Universitatea Babeş-Bolyai
Facultatea de Matematică şi Informatică
Specializarea Sisteme Distribuite În Internet

LUCRARE DE DISERTAŢIE
Descriptor XRDL şi echivalenţa cu serviciile XML-RPC

Conducător ştiinţiific:
Prof. Dr. BOIAN Florian

Absolvent:
TROANCĂ Diana

2013

Cuprins

1. Introducere

1.1 Generalităţi

Internetul evoluează în ultimii ani în direcţia aplicaţiilor distribuite, iar tehnologiile care se dezvoltă cel mai mult în acest sens sunt servciile web. Acestea reprezintă o soluţie foarte populară pentru a pune la dispoziţia clientului prin internet funcţionalităţi cât mai diverse, eficiente, dar şi pesonalizate. Avantajul principal al serviciilor web este interoperabilitatea, fapt pentru care serviciile web pot fi implementate în orice limbaj de programare, iar utilizatorul nu trebuie să ştie în ce limbaj a fost implementat sau ce distribuţie foloseşte serviciul pe care doreşte să îl apeleze. Pentru apelul unei funcţionalităţi oferite de un serviciu clientul are nevoie de puţine informaţi despre server, cum ar fi URL-ul la care serviciul este pus la dispoziţie, numele metodei apelate şi parametrii săi.

Datorită uşurinţei de a implementa o aplicaţie distribuită cu ajutorul unui serviciu web, toate companiile mari încep să ofere, în prezent, funcţionalităţile noi prin intermediul serviciilor web. De asemenea pentru funcţionalităţile deja existente, se implementează servicii web echivalente. Un exemplu în acest sens este Google care oferă funcţionalităţi precum motor de căutare sau hărţi prin intermediul serviciilor web. Astfel, înglobarea funcţionalităţilor oferite în cadrul altor aplicaţii devine foarte simplă.

Datorită dezvoltării rapide a serviciilor web, în funcţie de diferitele tipuri de servicii (SOAP, REST, XML-RPC) acestea se extind pe direcţii diferite. Un avantaj major în utilizarea serviciilor ar fi uniformizarea lor şi automatizarea unor acţiuni. Pentru această uniformizare rolul principal îl deţine utilizarea fişierelor de descriere a serviciilor web, ceea ce reprezintă tema centrală a lucrării de faţă.

1.2 Motivaţie şi obiective

Serviciile web pot fi caracterizate prin fişiere de descriere în format XML, care este un format universal recunoscut. Dezavantajul este că, dintre cele trei tipuri de servicii web (SOAP, REST şi XML-RPC), XML-RPC nu defineşte un format standard pentru

fişierul de descriere al serviciului. Serviciile de tip XML-RPC reprezintă categoria cea mai simplă de servicii web. De asemenea datorită simplităţii lor constituie o soluţie ieftină şi fiabilă pentru implementarea unor funcţionalităţi sub formă de servicii web. Cu toate acestea, este un tip de serviciu nu foarte extins. Consider că existenţa unui format de descriere a serviciului cu avantajele aferente le va creşte popularitatea serviciilor XML-RPC. De aceea, obiectivul principal al acestei lucrări este de a prezenta XRDL ca Dlimbaj de descriere pentru XML-RPC.

Utilizarea fişierelor de descriere a serviciilor web este multiplă. În primul rând, pentru a simplifica implementarea unui serviciu, o soluţie este să se genereze automat un schelet al serviciului web utilizând fişierul de descriere al serviciului. Această generare poate fi folosită şi în cazul în care este nevoie de reconstrucţia serviciului web. De asemenea, pe baza fişierului de descriere se pot genera automat schelete pentru clienţii care doresc să apeleze serviciul web.

Un alt obiectiv care nu a fost încă atins în domeniul cercetărilor serviciilor web este compararea a două servicii web de tip diferit. Deşi au existat diferite propuneri de aplicaţii şi metode de comparare a două servicii web [3, 8], nici una dintre acestea nu este viabilă în lipsa fişierelor de descriere a serviciilor web. De aceea existenţa unui standard pentru fişierul de descriere a oricărui tip de servciiu web este foarte importantă. Obiectivul lucrării de faţă este să atragă atenţia asupra formatului XRDL de descriere pentru serviciile XML-RPC. În viitor, XRDL va putea fi acceptat ca şi standard pentru descrierea unui serviciu web de tip XML-RPC, iar utilizarea lui va deveni, dacă nu obligatorie, cel puţin recomandată la implementarea unui serviciu.

