBindingStub.java
            TeleSoapWS.java
            TeleSoapWSService.java
            TeleSoapWSServiceLocator.java

        PersoanaNotFoundException.java {conţine clasa excepţie}
```

După cum se observă, nu mai este nevoie de crearea manuală a claselor de date, acestea fiind realizate automat de utilitarul WSDL2Java pe baza descrierii acestora în fişierul de configurare. În acest caz putem fi siguri de compatibilitatea claselor de pe client cu cele de pe server. Acum putem să editǎm aceste clase, adăugând noi metode, constructori şamd.

De asemenea clasele folosite pentru apelarea serviciului sunt generate automat. Acestea se găsesc, în cazul clientului TeleSoap, în pachetul `com.telesoap.service`. Descrierea, pe scurt, a fiecăreia, va fi făcută în paragrafele ce urmează.

6.3 Fişierele serviciu generate

Utilitarul WSDL2Java generează, pentru un serviciu Web, o structură de clase prin intermediul căreia să se poată face apelul la distanţă. Aceste clase respectă specificaţiile *JAX-RPC*. Se generează două tipuri de clase: clase de localizare şi clase de apel

Clasele de localizare sunt două la număr: interfaţa `TeleSoapWSService` care defineşte metodele de obţinere a unui stub valid şi clasa `TeleSoapWSServiceLocator` care implementează funcţionalitatea interfeţei de mai sus.

Clasele de apel mapează fiecare procedură la distanţă printr-o metodă simplă, cu aceeaşi semnătură. Interfaţa `TeleSoapWS` defineşte metodele care vor fi expuse de serviciul Web. Clasa `TelesoapBindingStub` implementează funcţionalitatea definită de interfaţa `TeleSoapWS` astfel: pentru fiecare metodă expusă se generează cod de mapare şi serializare a parametrilor, de transmitere a cererii către server şi de deserializare a răspunsului primit de la acesta. Programatorul nu va mai scrie cod de nivel jos, apelând doar metode cu semnătură identică cu cele de pe server.

Iată cum programatorul va executa un astfel de apel:

Pasul 1 Se construieşte un obiect de tip localizor. În cazul TeleSoap se foloseşte constructorul clasei `TeleSoapWSServiceLocator`.

Pasul 2 Se obţine o interfaţă de apel prin folosirea obiectului localizor. În cazul TeleSoap se obţine o interfaţă de tipul `TeleSoapWS`. Aceasta referă un obiect de tip `TelesoapBindingStub`, dar acest lucru nu este transparent programatorului.

Pasul 3 Se efectuează apelul de metodă dorit cu ajutorul interfeţei de apel obţinute la pasul anterior.

6.4 Clasa TeleSoapRemote

Deşi avem deja implementat nivelul de comunicare cu serverul, se recomandă păstrarea funcţionalităţii într-un singur loc. De aceea voi păstra clasa `TeleSoapRemote`. Ea va arăta astfel:

Listing 6.2: Clasa TeleSoapRemote pentru varianta WSDL2Java

```
1  package com.telesoap;
2
3  import java.net.URL;
4
5  import com.telesoap.remote.*;
6  import com.telesoap.service.TeleSoapWS;
7  import com.telesoap.service.TeleSoapWSServiceLocator;
8
9  public class TeleSOAPRemote
10 {
11    private static TeleSoapWS service;
12    public static String SERVICE_URL =
13       "http://localhost:8080/axis/services/Telesoap";
14
15    public static Persoana cautaDupaTelefon( Telefon telefon )
16       throws Exception
17    {
18       if( service == null )
19          throw new Exception( "Nu am localizat serviciul Web." );
20       else
21          return service.cautaDupaTelefon( telefon );
22    }
23
24    public static Persoana[] cautaDupaCriteriu( Criteriu criteriu )
25       throws Exception
26    {
27       if( service == null )
28          throw new Exception( "Nu am localizat serviciul Web." );
29       else
30          return service.cautaDupaCriteriu( criteriu );
31    }
32
```

```
33      static
34      {
35          TeleSoapWSServiceLocator locator = new TeleSoapWSServiceLocator() ;
36          try
37          {
38              service = locator.getTelesoap( new URL( SERVICE_URL ) ) ;
39          }
40          catch( Exception e )
41          {
42              service = null ;
43          }
44      }
45  }
```

Partea III

PITA
(Personal Intelligent Travel Assitant)

Capitolul 1

PITA pe scurt

1.1 În lumea reală ...

O mare parte din viaţă ne-o petrecem călătorind. Unele călătorii sunt simple şi uşor de dus la bun sfârşit, altele sunt complexe şi cu greu ne putem descurca. Unele sunt în interes personal, altele în interes de serviciu. Unele sunt scurte, intraurbane, altele sunt lungi şi necesită folosirea mai multor mijloace de transport.

În realitate o călătorie nu este un lucru simplu. Primul pas nici nu include o deplasare efectivă: trebuie să hotărâm ce rută vom urma. Dacă este vorba despre o călătorie lungă, acest pas poate cauza considerabile dureri de cap. Să presupunem că, după ore de căutări, planificări şi reogranizări, am rezolvat această primă problemă. Următorul pas ce trebuie întreprins este de multe ori chiar mai greu decât primul: executarea efectivă a călătoriei. Probabil că ştiţi despre ce vorbesc: multe staţii de schimbare a mijlocului de transport, căutarea noului punct de sorire (peron, linie), întârzieri, schimbări de plan şamd.

Chiar dacă stiţi cum să vă descurcaţi cu toate aceste probleme, uneori este mai uşor să aveţi pe cineva mai experimentat care să vă dea o mână de ajutor. Aţi salva timp preţios şi aţi evita numeroase probleme. Acesta este motivul pentru care Universitatea Tehnică din Delft a hotărât demararea unui proiect ce are ca rol rezolvarea acestui tip de probleme. Proiectul se numeşte *PITA* şi este acronimul de la *Personal Intelligent Travel Assitant* (Asistent Inteligent şi Personalizat de Călătorii).

1.2 Ce este PITA?

PITA este un proiect început de puţin timp. Are ca scop oferirea de asistenţă şi informaţii călătorului în cele două faze ale unei călătorii: planificarea şi execuţia călătoriei. Deci, în câteva cuvinte, este un proiect pentru călători, este un produs de gestionare a călătoriilor.

După cum am menţionat anterior, PITA este abrevierea pentru Personal Intelligent Travel Assistant. Să explic ce înseamnă aceşti termeni:

- *Personal (Personalizat)* deoarece PITA se focalizează pe persoanele ce îl folosesc.

Scopul proiectului este de a permite utilizatorilor să obţină informaţii cât mai uşor şi, de asemenea, să fie supravegheaţi cât timp efectuează o călătorie.

- *Intelligent (Inteligent)* datorită capabilităţilor acestui proiect. Va oferi doar informaţii de bună calitate, ţinând cont de cererea efectuată cât şi de setările şi profilul utilizatorului.
- *Travel (Călătorie)* deoarece este domeniul cu care PITA lucrează.
- *Assistant (Asistent)* deoarece PITA este un asistent care ajută utilizatorul. Pentru început, când acesta solicită rutele optime pentru o călătorie, iar apoi pe parcursul acesteia, anunţându-l de evenimente ce au loc (schimbarea mijlocului de transport, întârzieri şamd) .

Deşi îşi propune multe, acest proiect se află într-un stadiu incipient. Până în momentul de faţă nu există nici o implementare. Acest paragraf prezintă, pe scurt, evoluţia PITA. A început ca proiect de cercetare al Universităţii Tehnice din Delft. Dezvoltarea (sau mai bine zis cercetarea) a fost executată de studenţi ca lucrare pentru master sau pentru doctorat. Ei au realizat un număr de documente prin care detaliază problema, analizează ceea ce utilizatorul doreşte de la un astfel de produs, prezintă concepte de implementare şi propun diverse soluţii şi modularizări.

Stadiul curent al acestui proiect este *în dezvoltare*. Cu alte cuvinte, în acest moment există o echipă care se ocupă de studiile de caz, specificaţiile şi organizarea proiectului.

1.3 Cerinţele pentru PITA

Pentru a putea determina ce funcţionalităţi să implementeze, acest proiect a necesitat un studiu al elementelor cerute de eventualii utilizatori. Acesta s-a materializat într-un document detaliat, ce prezintă aşa numitele *requirements* ale proiectului. Iată cele mai importante idei din punctul de vedere al utilizatorului:

- Utilizatorul poate folosi *motorul de planificare* favorit. Programul va fi doar o interfaţă de comunicare cu un număr de Planificatoare De Călătorii (Route Planners).
- Utilizatorul poate *salva rutele* dorite şi să le marcheze pentru activare. Rutele pot fi multimodale (să conţină mai mult de un mijloc de transport).
- Utilizatorul are *propriul profil* (informaţii legate despre el). Acest profil poate fi folosit de motorul de căutare pentru a obţine informaţii cât mai potrivite pentru utilizatorul în cauză. parole.
- Accesul la sistemul PITA se face prin intermediul unei parole. Doar după acest pas utilizatorul are dreptul să acceseze informaţiile proprii (gestionarea profilului, a rutelor şamd).
- Sistemul PITA are ca rol *monitorizarea statică* a călătoriilor. Această monitorizare presupune şi transmiterea de informaţii către călător (stadiul călătoriei, nodurile de

schimbare şamd). Aceste informaţii trebuie transmise pe diverse tipuri de aparate: telefoane mobile, pagere, email şamd.

- Sistemul PITA are ca rol *monitorizarea dinamică* a călătoriilor. Cu alte cuvinte va monitoriza stadiul călătoriei cât şi legăturile viitoare. În cazul unei eventuale întârzieri sau altor evenimente nedorite sistemul PITA trebuie să recalculeze ruta optimă şi să execute o replanificare "din zbor" a călătoriei. Notificarea călătorului este obligatorie. În cazul unor aparate de comunicare potrivite, utilizatorului i se va prezenta posibilitatea de a alege noua rută.

Aceste idei specifică funcţionalităţile de bază necesare unui astfel de produs. Dar pe lângă acestea mai sunt necesare cerinţe cu privire la modul de implementare:

- Flexibilitatea în comunicarea cu planificatoarele externe.
- Adăugarea de noi moduri de transport recunoscute de PITA, cu efort minim.
- Adăugarea de referinţe către noi motoare de planificare.
- Adăugarea de referinte către noi site-uri de supraveghere dinamică a mijloacelor de transport.
- Determinarea profilului poate fi făcută static (utilizatorul introduce informaţii la cerere) dar poate fi în parte determinată în mod dinamic, prin analizarea deciziilor ce au fost luate de utilizator de-a lungul timpului.

La toate aceste cerinţe se mai adaugă cerinţele standard pentru designul şi implementarea oricărui sistem cu adevărat performant şi extensibil: modularizarea, buna structurare a informaţiei, optimizarea accesului şi transmiterii informaţiei şamd.

Capitolul 2

Soluţia PITA WS

2.1 Soluţii posibile

Având în vedere cerinţele enunţate în capitolul anterior, se disting mai multe modalităţi de a implementa acest proiect: crearea unei aplicaţii Web bazate pe scripturi pe server, crearea unui server şi protocol propriu de transmitere a datelor, folosirea unor protocoale RPC deja definite. sau chiar o combinaţie a acestor variante.

Aplicaţia Web cu scripturi pe server (server side scripting) este o soluţie Enterprise standard, tot mai folosită în ultima vreme. Ea presupune realizarea unui site Web. Paginile ce îl compun sunt generate dinamic, pe server, la fiecare apel. Această tehnologie oferă multe avantaje, printre care centralizarea codului pe server, folosirea unei structuri client-server, clientul nu mai trebuie implementat (pentru a accesa o astfel de aplicaţie este nevoie de un simplu browser de Web).

Limbajele cel mai des utilizate pentru realizarea de astfel de aplicaţii sunt: JSP (Java Server Pages, o extensie a limbajului Java pentru realizarea de scripturi pe server), PHP (limbaj simplu, cu sintaxă asemănătoare C, cu extensii pentru accesul la diverse tipuri de baze de date), ASP şi ASP .NET(soluţia Microsoft pentru scripturi pe server, presupune folosirea unui server IIS).

Crearea unui *Server şi protocol propriu* pentru accesarea datelor nu este o soluţie foarte bună. Ea presupune implementarea unui server care să preia cererile şi să genereze răspunsuri. Acest lucru presupune definirea unor mecanisme standard (păstrarea de sesiune, definirea unui nivel de transport şamd). Acestea, nefiind oficial acceptate, vor fi mai greu de folosit astfel că implementarea unui client pentru o astfel de aplicaţie va fi un adevărat chin pentru alţi dezvoltatori de programe. Avantajul îl reprezintă controlul total asupra serverului, dar balanţa este evident în defavoarea acestei soluţii.

Folosirea unor *protocoale PRC standard* este de asemenea o soluţie viabilă. În comparaţie cu aplicaţia Web, este necesar un volum mai mare de muncă deoarece implementarea clientului este separată (nu se va putea folosi un browser de Web). De asemenea presupune şi definirea unor mapări pentru transmiterea obiectelor (lucru realizat prin alegerea unui protocol).

Există mai multe tipuri de protocoale, printre care se remarcă: RMI-IIOP (Remote Method Invocation over IIOP, soluţia Java standard pentru RPC), CORBA (soluţie standard de

RPC, dar care oferă un API destul de complicat şi cu probleme de interoperabilitate), tehnologia COM/DCOM (soluţia Microsoft pentru programarea distribuită).

În afară de aceste tehnologii clasice, se poate alege dezvoltarea acestei aplicaţii sub forma unui serviciu Web complex. Acest lucru presupune folosirea unui protocol standard bazat pe XML, cel mai comun fiind SOAP (Simple Object Access Protocol). Această tehnică face subiectul lucrării de faţă, aşa că până acum ar trebui să ştiţi cam despre ce este vorba.

2.2 De ce servicii Web?

Având în vedere succesul de care se bucură tot mai mult serviciile Web, cât şi faptul că este o alegere potrivită acestui tip de aplicaţie (mărturie stau motivele enunţate în prima parte a acestei lucrări), am hotărât realizarea acestui proiect sub forma unui serviciu Web.

Pentru a conferi o mai mare flexibilitate produsului, am realizat logica efectivă sub forma unui modul separat (care nu are cunoştinţă despre serviciul Web) şi acestuia i-am ataşat o interfaţă de comunicare bazată pe SOAP (serviciu Web). Pentru a înţelege mai bine această idee, se recomandă parcurgerea secţiunii referitoare la structura proiectului.

Iată câteva din avantajele pe care le-am obţinut din alegerea acestei tehnologii:

- *Independenţa de limbaj, sistem de operare şi platformă.* Acest lucru presupune faptul că serverul poate comunica cu clienţi scrişi în diverse limbaje, rulând pe platforme diferite. Astfel un călător poate să îşi folosească o aplicaţie scrisă pentru telefonul său mobil pentru a accesa serviciul Web (să verifice desfăşurarea călătoriei, să interogheze agentul activ şamd).

- *Modularizarea codului* se obţine prin împărţirea serviciului în mai multe clase pe baza funcţionalităţii oferite, fiecare reprezentând un subservicu. Acest lucru va fi pe deplin clarificat după parcurgerea capitolului referitor la modulul de serviciu Web.

- *Dezvoltarea incrementală* este permisă datorită caracterului modularizat al unui serviciu Web. Astfel aplicaţia poate rula pe server, oferind o funcţionalitate redusă (de ex. doar partea de autentificare şi planificare de rute) urmând ca, în viitor să fie implementate şi componentele referitoare la agenţii de supraveghere.

- *Conlucrarea cu alte tehnologii Enterprise.* Se poate realiza o aplicaţie Web pe baza scripturilor pe server (JSP, ASP şamd) care să acceseze acest serviciu pentru a obţine informaţiile necesare. O astfel de aplicaţie va reprezenta doar stratul de afişare (de comunicare cu clientul), fiind asemănător unui client al aplicaţiei.

2.3 Specificaţiile

În urma activităţii de cercetare ce a avut loc pentru acest proiect au rezultat mai multe documente cu titlul de specificaţii. Acestea oferă informaţii despre cum ar trebui implementat şi ce funcţionalităţi oferă. Având în vedere că proiectul pe care l-am

implementat reprezintă o particularizare a problemei iniţiale, voi prezenta doar informaţiile legate de natura serviciului Web şi a funcţionalităţii implementate de acesta.

Iniţial proiectul PITA WS a avut ca scop doar definirea nivelului de transport şi a interfeţei oferite de sistem. Dar aceste elemente nu pot fi testate fără implementarea efectivă de la care să obţină informaţiile. Din această cauză scopul proiectului PITA WS a devenit: *implementarea unei versiuni preliminare a produsului, care să ofere funcţionalitatea de bază, să definească structura şi modularizarea proiectului şi să ofere o interfaţă logică de accesare a informaţiilor oferite.*

Proiectul se va folosi de protocolul SOAP pentru a realiza transmiterea datelor între aplicaţia client şi server. Pentru aceasta este necesară folosirea unor unelte speciale. Astfel, pentru server, acest produs necesită instalarea unui server de aplicaţii (precum Jakarta-Tomcat) şi serverului de SOAP Apache AXIS, cât şi a unui sistem de baze de date (recomandat PostgreSQL). Pentru a putea accesa serviciul Web, clientul are nevoie de o bibliotecă standard de SOAP compatibilă cu cea de pe server (SOAP 1.2).

Iată lista de funcţionalităţi ce sunt implementate pentru această versiune a sistemului PITA:

- *Autentificare utilizatorului* pe baza unui mecanism nume utilizatori şi parolă. Acest lucru împiedică accesul persoanelor neautorizate la informaţii personale ale unui client.
- *Profilul utilizatorului* conţine informaţii referitoare la utilizator: numele, prenumele, adresa de email folosită şi numărul de telefon la care poate fi contactat. Proprietarul profilului are drept de modificare asupra lui.
- *Planificarea rutelor* se face conform specificaţiilor originale, prin conectarea la un serviciu extern de planificare prin intermediul căruia se obţin informaţiile necesare. În momentul actual, nu este implementat un adaptor pentru vreun serviciu real, astfel că se foloseşte un serviciu `DUMMY` care generează rute aleatoare, pe baza informaţiei de intrare. Acesta are rolul de a testa celelalte componente ale sistemului PITA, cum ar fi agenţii şi interfaţa de gestionare a rutelor.
- *Gestionarea rutelor* se face prin intermediul unui subserviciu Web specializat. Utilizatorul are dreptul să salveze, să şteargă sau să modifice informaţii referitoare la rutele pe care le deţine.
- *Persistenţa datelor* reprezintă un punct cheie al acestei aplicaţii. Pentru aceasta se va folosi un sistem de baze de date relaţional (PostgreSQL). Este posibilă extinderea aplicaţiei la folosirea unor tehnologii precum EJB (Enterprise Java Beans) sau baze de date obiect orientate.
- *Agenţii PITA* sunt componente server ce urmăresc călătorul pe parcursul întregii sale călătorii. Ei vor oferi notificări referitor la diverse tipuri de evenimente. În momentul actual ei oferă informaţii despre staţia imediat următoare, despre ora la care va sosi trenul în nodul respectiv şi la ce oră va pleca. De asemenea transmit un mesaj special în momentul în care utilizatorul trebuie să schimbe mijloacele de transport.
- *Atenţionarea (notificarea) călătorilor* se poate face prin intermediul mai multor tehnologii: email, telefon celular şi pager. Aceste moduri sunt dependente de aparatură

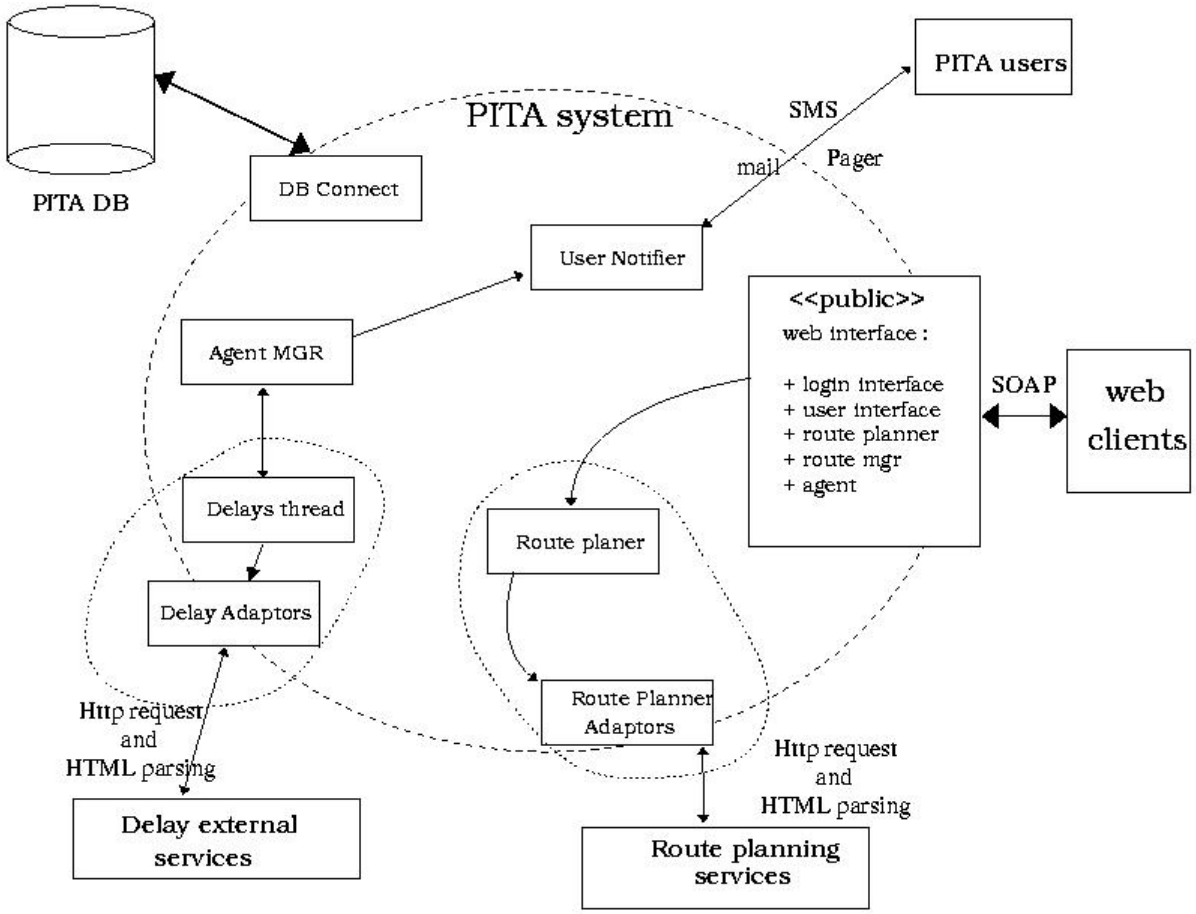

Figura 2.1: Diagrama de module pentru PITA

şi ţara pentru care este instalată aplicaţia PITA. Teoretic, cele trei elemente se reduc toate la transmiterea unui mesaj email la o anumită adresă.

- *Comunicarea cu agenţii activi* este posibilă şi prin intermediul unui subserviciu Web specializat. Acesta oferă metode prin intermediul cărora aplicaţia client poate interoga agentul în legătură cu stadiul călătoriei, cu mesajele transmise şi să obţină ruta asociată. Această funcţionalitate se poate dovedi utilă în cazul în care călătorul deţine posibilitatea de conectare la serviciul Web prin intermediul unor aparate mobile (mobile devices).

2.4 Structura proiectului

Pentru a crea un produs cât mai flexibil este necesară o modularizare cât mai riguroasă. Acest lucru include definirea modulelor ce vor exista, a funcţionalităţii pe care o vor implementa şi a modului în care acestea vor interacţiona între ele. Figura 2.1 prezintă diagrama modulelor pentru proiectul PITA WS.

Următoarele secţiuni vor explica diagrama de module şi vor prezenta motivele pentru care am ales să fac această împărţire. Dar, pentru început trebuie să precizez că există două

cazuri speciale, în care un modul este alcătuit din două componente: modulul de planificare a rutelor (alcătuit din planificatorul efectiv şi adaptorii pentru serviciile externe) şi modulul de întârzieri (alcătuit din threadul de verificare a întârzierilor şi adaptorii pentru accesarea serviciilor externe ce furnizează informaţii referitoare la întârzieri).

2.4.1 Interfaţa Web (Web Interface)

Din punctul de vedere al acestei lucrări, modulul de interfaţă Web este elementul central al proiectului. Acesta defineşte modul prin care utilizatorii vor accesa sistemul ce se află în spatele serviciului. Asta înseamnă metode cu semnături clare, succesiuni stabilite de apeluri (de ex. pentru crearea şi apoi salvarea unei rute, loginare şamd) şi un protocol uşor de depanat.

Având în vedere numărul mare de metode ce ar fi fost prezente într-un singur punct de acces, am ales să împart serviciul Web în patru subservicii, fiecare dintre acestea expunând anumite metode, pe baza funcţionalităţii pe care o vor oferi. Aceste subservicii se numesc: *pita:User*, *pita:Route*, *pita:planner* şi *pita:Agent*.

Pentru mai multe informaţii referitor la acest element al sistemului PITA, consultaţi capitolul ce prezintă acest modul.

2.4.2 Modulul de planificare a rutelor

Acest modul are ca rol crearea rutelor pe baza informaţiilor obţinute de la servicii externe specializate şi salvarea acestora legate de profilul utilizatorului ce le-a solicitat. Acest modul este foarte complex din mai multe motive:

- Trebuie să ofere o interfaţă prin intermediul căreia clientul execută operaţiuni complexe: execută interogări ale serviciilor externe, îşi salvează temporar rutele obţinute în urma căutărilor într-un buffer, reface căutările, salvează o anume rută. Această interfaţă presupune apelarea metodelor într-o ordine prestabilită şi de asemenea necesită o cantitate minimă de date ce trebuie transportată.

- Natura serviciilor externe de planificare este extrem de diversificată. Din păcate aceste sisteme nu oferă o interfaţă logică standard (cum ar fi un serviciu Web) ci prezintă informaţia sub forma unor documente HTML. Ceea ce face lucrul şi mai rău, aceste documente diferă de la un site la altul, astfel că interpretarea lor se face diferit.

- Modificarea interfeţei HTML a serviciilor externe duce la necesitatea de a reactualiza codul de interpretare în cazul unor minore schimbări ale documentului HTML.

- Complexitatea procesului de obţinere a rutelor pentru o anumită cerere este considerabilă. Astfel este necesară completarea mai multor pagini Web cu informaţii specifice site-ului, abia după aceea efectuându-se cererea de rute. Acest lucru presupune executarea unui număr de cereri HTTP (documente HTTP) echivalente execuţiei unei singure metode a serviciului.

Pentru a rezolva o parte din aceste probleme am recurs la un mecanism des întâlnit: realizarea unei clase adaptor pentru fiecare din serviciile externe recunoscute. Această clasă adaptor implementează o interfaţă standard definită de PITA, care conţine doar o metodă pentru căutare pe baza informaţiilor aferente. Dacă se cere o abordare mai complexă, atunci interfaţa poate fi redefinită pentru a conţine mai multe metode.

Obţinerea acestor adaptori se face prin intermediul unei fabrici (factory) iar iniţializarea şi încărcarea acestora se face (în momentul de faţă) de mână (hardcoded). Pentru o versiune mai complexă privind acest subiect, iniţializarea şi încărcarea acestor adaptori se va putea executa prin intermediul unor fişiere de configurare.

2.4.3 Modulul de agenţi

Acest modul conţine elementele prin intermediul cărora PITA supraveghează călătoriile în timp real.

Elementul central îl constituie clasa `Agent`. Aceasta este de tip Thread şi are atribuită o rută pe care o supraveghează. Această clasă se foloseşte de funcţionalitatea oferită de modulul de notificare pentru a trimite mesaje călătorului.

Pentru a porni agenţii, modulul se foloseşte de o altă clasă Thread denumită `AgentManager`. Aceasta verifică, la un anumit interval de timp, dacă există rute ce se activează. În caz afirmativ crează un obiect de tip `Agent` ce referă ruta în cauză, îl înregistrează şi îl lansează în execuţie. De asemenea această clasă permite accesarea agenţilor activi prin intermediul identificatorilor rutei pe care aceştia o supraveghează.

Modulul de agenţi implementează mare parte din funcţionalitatea cerută de subserviciul `pita:Agent`. Acest lucru se materializează în metodele conţinute de managerul de agenţi, prin intermediul cărora clasa serviciu poate comunica direct cu agenşii activi.

Mai multe despre conceptul de agent şi modalităţile de implementare în capitolul dedicat acestui subiect.

2.4.4 Modulul de notificare

Acest modul are ca rol încapsularea funcţionalităţii de transmitere de mesaje către diferite tipuri de aparate. Între acestea se includ telefoanele mobile (SMS), pagerele. De asemenea pot fi utilizate şi mesajele email.

Flexibilitatea este conceptul cheie pe care se bazează acest modul. El trebuie să permită adăugarea de noi tehnologii (precum trimiterea de mesaje prin voice mail) şi configurarea celor existente într-un mod foarte uşor de realizat. Pentru aceasta s-a folosit acelaşi concept de adaptor, încapsulând funcţionalitatea necesară în mai multe clase, fiecare pentru un tip de aparat în parte.

2.4.5 Modulul de întârzieri

Aceasta este doar o versiune preliminară a proiectului PITA, de aceea nu conţine implementări efective pentru funcţionalitatea de verificări dinamice ale rutelor supervizate de agenţi. Din această cauză conceptul de modul de întârzieri apare doar din punct de vedere teoretic, existând posibilitatea (pe baza designului flexibil) de a-l adăuga într-un viitor nu prea îndepărtat.

El se bazează tot pe servicii externe, care nu se folosesc de un protocol de comunicare standard. De aceea structura lui este asemănătoare modulului de planificare a rutelor, având în spate acelaşi concept de adaptor şi fabrică (factory).

2.4.6 Modulul de baze de date

Acest modul are ca scop generalizarea modului în care este accesată baza de date. El ajută programatorul în a obţine şi executa elemente standard de baze de date (precum obţinerea unei conexiuni JDBC sau obţinerea valorii următoare a unui câmp de tip `autoincrement`). Spre exemplu, în faza actuală aplicaţia PITA defineşte două moduri de obţinere a conexiunilor: crearea de conexiuni la cerere şi folosirea unui bazin de conexiuni (Connection Pool) pentru a optimiza timpul de acces. Această setare se face în fişierul de configurare al aplicaţiei.

De asemenea fişierul de configurare al aplicaţiei conţine mai multe informaţii referitoare la conexiunea la baza de date: clasa Driver folosită (deobicei un driver JDBC), numele şi parola prin care se accesează şi tipul de obţinere al conexiunilor. Din această cauză, dacă se doreşte migrarea de la un sistem de baze de date la altul, singurul lucru ce trebuie făcut este modificarea parametrilor respectivi în interiorul fişierului de configurare.

Modulul de baze de date conţine şi două clase auxiliare, prin intermediul cărora se poate crea structura bazei de date pentru un sistem de baze de date PostgreSQL şi de asemenea se poate testa accesibilitatea bazei de date.

Acestea sunt modulele de bază ale aplicaţiei PITA. Unele dintre ele vor fi prezentate mai în detaliu în capitolele ce urmează. Criteriul pe baza căruia au fost alese este simplu: importanţa modulului conform topicii acestei lucrări (şi în unele cazuri importanţa anumitor concepte de programare distribuită).

Capitolul 3

Organizarea datelor

Până acum am prezentat conceptele de bază referitoare la organizarea şi funcţionarea aplicaţiei, dar nu am pomenit nimic despre structurarea şi organizarea datelor folosite de PITA. Acestea sunt un element cheie, pe lângă modularizare. O structurare corectă a informaţiilor şi o granularizare corectă a interfeţei Web oferite pot duce la optimizarea accesului la serviciu: obţinerea datelor din bazele de date în timp scurt, transferul optim în reţea (transmiterea informaţiei strict necesare).

În cazul PITA, datele sunt împărţite în două categorii mari: datele ce descriu utilizatorul şi cele ce descriu rutele. Fiecare dintre aceste categorii defineşte una sau mai multe clase speciale, care pot fi serializate şi deserializate de un motor SOAP, cum este Apache AXIS.

3.1 Date despre utilizator

Programul trebuie să deţină anumite informaţii de bază despre utilizator. Acestea sunt grupate în clasa `UserData` (pachetul `com.pita.common.user`) şi reprezintă informaţii despre login (nume utilizator şi parolă), numele real al utilizatorului şi anumite informaţii de contact (adresa de email, numărul de telefon şamd).

Conform specificaţiilor originale, PITA trebuie să deţină informaţii detaliate referitor la preferinţele utilizatorului cu privire la planificarea rutelor şi de asemenea referitor la supravegherea şi trimiterea mesajelor. Aceste date nu sunt prezente în implementarea curentă, urmând a fi adăugate odată cu specificarea completă a tipului acestor informaţii. Ele sunt structurate în două clase distincte:

- Clasa `PlannerPrefs` conţine setările implicite ale clientului pentru planificarea unei rute. Aceste setări includ mijlocul favorit de transport, dorinţa de a obţine deobicei ruta cea mai scurtă sau cea mai ieftină. Poate fi folosit ca filtru pentru a afişa utilizatorului doar elementele care satisfac şi aceste cerinţe suplimentare.
- Clasa `TravelPrefs` conţine preferinţele implicite ale utilizatorului cu privire la modul în care va fi contactat de agenţii PITA în perioada călătoriei. Această clasă include informaţii despre modul de contactare, adresa de contactare (adresă mail, număr de

telefon, număr de pager şamd), despre frecvenţa mesajelor (la fiecare staţie, doar la nodurile în care călătorul trebuie să schimbe mijlocul de transport).

Aceste date sunt utile pentru a determina anumite preferinţe ale utilizatorului, setând implicit anumite câmpuri. Dar aceste componente nu reprezintă elemente vitale. De remarcat că ele pot fi generate automat în cazul folosirii PITA de mai multe ori, pe baza unor algoritmi de descoperire a asemănărilor dintre mai multe cereri.

3.2 Date despre rute

Ruta este informaţia cea mai des accesată de sistemul PITA. Ea este transmisă de planificatori, de agenţi şi chiar de programele client. Din cauza acestui lucru trebuie tratată cu foarte multă atenţie, o eventuală eroare conceptuală putând duce la îngreunarea accesării şi transmiterii datelor, ba chiar la blocări ale sistemului.

Un exemplu de rută poate fi studiat în figura 3.1.

O rută conţine informaţii de mai multe tipuri, acestea fiind grupate astfel:

- Clasa `Route` grupează informaţii referitoare la punctul de plecare, destinaţia, data de plecare. De asemenea conţine şi un nume prin intermediul căruia utilizatorul poate să o recunoască. Conţine o listă ordonată de obiecte `RouteItem`. Clasa `Route` este elementul central ce reprezintă ruta efectivă. De ea sunt ataşate celelalte clase.

- Clasa `RouteSearch` conţine informaţiile de căutare în urma cărora ruta a fost găsită. Această clasă este necesară pentru a putea reface procesul de căutare în cazul unor pierderi de date sau în eventualitatea folosirii verificărilor dinamice de traseu. Odată ruta creată, această informaţie nu mai poate fi modificată. Ea conţine identificatorul motorului de planificare folosit, punctul de pornire, punctul de sosire, punct intermediar obligatoriu (via), data de pornire sau data de sosire recomandate.

- Clasa `JourneySettings` conţine informaţii cerute de agentul PITA care va supraveghea călătoria. Ele pot fi modificate de utilizator înainte de a începe călătoria. Câmpurile clasei speicifă: modul de notificare şi adresa, intervalul de timp dinaintea unei staţii sau a unui nod de schimbare necesar pentru a face notificarea.

- Clasa `RouteItem` reprezintă o etapă a unei rute. Ea este caracterizată prin faptul că este întreprinsă cu acelaşi mijloc de transport. Ruta din figura 3.1 conţine două elemente `RouteItem`, unul reprezentând drumul de la Delft la Rotterdam iar următorul etapa de la Rotterdam la Amsterdam. O clasă `RouteItem` conţine un identificator PITA pentru tipul mijlocului de transport, identificatorul oficial al mijlocului de transport (numărul de înmatriculare) şi o listă ordonată cu obiecte `RouteItemStop` echivalente fiecărei staţii a unei astfel de etape.

- Clasa `RouteItemStop` este echivalenta unei staţii obişnuite dintr-o călătorie. Ea poate fi staţie intermediară sau nod de schimbare. În ambele cazuri este reprezentată la fel şi conţine numele staţiei, platforma (peronul), data de sosire sau plecare (după cum este oferită de planificatorul de rute). Conform exemplului din figura 3.1, toate

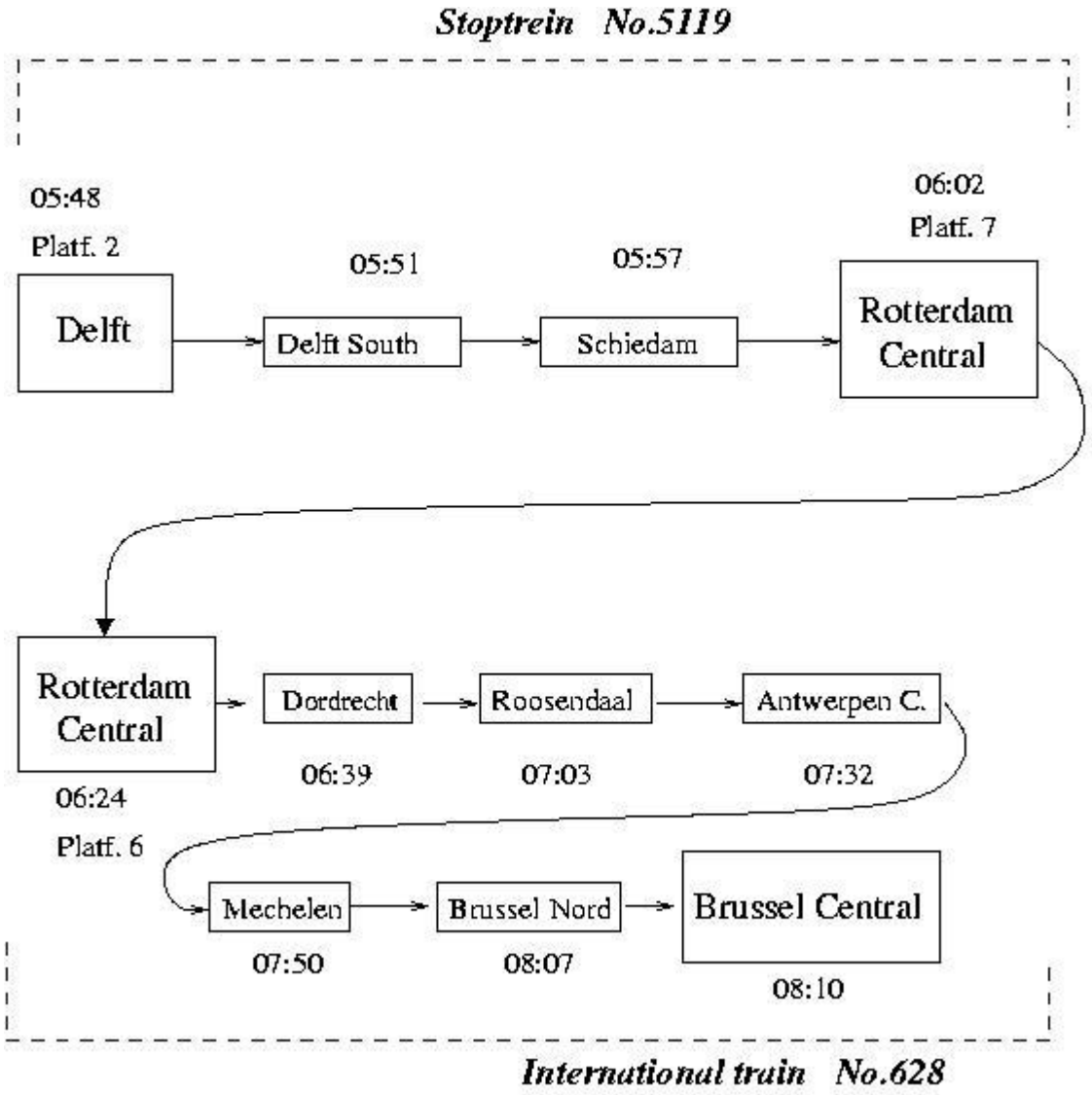

Figura 3.1: Exemplu de rută din Delft la Amsterdam

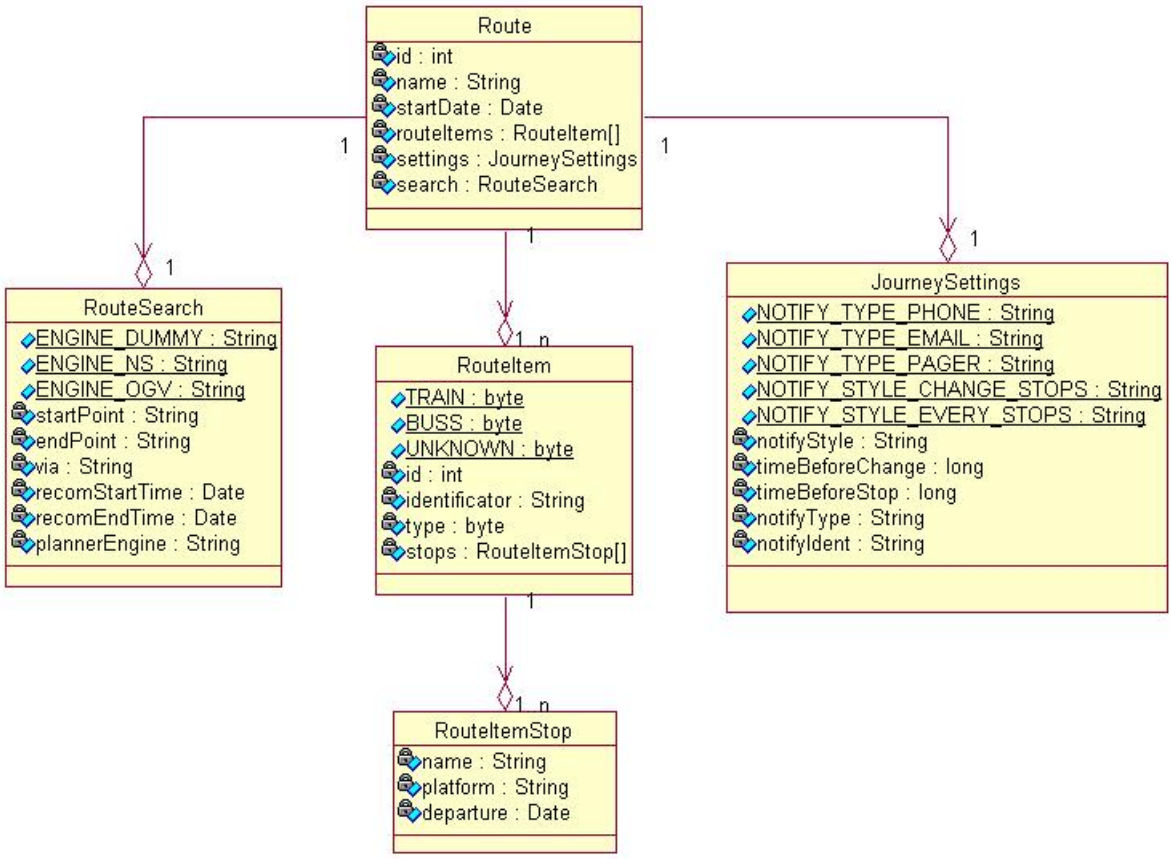

Figura 3.2: Diagrama UML a claselor pentru rutele PITA

elementele din dreptunghi reprezintă o staţie deci sunt încapsulate într-un obiect de tip `RouteItemStop`.

Pentru a înţelege mai bine modul în care este reprezentată intern o rută, se recomandă studierea diagramei UML 3.2.

Capitolul 4

Serviciile externe

Accesarea serviciilor externe este o problemă spinoasă în cazul PITA. Aceste servicii oferă informaţii diverse, de la planificarea de rute până la informaţii realtime despre starea unor mijloace de transport, întârzieri, blocaje pe autostradă şamd. Adevărata problemă nu o constituie natura informaţiilor transmise ci faptul că nu prezintă un mod standard de încapsulare.

Fiecare astfel de serviciu oferă utilizatorului direct o interfaţă HTML (mai mult sau mai puţin complicată). Călătorul poate executa interogări simple sau complexe completând câmpuri din forme HTML, apăsând butoane sau linkuri către alte pagini. Din păcate un program ca PITA nu va putea accesa aceste informaţii la fel de uşor ca un utilizator uman deoarece va trebui să încapsuleze toate informaţiile de transmis într-un mod asemănător unei forme Web şi, ceea ce este mai rău, să proceseze răspunsurile, care de multe ori vin încărcare de elemente HTML care nu sunt necesare.

Dezavantajele sunt evidente, dar nu există altă alternativă datorită faptului că aceste site-uri sunt independente de sistemul PITA:

- Executarea unei simple căutări poate constitui apeluri HTML multiple (forme pentru transmiterea informaţiei, linkuri pentru a detalia o rută). Din această cauză accesul din partea PITA va consuma resurse multiple ale serverului (conexiuni Web, posibil lente, încapsulare), şi deci în perioade lungi de răspuns.
- Parsarea răspunsurilor primite de la serviciile externe se face anevoios, cu consum de memorie, resurse şi implicit timp. De asemenea mesajele răspuns pot conţine elemente JavaScript, linkuri către imagini, elemente complet nefolositoare pentru PITA
- Modificarea, chiar şi minoră, a modului de organizare al unui astfel de document răspuns poate duce la zile sau chiar săptămâni de muncă în schimbarea codului de parsare şi transmitere a datelor. Acest lucru poate duce la dezavantaje majore pentru clienţii PITA, care vor fi privaţi de accesul la resursele unui planificator.
- Această complexitate excesivă este cauzatoare de erori, ridicând probabilitatea apariţiei unor probleme de intercomunicare cu serviciile externe.
- Fiecare serviciu are propriul mod de a organiza informaţia, propriul mod de afişare

şamd. Din această cauză, pentru fiecare din aceste servicii trebuie implementată o structură de clase care să permită conectarea şi folosirea informaţiilor.

Soluţiile pentru astfel de probleme sunt numeroase dar pot să vină numai din partea serviciilor externe folosite de PITA. Iată câteva dintre posibilităţile de remediere a acestor inconveniente:

- Folosirea unor protocoale standard de comunicare bazate pe codare ca stream: RMI, CORBA, DCOM.
- Implementarea unor protocoale RPC standard cu suport XML: SOAP, JAX-RPC.
- Definirea unor protocoale proprii, bazate pe XML.
- Utilizarea EJB (Enterprise Java Beans) cu Session Beans (aproximativ echivalente cu serviciile Web).

După cum probabil v-aţi dat seama, acest capitol nu conţine foarte multă informaţie referitoare la accesarea datelor de la serviciile Web externe, deoarece nu este prea mult de spus. De fapt acest capitol are ca rol sublinierea (cu exemple practice, clare) a importanţei folosirii unui protocol de transmitere a datelor, mai ales de către aceste firme ce oferă servicii ce pot fi folosite de alte aplicaţii.

Totuşi câteva informaţii sunt necesare, pentru a şti cum pot fi tratate aceste probleme. Iată un exemplu în care un utilizator PITA cere sistemului să îi prezinte un set de rute obţinute prin intermediul planificatorului NS (echivalentul CFR din Olanda). Acesta este prezentat în figura 4.1

Pasul 1 Sistemul PITA transmite cererea către Managerul de Adaptori

Pasul 2 Managerul determină serviciul ce trebuie accesat (pe baza informaţiilor oferite de client) şi transmite cererea mai departe la adaptorul serviciului respectiv

Pasul 3 Adaptorul preia cererea şi o transformă în formatul cunoscut de serviciul extern (deobicei HTML)

Pasul 4 Adaptorul execută apelul către serviciul extern. Acest pas poate reprezenta de fapt mai multe apeluri distincte, diferite, pentru a putea obţine toată informaţia necesară.

Pasul 5 Adaptorul primeşte de la serviciu informaţiile cerute.

Pasul 6 Adaptorul transformă informaţiile în obiecte recunoscute de sistemul PITA

Pasul 7 Adaptorul returnează managerului răspunsul primit

Pasul 8 Managerul de Adaptoare returnează sistemului răspunsul primit.

După cum se vede, procedeul este destul de complicat, adevăratele probleme fiind create de paşii 3, 4, 5 şi 6.

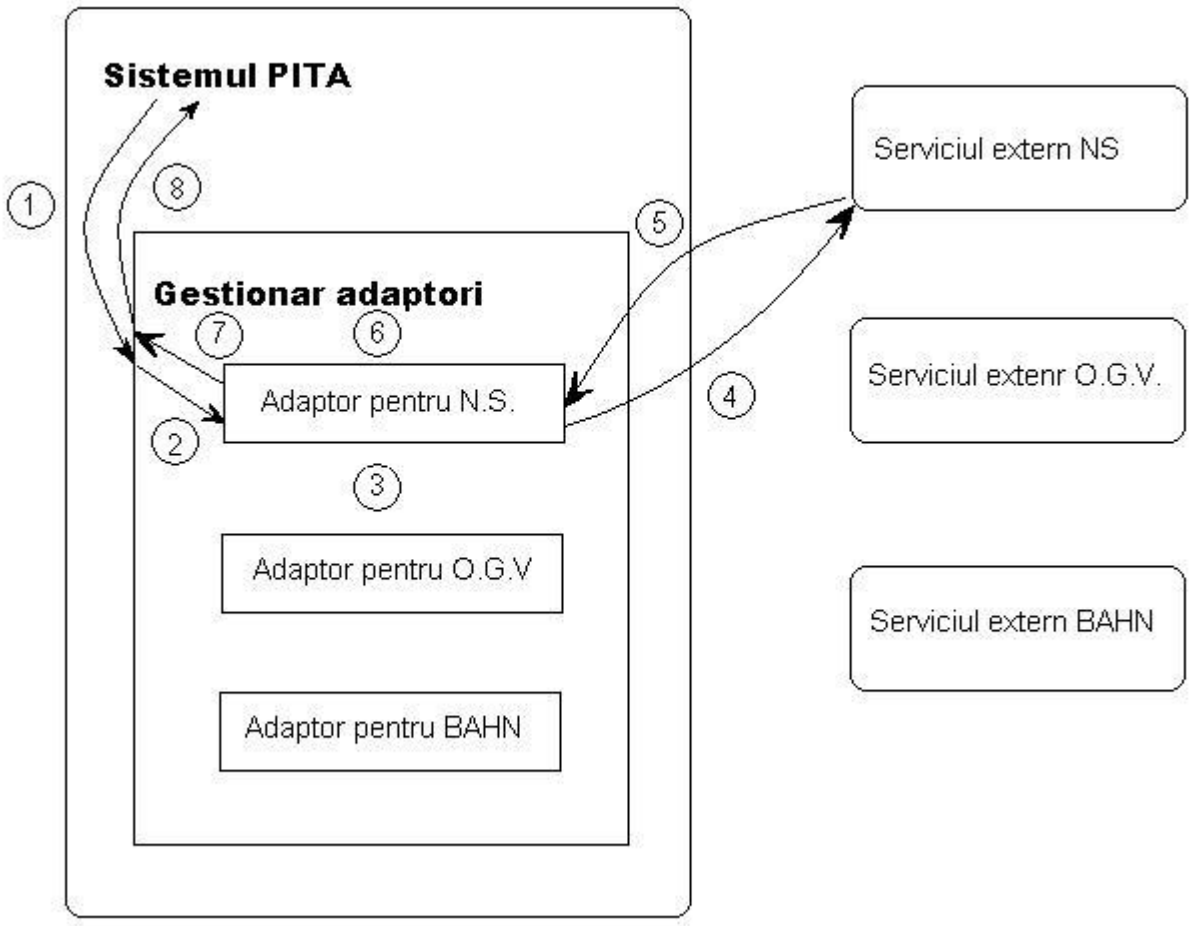

Figura 4.1: Apel de serviciu extern

Capitolul 5

Agenţi

Agenţii sunt un element important pe care îl tratează proiectul PITA. În prima parte a acestui capitol voi prezenta conceptul de agent PITA după care voi explica strategiile ce pot fi alese pentru a implementa astfel de componente. Ultima parte va conţine explicarea modulului de agenţi (structura şi funcţionalitatea acestuia).

5.1 Agentul PITA

Agentul PITA este soluţia propusă de acest proiect problemei supravegherii unei călătorii în curs de desfăşurare. Din punct de vedere general, agentul trebuie să îndeplinească următoarele roluri:

- Porneşte odată cu activarea rutei. Pornirea prealabilă a acestuia poate duce la degradarea performanţelor serverului.
- Transmite mesaje în una din situaţiile următoare: înainte de începerea călătoriei (pentru a avertiza călătorul), înainte de a ajunge la o staţie (doar dacă este setat să execute această comandă), înainte de un nod de schimbare al mijlocului de transport, înainte de îmbarcarea într-un nou mijloc de transport. Aceste mesaje conţin informaţii referitoare la data şi ora la care va ajunge călătorul la staţia respectivă, numele staţiei, peronul.
- Verifică, la intervale regulate de timp dacă există întârzieri pe parcurs. În caz afirmativ, dacă aceste întârzieri perturbă programul călătoriei, atunci anunţă utilizatorul şi calculează un nou curs optim care să evite problemele apărute. Această funcţionalitate nu este implementată la momentul de faţă deoarece implică prea multe elemente variabile.
- La terminarea rutei, se eliberează din memorie.

În cazul PITA, agentul este o particularizare a unui concept cu acelaşi nume mult mai general, ce îşi are rădăcinile în programarea distribuită.

5.2 Strategii de implementare a agentului PITA

Pe scurt, există două direcţii majore ce definesc moduri de implementare ale agentului PITA: implementarea centrată pe server şi implementarea distribuită (decentralizată).

5.2.1 Soluţia centralizată ...

... este una dintre cele mai comune implementări. Ideea este de a dezvolta o aplicaţie Web şi a o desfăşura pe un singur server central, la care se vor conecta toţi utilizatorii. În cele mai multe dintre cazuri, această soluţie este suficientă. Acest mod de lucru oferă anumite avantaje: dezvoltarea se va face rapid deoarece serverul oferă infrastructura pentru protocolul de transfer şi pentru tratarea sesiunilor de lucru. Întreţinerea unei astfel de aplicaţii este şi ea mult mai uşor de realizat.

În cazul PITA, această metodă presupune realizarea unui serviciu Web care va fi plasat pe un server central. Acest server va găzdui agenţii pe care serviciul Web îi va porni. Astfel, toată informaţia referitoare la aplicaţia PITA va fi localizată pe un singur server.

Dezavantajele însă sunt destul de importante:

- În cazul unei defecţiuni a serverului, întregul sistem PITA este oprit. Din acest motiv călătorii nu vor mai putea beneficia de nici un fel de funcţionalitate, incluzând notificări (schimbări de tren şamd). Această problemă poate fi evitată prin folosirea unor servere de siguranţă care vor porni în cazul defectării serverului principal. Mecanismul este complicat şi necesită transfer de sesiuni, conexiuni la baze de date şamd.
- Serverul poate fi suprasolicitat şi astfel performanţele acestuia să se degradeze. Este posibilă o astfel de situaţie în cazul rulării prea multor agenţi sau a apelării excesiv de des a unor metode costisitoare. Problema poate fi ameliorată folosind mecanisme de apelare balansată (Web balancing), de folosire a mai multor maşini server, fiecare cu rolul ei (de ex. una oferă serviciul Web, alta rulează agenţii, iar alta rulează motorul de baze de date).

5.2.2 Soluţia decentralizată

Dacă soluţia anterioară avea ca element central arhitectura client-server, această a doua posibilitate mai elimină din această legătură, încercând să plaseze unele componente în locaţii şi la adrese de accesare diferite. O astfel de soluţie presupune rularea efectivă a agentului pe client, astfel că serverul ar putea beneficia de o mai mare putere de lucru, scăpând de multe din atribuţiile consumatoare de resurse.

Acest mod de lucru are mult mai multe implicaţii. Dacă agentul va rula pe client, atunci modul de verificare dinamic se va putea face de la mai multe locaţii. Astfel se poate defini un server care să supravegheze doar trenurile dintr-o anumită zonă, alt server pentru trenurile din altă zonă şamd. De asemenea ar fi servere speciale pentru supravegherea traficului de pe autostradă şamd. Agentul va şti, în funcţie de locaţia călătorului, să se conecteze la serviciul corect.

Avantajul cel mai mare este constituit de flexibilitatea sistemului. Chiar dacă serverul PITA original are probleme şi este oprit, acesta furnizează doar elemente de planificare a rutei. Agenţii vor rula în continuare şi se vor folosi de serviciile aferente pentru a obţine informaţiile dorite. Dacă unul dintre serverele de informaţii are probleme, atunci agentul se poate conecta la un server de siguranţă sau un alt server care acoperă aceeaşi zonă. Iar în caz că nici acesta nu funcţionează, agentul nu va putea comunica informaţii clientului doar în zona respectivă, în momentul părăsirii acesteia conectându-se la alt server.

Deşi soluţia sună promiţător, apar numeroase probleme:

- Aceste servere ce oferă informaţii dinamice sau statice trebuie să implementeze acelaşi protocol de comunicare cu clientul şi aceeaşi interfaţă. În caz contrar, clientul ar avea de efectuat o muncă titanică de stabilire a comunicării.
- Întreţinerea unei astfel de reţele de servere este extrem de dificilă şi de costisitoare.
- Implementarea unor astfel de agenţi necesită un efort considerabil, asta deoarece pentru fiecare tip de aparat mobil (mobil device) se foloseşte un alt mod de programare.
- Aparatele mobile din generaţia actuală oferă suport scăzut, dacă nu chiar inexistent pentru rularea de threaduri în background. Din această cauză un agent va trebui să ruleze pe toată durata călătoriei, lucru deloc convenabil.
- Transferul de informaţii referitoare la rută către client este costisitor (şi resurse şi timp).

De asemenea este posibilă implementarea unei soluţii de compromis, care să încerce să se mapeze cât mai bine pe cerinţele problemei şi să profite de avantajele oferite de fiecare mod de lucru. Acest lucru este dificil de realizat şi poate duce la probleme mari de scalabilitate.

5.3 Modulul de agenţi

Datorită problemelor destul de mari ce apar la definirea unui sistem distribuit descentralizat, am ales folosirea primei soluţii pentru organizarea agenţilor PITA. Datorită acestei decizii aplicaţia PITA conţine un modul de agenţi.

Clasa `Agent` este elementul de bază al acestui modul. Ea este de tip thread şi are o singură rută asociată, cea pe care o supraveghează. Are responsabilitatea de a transmite mesaje de înştiinţare călătorului prin intermediul modulului de notificare. Ca particularitate, agentul se foloseşte doar de informaţie statică. De aceea el rulează astfel: trimite un mesaj conform stării în care se află călătoria, apoi calculează când va trebui să transmită următorul mesaj şi intră în starea de somn (sleep) până în momentul respectiv. Din această cauză serverul nu este prea solicitat, majoritatea timpului agentul fiind în stare de aşteptare.

Clasa `AgentManager` este tot de tip thread şi are ca rol gestionarea agenţilor activi. Ea verifică, la intervale prestabilite de timp, dacă există rute ce trebuiesc activate. În caz afirmativ, initializează obiecte de tip `Agent`, le asociază rutele respective şi le activează. Această clasă mai conţine şi metode de accesare a agenţilor activi pe care îi gestionează, cât

şi metode de control a acestor agenţi (expune o metodă de oprire a unui astfel de agent). Această clasă este folosită în mod special de subserviciul Web `pita:Agent`.

Capitolul 6

Serviciul Web

Interfaţa Web este punctul de legătură dintre aplicaţia client şi implementarea efectivă a funcţionalităţii pentru sistemul PITA. Acesta trebuie să fie singurul mod prin care un client se poate conecta la sistem, să efectueze cereri şi să primească răspunsuri.

Deci în primul rând acest modul trebuie să ofere metode pentru a reflecta cerinţele clientului. De asemenea aceste metode trebuie organizate în structuri predefinite, pentru o mai bună organizare în vederea accesării. Împărţirea metodelor pe grupuri se va face prin intermediul funcţionalităţii pe care o oferă fiecare.

Un alt element ce trebuie stabilit în momentul specificării unei interfeţe este dacă clientul are nevoie să păstreze anumite informaţii între două apeluri distincte către server. În alte cuvinte serviciul trebuie să fie cu stare (statefull) sau fără stare (stateless)? Răspunsul în cazul PITA este afirmativ: serverul trebuie să păstreze o stare pentru fiecare client ce se conectează.

Acest mecanism permite unui serviciu să adauge sau să modifice informaţii într-o variabilă sesiune. Din această cauză toate subserviciile PITA pot beneficia de această facilitate, fiind un mod elegant de a comunica între ele. De exemplu, pentru a specifica faptul că un client a trecut de faza de login, subserviciul `pita:User` (care pune la dispoziţie metoda de login) plasează în variabila sesiune identificatorul utilizatorului. Alte servicii pot verifica dacă utilizatorul este valid (a trecut de operaţia de login) prin verificarea existenţei variabilei identificator utilizator plasată în sesiune.

6.1 Organizarea serviciului Web

Din punct de vedere al organizării şi implementării, serviciul Web este împărţit în mai multe subservicii. Acestea au rolul de a încapsula cererile de un anumit tip într-o singură clasă. Datorită faptului că toate aceste subservicii au nevoie de anumite metode comune, am decis implementarea unei clase abstracte intitulate `PITAService`. Această clasă implementa metodele respective (de verificare dacă utilizatorul este loginat, de extragere a variabilelor din sesiune) dar nu putea fi instanţiată (restricţie intenţionată).

Serviciul Web este partiţionat în următoarele patru subservicii: `pita:User`, `pita:Route`,

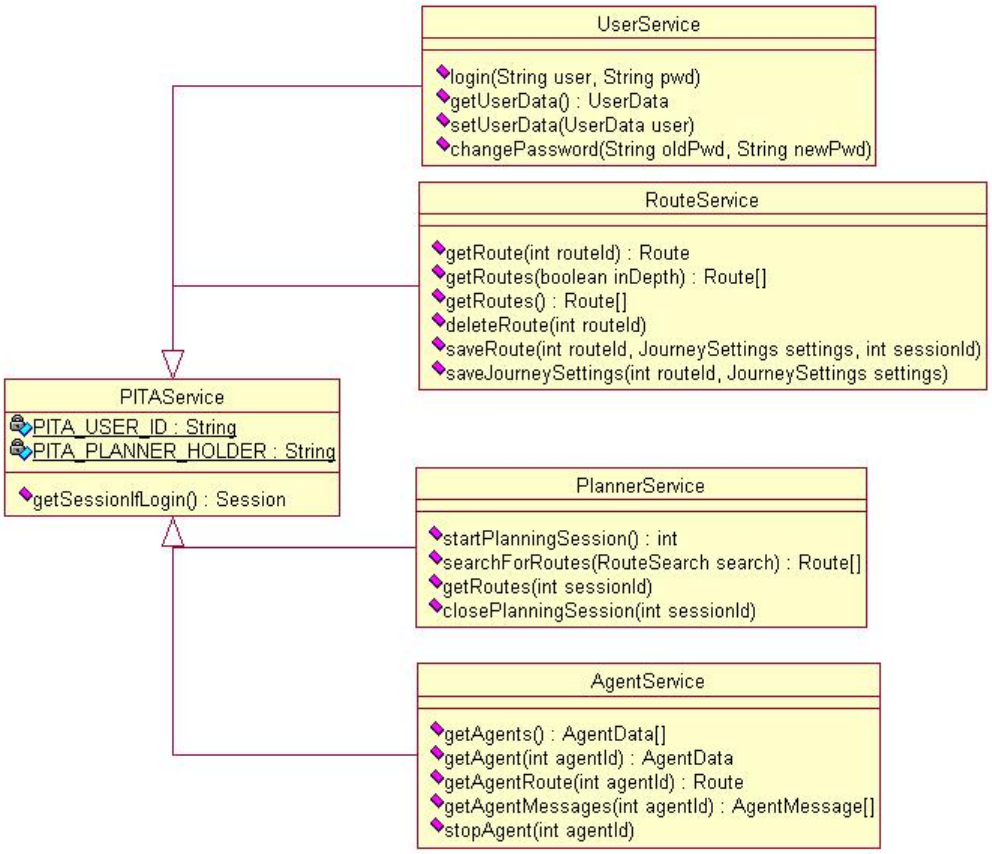

Figura 6.1: Diagrama claselor pentru interfata Web

`pita:Planner` şi `pita:Agent`. Fiecărui subserviciu îi corespunde o clasă ce extinde `PITAService`:`UserService`, `RouteService`, `PlannerService` şi `AgentService`. Diagrama UML ce prezintă această organizare este prezentată în figura 6.1.

6.2 pita:User

Acest serviciu este folosit pentru loginarea la sistemul PITA şi pentru gestionarea informaţiei referitoare la propriul profil. Pentru aceasta subserviciul `pita:User` defineşte următoarele metode:

- *public void login(String user, String pwd)* Această metodă este folosită pentru a executa activitatea de login. Dacă parola este corectă, metoda initializează variabila sesiune pentru a marca faptul că s-a executat autentificarea utilizatorului cu succes.
- *public UserData getUserData()* Această metodă returnează profilul utilizatorului.
- *public void setUserData(UserData user)* Această metodă are ca rol setarea profilului utilizatorului. După cum se vede, ea nu primeşte identificatorul utilizatorului ca parametru. Un utilizator are voie să acceseze şi să modifice doar propriul profil, astfel că metoda preia identificatorul din variabila sesiune.

- *public void changePassword(String oldPwd, String newPwd)* Această metodă schimbă parola utilizatorului ce o execută.

6.3 pita:Route

Prin intermediul serviciului `pita:Rout` aplicaţia client execută metode pentru gestionarea rutelor ce aparţin unui utilizator. De asemenea metodele accesează doar rutele utilizatorului ce execută apelul, astfel că nu este necesară transmiterea identificatorului utilizatorului, acesta fiind obţinut din variabila sesiune.

Subserviciul `pita:Route` defineşte următoarele metode:

- *public Route getRoute(int routeId)* Această metodă returnează ruta cu identificatorul specificat, doar dacă este deţinută de utilizatorul ce execută apelul. În caz contrar aruncă o excepţie.
- *public Vector getRoutes(boolean inDepth)* Returnează toate rutele deţinute de utilizatorul ce a executat apelul. Dacă parametrul are valoarea `false` atunci rutele vor conţine doar elementele din obiectul `Route` şi din obiectele `RouteSearch` şi `JourneySettings` asociate. Dacă parametrul are valoarea `true` atunci obiectul `Route` va conţine şi lista de `RouteItem` şi de `RouteItemStop`.
- *public Vector getRoutes()* este echivalentul metodei `getRoutes( true )`.
- *public void deleteRotue(int routeId)* Şterge ruta cu identificatorul specificat doar dacă aparţine utilizatorului ce efectuează apelul.
- *public int saveRoute(int routeId, JourneySettings settings, int sessionId)* Salvează ruta cu identificatorul specificat, cu setările `settings` aflat în sesiunea de planificare cu identificatorul egal cu `sessionId`. Pentru a înţelege modul de planificare, se recomandă citirea secţiunii următoare.
- *public int saveJourneySettings(int routeId, JourneySettings settings)* Asociază unei rute salvate (ce aparţine utilizatorului care a efectuat apelul) nişte setări pentru comunicarea din timpul călătoriei.

6.4 pita:Planner

Acest subserviciu este mai complicat datorită modului complex în care se face planificarea unei călătorii. Pentru a putea executa planificări de călătorii este necesară pornirea unei sesiuni de planificări. Acest lucru este realizat prin apelarea metodei `startPlanningSession`. Această metodă returnează un identificator pentru sesiunea de planificare nou creată. Acesta va fi utilizat ca parametru pentru toate metodele conţinute de acest serviciu. Odată terminată folosirea unei sesiuni de planificări, este recomandată închiderea acesteia pentru a elibera obiectele asociate ei.

Subserviciul `pita:Planner` defineşte următoarele metode:

- *public int startPlanningSession( )* Iniţializează o sesiune de planificare.
- *public Vector searchForRoutes(RouteSearch search, int sessionId)* Caută rute pe baza informaţiilor conţinute de obiectul `search`. Rezultatele sunt depuse în sesiunea de planificare cu identificatorul egal cu `sessionId`.
- *public Vector getRoutes(int sessionId)* Returnează o listă de Rute conţinute de sesiunea specificată.
- *public void closePlanningSession(int sessionId)* Eliberează o sesiune de planificare şi obiectele aferente acesteia.

6.5 pita:Agent

Acest subserviciu defineşte inferfaţa prin care se pot realiza interogări asupra agenţilor activi din sistemul PITA. Interogările pot fi executate de utilizatori doar asupra agenţilor proprii. Un agent este identificat prin intermediul identificatorului rutei pe care o supervizează.

Subserviciul `pita:Agent` defineşte următoarele metode:

- *public AgentData[] getAgents()* Returnează agenţii activi ai unui utilizator.
- *public AgentData getAgent(int agentId)* Returnează agentul cu identificatorul specificat.
- *public Route getAgentRoute(int agentId)* Returnează ruta agentului referit prin identificator.
- *public AgentMessage[] getAgentMessages(int agentId)* Returnează mesajele trimise de agent.
- *public void stopAgent(int agentId)* Opreşte agentul.

Capitolul 7

Puncte de extensie

7.1 Folosirea fişierelor de configurare

În stadiul curent aplicaţia PITA se foloseşte de fişiere de configurare pentru a citi informaţii referitoare la conectarea la baza de date, la modul de accesare al conexiunilor cu baza de date şi referitor la elementul de jurnal.

Dar acest mecanism poate fi extins pentru o mai bună reglare a mecanismelor şi la iniţializarea adaptorilor pentru diverse servicii externe de planificare a rutelor şi de verificare dinamică. De asemenea fişierele de configurare ar fi binevenite la iniţializarea modulului de notificare, pentru a configura adaptorii de mesaje.

Există două posibilităţi: folosirea fişierelor de proprietăţi (simplu de folosit, pe baza perechii cheie valoare) şi fişierelor XML (mai complicate, complexe, mai de greu de interpretat). Am optat pentru a doua variantă deoarece oferă mai multă flexibilitate şi informaţia este organizată ierarhic. Problema o constituie complexitatea cu care se face interpretarea fişierului XML.

7.2 Folosirea polimorfismului pentru rute

În momentul actual elementele unei rute (în mod special clasa `RouteItem`) conţine un câmp `type` care specifică tipul mijlocului de transport. Această metodă poate fi îmbunătăţită prin intermediul folosirii polimorfismului.

Cu alte cuvinte, pentru fiecare tip de transport ce este recunoscut de sistemul PITA se adaugă o clasă care extinde clasa `RouteItem` şi adaugă atributele specifice ei. În cazul de faţă clasa `RouteItem` ar fi fost abstractă şi ar fi existat două implementări ale acesteia: `TrainRouteItem` şi `BussRouteItem`.

Acelaşi mecanism se poate extinde şi la clasa `RouteItemStop`.

Avantajul ar fi folosirea programării obiect orientate, accesarea mult mai naturală a informaţiei. Dar în momentul de faţă nu există diferenţe notabile între tipu tren şi autobuz astfel că această tehnică ar întreuna metodele de codare SOAP prin adăugarea unor clase şi

mapări suplimentare.

7.3 Folosirea EJB sau OODB

Aceasta este una dintre cele mai radicale îmbunătăţiri ce pot fi aduse sistemului PITA. Folosirea uneia dintre aceste tehnologii va înlătura modulul de baze de date şi de asemenea codul ce se ocupă de transformarea dintr-un obiect `RecordSet` obţinut prin interogarea unei baze de date într-un obiect recunoscut de PITA.

Folosirea EJB crează un nivel de clase ce mapează baza de date. În realitate programatorul nu mai accesează direct baza de date ci prin intermediul unor metode de căutare. Acest lucru este benefic din mai multe motive: migrarea de la un tip de bază de date la altul se face uşor (prin intermediul setărilor EJB), accesul la elementele stocate este efectuat automat de motorul EJB programatorul primind doar obiectele valoare.

Folosirea unei baze de date obiect orientate sporeşte viteza aplicaţiei şi de asemenea timpul de dezvoltare. În acest caz nu mai este nevoie de dezvoltarea unui nivel intermediar de comunicare sau de untilizarea transformării din tabele relaţionale în obiecte (ceea ce făcea EJB în spatele scenei). Programatorul va stoca şi va obţine obiecte, fără alţi paşi intermediari. Un punct în plus îl reprezintă exprimarea mult mai naturală a relaţiilor dintre obiecte.

7.4 Verificări dinamice ale rutelor

Aceasta este o funcţionalitate ce va fi adăugată în viitor. Ea presupune conectarea la servicii externe de verificare a stării mijloacelor de transport şi executarea de interogări la intervale stabilite.

Sunt posibile numeroase optimizări, printre care folosirea unei clase intermediar la care să se înregistreze agenţii pentru a primi informaţii despre anumite mijloace de transport. De exemplu dacă avem mai mulţi agenţi care sunt interesaţi de acelaşi tren, sistemul nu va executa mai multe interogări în acelaşi timp ci va executa o singură interogare, prin intermediul acestei clase speciale. În cazul unor evenimente speciale, aceasta va anunţa agenţii de problemele apărute. Agentul devine un observator (listener).

Capitolul 8

Aplicaţia client

Realizarea unui serviciu Web fără a implementa o aplicaţie client poate duce la mari probleme în privinţa testării şi a corectitudinii implementării componentei server. De aceea am decis realizarea unei implementări model pentru un client al serviciului oferit de sistemul PITA. Această aplicaţie se numeste *PITA Client*.

Pentru a nu apărea probleme de interoperabilitate, am ales folosirea bibliotecii Apache AXIS pentru a realiza partea de transport pentru client. Acest lucru a determinat alegerea limbajului de programare: Java, deoarece era cel în care este scris AXIS. Ca interfaţă grafică am optat pentru tehnologia SWING, deoarece aceasta vine încorporată cu mediul de rulare Java (Java Runtime Environment).

Organizarea clientului este simplă. Primul ecran este cel de autentificare. Acesta preia numele utilizatorului şi parola şi încearcă să se conecteze la sistemul PITA. În caz de eşec (nu este disponibilă reţeaua, numele utilizatorului sau parola sunt greşite) se afişează un mesaj de eroare. În caz de autentificare reuşită, se deschide fereastra principală.

Fereastra principală conţine butoane (plasate în partea din stânga) prin intermediul cărora se pot accesa funcţionalităţile de bază ale serviciului Web: afişarea rutelor deţinute de utilizator, afişarea agenţilor activi, afişarea profilului. De asemenea permite accesarea unei componente de ajutor (help), care explică ecranele aplicaţiei şi modul în care acestea pot fi folosite.

Pentru a înţelege mai bine cum trebuie folosită această aplicaţie se recomandă pornirea ei şi folosirea uneltei de help.

Anexe

Anexa A

Lista de implementari SOAP

Această anexă prezintă un tabel cu câteva din implementările existente de SOAP. Ea este centrată pe limbajul Java dar oferă informaţii succinte şi despre alte biblioteci pentru Perl, Python şamd. Pentru o listă cât mai completă consultaţi tabelul de implementări aflat la adresa `http://www.soapware.org`.

Nume	**Limbaj**	**Producător**	**URL**
		Descriere	
Apache AXIS	*Java*	*Apache*	*http://xml.apache.org/axis*
	Server & Client, compatibil SOAP 1.2, free		
Apache SOAP	*Java*	*Apache*	*http://xml.apache.org/soap*
	Server & Client, compatibil SOAP 1.1, free		
	Nu se mai continuă dezvoltarea, doar reparae de bug		
kSoap	*Java*	*Enhydra*	*http://ksoap.enhydra.org*
	Client de dimensiuni reduse, SOAP 1.1, free		
	Pentru dispozitive mobile (phones, PDA)		
Server XMLBus	*Java*	*Iona*	*http://www.xmlbus.com*
	Server, compatibil SOAP 1.2		
Server WebLogic	*Java*	*BEA*	*???*
	Server, compatibil SOAP 1.2		
Server WebSphere	*Java*	*IBM*	*???*
	Server, compatibil SOAP 1.2		
Server JRun	*Java*	*Macromedia*	*???*
	Server , compatibil SOAP 1.2		
.NET	*C#,VB,C++*	.	*http://gotdotnet.com*
	Are nevoie de server (IIS), free		
	compatibil SOAP 1.2		
SOAP::Lite	*Perl*	*Paul Kulchenko*	*www.soaplite.com*
	Server , compatibil SOAP 1.2		
SOAP.py	*Python*	*Cayce Ullman*	*http://sourceforge.net/projects/...*
	Server , compatibil SOAP 1.2		*...pywebsvcs/*

Anexa B

Normalizarea bazei de date pentru aplicaţia PITA

Acestă anexă are ca scop detalierea organizării bazei de date folosite de aplicaţia PITA (Personal Intellignet Travel Assistant). Pentru aceasta mă voi folosi de tehnici standard şi de concepte caracteristice bazelor de date (entităţi, relaţii, diagrama entităţi-relaţii, normalizare, tranzacţii), elemente ce ar trebui să fie cunoscute cititorului.

Prima parte a capitolului va prezenta entităţile cu relaţiile aferente prin intermediul diagramei entităţi-relaţii (entity-relation) după care va detalia elementele componente ale sistemului PITA, din punctul de vedere al bazei de date. Următorul pas îl reprezintă demonstrarea faptului că baza de date este normală de forma a treia, cât şi descrierea unor particularităţi de design. Apoi voi descrie tranzacţiile ce intră în componenţa acestui sistem. La sfârşit voi prezenta un script de creere al bazei de date pentru sistemul PostgreSQL.

B.1 Entităţi şi relaţii

Pentru a înţelege mai bine structura bazei de date folosită de aplicaţia PITA se recomandă analizarea diagramei entity-relation din figura B.1. De asemenea secţiunile următoare detaliază entităţile (tipul atributelor, elemente speciale) precum şi organizarea relaţiilor.

B.1.1 Entităţile PITA

Acestea sunt entităţile PITA:

PITA_USER conţine informaţii referitoare la un utilizator PITA.

- ***ID : SERIAL*** este cheia primară a tabelei,
- *USERNAME : VARCHAR(30)*: numele de utilizator folosit pentru login,
- *PASSWORD : VARCHAR(30)*: parola folosită pentru login,
- *FIRST_NAME : VARCHAR(40)*: prenumele utilizatorului,

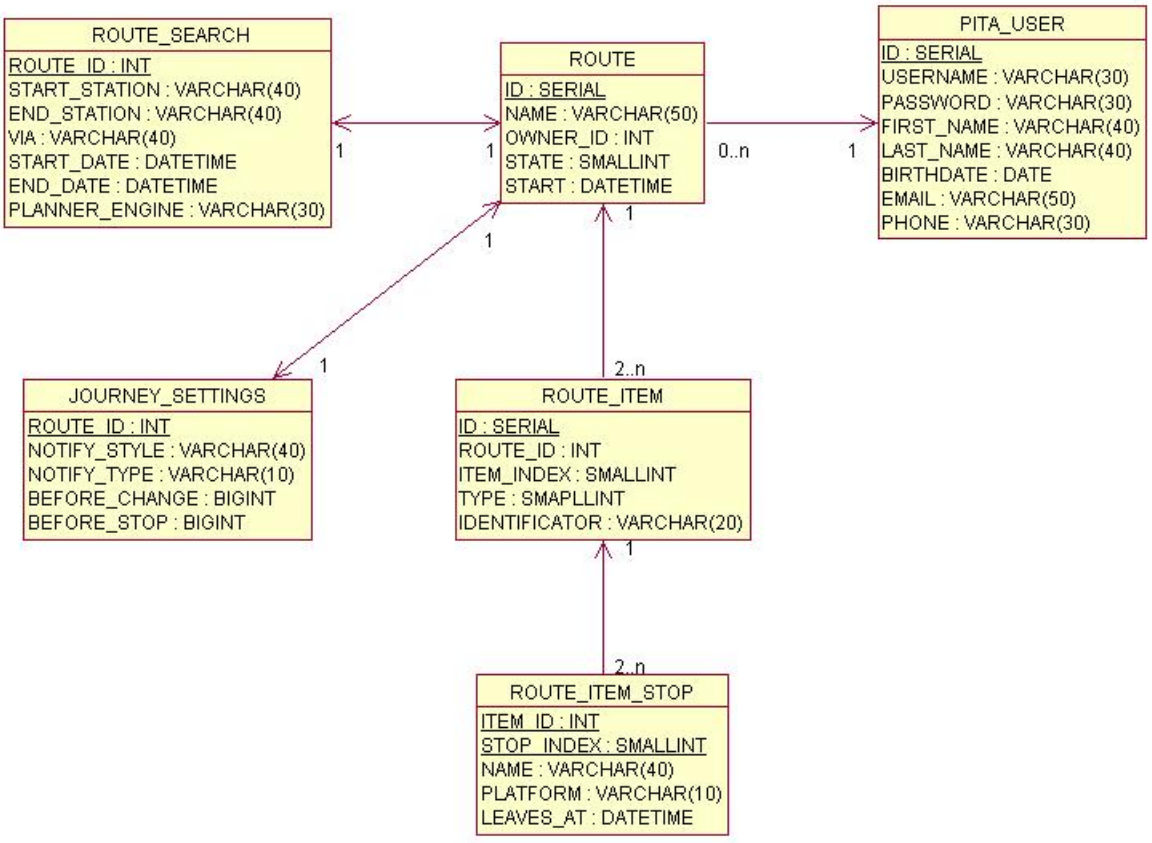

Figura B.1: Diagrama entity-relation pentru baza de date PITA

- *LAST_NAME : VARCHAR(40)*: numele de familie al utilizatorului,
- *BIRTHDATE : DATE*: data naşterii utilizatorului,
- *EMAIL : VARCHAR(50)*: adresa de mail a utilizatorului,
- *PHONE : VARCHAR(30)*: numărul de telefon al utilizatorului.

ROUTE conţine informaţii referitoare la o rută PITA. Este elementul central care face referire la celelalte tabele pentru a se putea obţine structura completă a rutei.

- ***ID : SERIAL*** este cheia primară a rutei,
- *NAME : VARCHAR(50)*: numele rutei, dat de utilizator la salvare,
- *OWNER_ID : INT FOREIGN KEY PITA_USER.ID* : identificatorul posesorului rutei,
- *STATE : SMALLINT*: starea rutei (va fi executată, este în derulare, a fost derulată)
- *START : DATETIME*: ora de start a călătoriei pe ruta în cauză.

ROUTE_SEARCH conţine informaţii oferite de utilizator în procesul de căutare al rutei referite. În acest caz cheia primară a acestei tabele este şi cheie străină (FOREIGN KEY), având aceeaşi valoare cu identificatorul rutei referite(cheia primară). Voi menţiona la sfârşitul acestei secţiuni de ce această entitate nu a fost înglobată direct în entitatea `Route`.

- ***ROUTE_ID : INT FOREIGN KEY ROUTE.ID*** este cheia primară a căutării rutei şi este egală cu identificatorul rutei.
- *START_STATION : VARCHAR(40)*: staţia de pornire a rutei
- *END_STATION : VARCHAR(40)*: staţia de oprire a rutei
- *VIA : VARCHAR(40)*: staţia intermediară obligatorie a rutei
- *START_DATE : DATETIME*: data în jurul căreia utilizatorul cere începerea călătoriei. Acest câmp conţine valoarea introdusă la căutare şi este foarte probabil să fie diferit de valoarea reală de începere a călătoriei (conţinută de `Route.START`.
- *END_DATE : DATETIME*: data în jurul căreia să se termine călătoria. Acest câmp conţine valoarea introdusă de utilizator la căutare şi este foarte probabil să fie diferit de data reală la care se va termina călătoria.
- *PLANNER_ENGINE : VARCHAR(30)*: identificatorul motorului extern de planificare. Acesta este recunoscut intern de aplicaţia PITA.

JOURNEY_SETTINGS conţine informaţii necesare agentului PITA în momentul activării (cum şi la ce număr să îl contacteze pe călător, să anunţe doar staţiile de schimbare sau toate staţiile şi cu cât timp înainte şamd).

- ***ROUTE_ID : INT FOREIGN KEY ROUTE.ID*** este cheia primară a setărilor călătoriei şi este egală cu identificatorul rutei.
- *NOTIFY_STYLE : VARCHAR(40)*: adresa la care va fi apelat călătorul (adresă de mail, număr de telefon, în funcţie de valoarea conţinută de câmpul `NOTIFY\_TYPE`).

- *NOTIFY_TYPE : VARCHAR(10)*: tehnologia prin intermediul căreia se anunţă călătorul (telefon mobil, pager, email).
- *BEFORE_CHANGE : BIGINT*: cu cât timp (milisecunde) înainte de a se ajunge la un nod de schimb să fie anunţat călătorul.
- *BEFORE_STOP : BIGINT*: cu cât timp (milisecunde) înainte de a se ajunge la o staţie (nu nod de schimb) să fie anunţat călătorul.

ROUTE_ITEM sunt componentele principale ale unei rute. Un `RouteItem` este echivalentul unei etape dintr-o călătorie, etapă pe parcursul căreia s-a folosit un singur mijloc de transport. O rută conţine unul sau mai multe `RouteItem`.

- ***ID : SERIAL*** este cheia primară a `RouteItem`.
- *ROUTE_ID : INT FOREIGN KEY ROUTE.ID* este cheia străină şi referenţiază părintele acestui `RouteItem`, adică ruta ce îl conţine.
- *ITEM_INDEX : SMALLINT*: a câta etapă a călătoriei este. Ruta conţine o listă ordonată de etape, de aceea pentru stocare este folosit acest câmp.
- *TYPE : SMALLINT*: modul de transport ales (tren, autobuz şamd).
- *IDENTIFICATOR : VARCHAR(20)*: identificatorul mijlocului de transport. Însemnătatea sa depinde de valoarea câmpului `TYPE` (dacă este vorba de tren atunci conţine numărul trenului, dacă este vorba de autobuz conţine numărul de înmatriculare al acestuia şamd).

ROUTE_ITEM_STOP sunt componentele atomice ale unei rute. Acestea sunt echivalentul staţiilor (de tren, de autobuz şamd). Un `RouteItem` conţine două sau mai multe `RouteItemStop`.

- ***STOP_INDEX : SMALLINT*** este parte din cheia primară a acestui tabel. O etapă (`RouteItem`) conţine staţiile sub forma unei liste ordonate iar acest câmp conţine indexul staţiei.
- ***ITEM_ID : INT FOREIGN KEY ROUTE_ITEM.ID*** este parte din cheia primară şi este cheie străină ce referenţiază etapa părinte (cea care conţine această oprire).
- *NAME : VARCHAR(40)*: numele staţiei (sau opririi).
- *PLATFORM : VARCHAR(10)*: peronul la care se ajunge.
- *LEAVES_AT: DATETIME*: data la care se pleacă de la oprirea respectivă.

B.1.2 Precizări suplimentare

Pe parcursul structurării acestei baze de date am avut de luat decizii referitoare la anumite componente ce ar putea să strice această organizare. Aceste decizii, chiar dacă uneori inoportune, s-au bazat pe următoarele criterii: *modularitatea* (tabelele să fie împărţite conform informaţiei pe care o conţin), *optimizarea accesului* (am recurs la redundanţă pentru a evita interogări constisitoare ale bazei de date). Această sesiune are ca scop prezentarea acestor 'anomalii' şi a motivelor pentru care le-am introdus în designul bazei de date.

Tabela `Route` conţine câmpul `Start` care reprezintă data şi timpul real la care se va începe călătoria. Este echivalent cu valoarea datei din prima staţie a rutei (adică a primului element `RouteItemStop` din primul element `RouteItem` conţinut de rută). Am ales plasarea unui câmp ce duplifică această informaţie în interiorul tabelei `Route` deoarece:

- Obţinerea datei de început a rutei este lentă şi necesită putere de procesare ridicată (prin intermediul celor trei tabele). Aplicaţia PITA foloseşte des acest element astfel că executarea unei astfel de interogări trebuie să fie uşor de executat (verificarea rutelor care se activează se face o dată pe minut).
- Odată ruta salvată, aceasta nu va mai suferi modificări (ea sau elementele componente: `RouteItem` şi `RouteItemStop`). Din această cauză nu există posibilitatea recalculării acestui câmp, informaţia stocată fiind mereu corectă.

Tabelele `RouteSearch` şi `JourneySettings` sunt în relaţie *unu la unu* cu tabela `Route`. Câmpurile acestora pot fi adăugate tabelei de rute fără a cauza probleme în organizarea bazei de date. Am evitat acest lucru din cauza următoarelor considerente:

- Modularizarea datelor. Datele referitoarea la rută sunt păstrate în tabela `Route`. Datele referitoare la căutare sunt păstrate în tabela `ROUTE_SEARCH` iar datele referitoare la setările agentului sunt păstrate în tabela `JOURNEY_SETTINGS`.
- Elementele din tabelele `ROUTE` şi `ROUTE_SEARCH` nu vor fi modificate. Odată salvată o rută, informaţia despre aceasta nu va fi modificată. Spre deosebire de aceste două componente, elementele din tabela `JOURNEY_SETTINGS` pot fi modificate deoarece ele reprezintă setări ale agentului la care utilizatorul are acces. Această împărţire a datelor este naturală şi permite adăugarea unui nivel de securitate suplimentar pentru a împiedica modificarea elementelor din cele două tabele.
- Această abordare oferă mai multe posibilităţi de extindere a sistemului PITA, permiţând modificarea cardinalităţii relaţiilor dintre tabelul `ROUTE` şi celelalte două tabele (`ROUTE_SEARCH` şi `JOURNEY_SETTINGS`).

Având în vedere relaţia de unu la unu dintre tabelul `ROUTE` şi tabelul `ROUTE_SEARCH`, am decis să folosesc identificatorul rutei ca şi cheie primară pentru informaţiile de căutare asociate rutei. Acest lucru asigură o legătură unu la unu sigură şi unicitatea cheii primare alese. Astfel am evitat folosirea unui câmp identificator suplimentar. Acelaşi raţionament l-am folosit şi la designul tabelei `JOURNEY_SETTINGS`.

Referitor la tabela `ROUTE_ITEM`, există o cheie candidat formată din două câmpuri: cel de indexare şi referinţa către ruta părinte. Am ales să introduc un nou câmp identificator (`ID`) deoarece elementele acestei tabele trebuie referenţiate de cele din tabela `ROUTE_ITEM_STOP` (referenţierea printr-o cheie străină compusă este extrem de complicată, dacă nu chiar imposibilă în cazul unor sisteme de baze de date).

B.1.3 Relaţiile între entităţile PITA

În urma identificării relaţiilor şi determinării cardinalităţii şi a participării, am sintetizat informaţia în următorul tabel:

Număr	Entitate	Relaţie	Entitate	Card.	Participarea	
					Directă	Inversă
1	User	deţine	Route	1 : N	Parţială	Totală
2	Route	are	Route_Search	1 : 1	Parţială	Totală
3	Route	are	Journey_Settings	1 : 1	Parţială	Totală
4	Route	are	RouteItem	1 : N	Totală	Totală
5	RouteItem	are	RouteItemStop	1 : N	Totală	Totală

Mai detaliat, fiecare relaţie arată astfel:

Relaţia nr. 1 Un utilizator poate deţine mai multe rute. O rută are un singur posesor. Există utilizatori care nu deţin rute dar rutele trebuie să aibă întotdeauna un proprietar (utilizator valid). Integritatea relaţională:

- ON DELETE CASCADE (la ştergerea utilizatorului se şterg toate rutele acestuia)
- ON MODIFY CASCADE (la modificarea identificatorului utilizatorului se modifică şi câmpul cheie străină al rutei)

Relaţia nr. 2 O rută poate avea asociat un element `Route_Search` dar nu este obligatoriu. Elementul de căutare, din contră, trebuie să fie ataşat de o rută. Integritatea relaţională:

- ON DELETE CASCADE (la ştergerea rutei se şterge si informaţia de căutare)
- ON MODIFY CASCADE (la modificarea identificatorului rutei se modifică şi câmpul cheie străină al informaţiei de căutare)

Relaţia nr. 3 O rută poate avea asociat un element `Journey_Settings` dar nu este obligatoriu. Elementul de setări, din contră, trebuie să fie ataşat de o rută. Integritatea relaţională:

- ON DELETE CASCADE (la ştergerea rutei se şterg şi setările pentru rută)
- ON MODIFY CASCADE (la modificarea identificatorului rutei se modifică şi câmpul cheie străină al setărilor)

Relaţia nr. 4 O rută trebuie să aibă asociat cel puţin un element `RouteItem`. De asemenea un element `RouteItem` trebuie să facă parte dintr-o rută. Integritatea relaţională:

- ON DELETE CASCADE (la ştergerea rutei se sterg şi toate etapele acesteia)
- ON MODIFY CASCADE (la modificarea identificatorului rutei se modifică şi câmpul cheie străină al etapei)

Relaţia nr. 5 O etapă (`RouteItem`) trebuie să aibă asociate cel puţin două elemente `RouteItemStop`. De asemenea un element `RouteItemStop` trebuie să facă parte dintr-o etapă a unei rute. Integritatea relaţională:

- ON DELETE CASCADE (la ştergerea unei etape se şterg şi opririle acesteia)
- ON MODIFY CASCADE (la modificarea identificatorului unei etape se modifică şi câmpul cheie străină al opririlor asociate acelei etape)

În câteva cuvinte, aceste relaţii rezumă următorul comportament: ştergerea unui utilizator duce la ştergerea rutelor acestuia. Ştergerea unei rute duce la ştergerea componentelor acesteia: informaţii de căutare, setări pentru agenţi, etapele rutei. Ştergerea unei etape duce la ştergerea opririlor ce o compun. Alte ştergeri de elemente nu produc nici un efect în cascadă (fără acţiune).

B.2 Validarea modelului prin normalizare

Informaţiile referitoare la entităţi au fost prezentate într-o secţiune anterioară, incluzând conceptele de cheie primară şi cheie străină. Acest capitol demonstrează, pe baza acestor informaţii, că baza de date a aplicaţiei PITA este în formă normală trei.

Forma normală unu (1NF) presupune eliminarea repetiţiilor din atribute. Este evident că nu există grupuri repetitive în nici una dintre entităţi deci relaţia este în formă normală unu.

Forma normală doi (2NF) presupune eliminarea dependenţelor parţiale de cheia primară. În cazul bazei de date PITA nu avem dependenţă parţială de cheie, deci relaţia este în formă normală doi.

Forma normală trei (3NF) presupune eliminarea dependenţelor tranzitive de cheia primară. Nu există dependenţe tranzitive de cheia primară deci relaţia este în formă normală trei.

B.3 Validarea modelului prin tranzacţii

Această secţiune va prezenta detaliat doar tranzacţiile mai complicate ale bazei de date PITA. O listă completă a tranzacţiilor este următoarea: crearea unui utilizator, ştergerea unui utilizator, modificarea datelor unui utilizator, *crearea unei rute*, *ştergerea unei rute*, modificarea setărilor pentru o rută, *listarea informaţiilor de bază despre o rută*, *listarea tuturor informaţiilor despre o rută*.

B.3.1 Crearea unei rute

Crearea unei rute presupune executarea, pe rând a următorilor paşi:

Pasul 1) Inserarea unui element în tabela `Route`.

Pasul 2) Inserarea unui element în tabela `Route_Search` care să refere ruta introdusă la primul pas.

Pasul 3) Inserarea unui element în tabela `Journey_Settings` care să refere ruta introdusă la primul pas.

Pasul 4) Inserarea tuturor elementelor `RouteItem` care compun ruta în tabela `Route_Item`, evident setând câmpul `Route_ID` la valoarea identificatorului rutei. După persistarea unui element `RouteItem` trebuie persistate şi elementele `RouteItemStop` conţinute de acesta (în tabela `Route_Item_Stop`).

B.3.2 Ştergerea unei rute

Ştergerea unei rute duce automat la ştergerea informaţiei aferente acesteia: informaţia de căutare, informaţia de setare, lista de etape a rutei şi listele de opriri pentru fiecare etapă. Acest lucru este realizat de sistemul de baze de date prin intermediul integrităţii relaţiilor (specificarea elementului ON DELETE CASCADE pentru cheile străine în cauză).

Această acţiune afectează conţinutul tabelelor: `Route`, `Route_Search`, `Journey_Settings`, `Route_Item` şi `Route_Item_Stop`.

B.3.3 Listarea informaţiilor despre o rută

Informaţiile de bază despre o rută nu includ etapele şi opririle. Ele includ doar informaţia aflată în următoarele tabele: `Route`, `Route_Search`, `Journey_Settings`. Pentru aceasta se execută o interogare care include aceste tabele, folosind relaţiile existente.

B.3.4 Listarea informaţiilor complete despre o rută

Informaţiile complete despre o rută includ toate informaţiile de bază cât şi traseul detaliat al acesteia. Acest lucru presupune realizarea unei interogări complexe asupra tabelelor: `Route`, `Route_Search`, `Journey_Settings`, `Route_Item` şi `Route_Item_Stop`. Legarea valorilor în mod adecvat se face prin folosirea relaţiilor existente.

B.4 Script PostgreSQL

Această secţiune prezintă scriptul PostgreSQL prin intermediul căruia se crează baza de date pentru aplicaţia PITA. Acesta arată conform listingului următor.

Listing B.1: Fişier SQL de creare al bazei de date PITA

```
1  CREATE DATABASE pita ;
2
3  CREATE TABLE PITA_USER (
4        ID SERIAL PRIMARY KEY ,
5       USERNAME VARCHAR(30) NOT NULL UNIQUE,
6      PASSWORD VARCHAR(30) NOT NULL,
7      FIRST_NAME VARCHAR(40) NOT NULL,
8      LAST_NAME VARCHAR(40) NOT NULL,
9      BIRTHDATE DATE,
10     EMAIL VARCHAR(50),
11     PHONE VARCHAR(30)
12   ) ;
13
14 CREATE TABLE ROUTE (
15       ID SERIAL PRIMARY KEY ,
16     NAME VARCHAR(50) NOT NULL,
```

```
17      OWNER_ID INT   ,
18      STATE SMALLINT DEFAULT 0 ,
19      START DATETIME  ,
20      FOREIGN KEY ( OWNER_ID ) REFERENCES PITA_USER ON DELETE CASCADE
21  ) ;
22
23  CREATE TABLE ROUTE_ITEM (
24          ID SERIAL PRIMARY KEY,
25          ROUTE_ID INT NOT NULL,
26          ITEM_INDEX SMALLINT NOT NULL,
27          TYPE SMALLINT NOT NULL,
28          IDENTIFICATOR VARCHAR(20) NOT NULL,
29          FOREIGN KEY ( ROUTE_ID ) REFERENCES ROUTE ON DELETE CASCADE
30  ) ;
31
32  CREATE TABLE ROUTE_ITEM_STOP(
33          ITEM_ID INT NOT NULL,
34          STOP_INDEX SMALLINT NOT NULL,
35          PLATFORM VARCHAR(10) NOT NULL,
36          NAME VARCHAR(40) NOT NULL,
37          LEAVES_AT DATETIME,
38          FOREIGN KEY ( ITEM_ID ) REFERENCES ROUTE_ITEM ON DELETE CASCADE
39  ) ;
40
41  CREATE TABLE JOURNEY_SETTINGS (
42          ROUTE_ID INT NOT NULL UNIQUE,
43          NOTIFY_STYLE VARCHAR(40) NOT NULL,
44          NOTIFY_TYPE VARCHAR(10) NOT NULL,
45          NOTIFY_IDENT VARCHAR(40),
46          BEFORE_CHANGE BIGINT,
47          BEFORE_STOP BIGINT DEFAULT 60000,
48          FOREIGN KEY ( ROUTE_ID ) REFERENCES ROUTE ON DELETE CASCADE
49  )   ;
50
51  CREATE TABLE ROUTE_SEARCH (
52          ROUTE_ID INT NOT NULL UNIQUE,
53          START_STATION VARCHAR(40) NOT NULL,
54          END_STATION VARCHAR(40) NOT NULL,
55          VIA VARCHAR(40),
56          START_DATE DATETIME,
57          END_DATE DATETIME,
58          PLANNER_ENGINE VARCHAR(30) NOT NULL,
59          FOREIGN KEY ( ROUTE_ID ) REFERENCES ROUTE ON DELETE CASCADE
60  ) ;
```

Bibliografie

[1] Steve Graham, Glen Daniels et all, *Servicii Web cu Java*, Teora (2003)

[2] David Chappell, Tyler Jewell, *Java Web Services*, O'Reilly (March 2002)

[3] *Specificaţiile XML 1.0*, `http://www.w3.org/TR/2000/WS-xml-2e-20000814`

[4] *XML Namespaces*, `http://www.w3.org/TR/1999/REC-xml-names-19990114`

[5] *Specificaţiile SOAP 1.1*, `http://`

[6] *Specificaţiile SOAP with Attachments*, `http://`

[7] *Specificaţiile WSDL*

[8] *Specificaţiile UDDI*

[9] *Documentatia Apache AXIS*

[10] *Documentatia Jakarta Tomcat*