1.3 Contribuţie personală

XRDL a pornit ca un proiect open-source şi nu este încă acceptat ca şi standard pentru XML-RPC Description Language. Deşi proiectul include câteva aplicaţii, precum generarea automată a documentului XRDL pentru servicii şi clienţi scrişi în PHP sau C++/Qt [20], nu a fost demonstrată echivalenţa formatului XRDL cu serviciile XML-RPC. Contribuţia personală principală în această lucrare este demonstraţia faptului că XRDL este un limbaj de descriere valid pentru XML-RPC. Acest rezultat va fi publicat în articolul intitulat "XRDL: a valid Description Language for XML-RPC", în cadrul conferinţei Knowledge Engineering, Principles and Techniques Conference (KEPT) 2013. Scopul articolului şi al lucrării de faţă este să aducă o acceptare mai largă a formatului XRDL în lumea ştiinţifică.

Ca parte aplicativă inovativă lucrarea prezintă un generator automat pentru un servi-

ciu web de tip XML-RPC scris în Java. Acestă aplicaţie are un rol foarte important. Dacă se doreşte ca toate serviciile web să aibă fişiere de descriere, trebuie să se ia în considerare că există deja foarte multe servicii web care nu au fişiere de descriere. De asemenea nu se doreşte complicarea procesului de implementare al unui serviciu web. Astfel, utilizând această aplicaţie, fişierul de descriere XRDL al unui serviciu web poate fi generat automat. Avantajul soluţii implementate este că nu depinde de distribuţia XML-RPC folosită în implementarea serviciului web (de exemplu Apache XML-RPC sauc Redstone XML-RPC pentru Java, XmlRpc++ pentru C++, xmlrpclib pentru Python, etc.). În cazul serviciilor web deja existente, modificările care trebuie făcute sunt minore, find nevoie doar de apelul unei metode care genereaza documentul XRDL. Aplicaţia prezentată completează setul de generatoare automate existente în proiectul opensource XRDL [20].

1.4 Structura lucrării

Lucrarea este structurată pe 5 capitole: un capitol introductiv, trei capitole teoretice şi un capitol aplicativ. Capitolul introductiv, cel de faţă, prezintă pe scurt motivaţia şi obiectivele lucrării, contribuţia originală şi structurarea lucrării.

Capitolul 2 intitulat "Servicii Web" conţine trei subcapitole şi începe cu prezentarea pe scurt a istoriei apariţiei serviciilor web. În următorul subcapitol sunt prezentate caracteristicile unui serviciu, cu accent pe avantajele oferite de serviciile web. Ultimul subcapitol descrie componentele şi arhitectura unui serviciu web.

Capitolul 3 se intitulează "Servicii Web de tip XML-RPC", iar primul subcapitol prezintă caracteriticile generale ale acestui tip de serviciu în contrast cu celălalte două tipuri, SOAP şi REST. Al doilea subcapitol descrie protocolul XML-RPC care documentează modelul de date folosit, formatul cererii şi al răspunsului.

Capitolul 4 intitulat "Descriptorul XRDL" cuprinde încă trei subcapitole. Primul dintre acestea prezintă pe scurt scopul descriptorilor de servicii web. Următroul subcapitol descrie strucutra documentului XRDL. Ultimul subcapitol tratează validitatea formatului XRDL ca limbaj de descriere pentru XML-RPC.

Ultimul capitol prezintă partea aplicativă a lucrării începănd cu motivarea şi scopul aplicaţiei. În al doilea subcapitol se specifică aplicaţia cu funcţiile şi metodele oferite, urmând ca în ultimul subcapitol săse prezinte viitoare extinderi în urma rezultatelor prezentate în această lucrare.

2. Servicii Web

2.1 Scurtă istorie

La începutul anilor '90 pentru servicii software distribuite se folosea tehnologia DCE (Distributed Computing Environment). DCE include un framework şi un toolkit care ajută la dezvoltarea aplicaţiilor distribuite client-server. Framework-ul are diferite componente, şi anume: DCE/RPC care este un mecanism pentru apeluri Remote Procedure Call, un serviciu de nume (naming service), un serviciu de timp (time service), un serviciu de autentificare şi un sistem de fişiere distribuit numit DCE/DFS (DFS- Distributed File System) [12]. DCE era bazat pe Unix, iar ca variantă pentru Windows, Microsoft a implementat o altă versiune cunoscută ca MSRPC. Primele framework-uri pentru sisteme distribuite (CORBA- Common Object Request Broker Arhitecture, Microsoft Distributed COM, Java RMI - Remote Method Invocation) se bazează tot pe framework-ul DCE/RPC. Deşi aceste framework-uri nu oferă un standard pentru aplicaţii distribuite, ele au reprezentat un pas important pentru dezvoltarea aplicaţiilor client-server, fiind totodată precursoarele serviciilor web.

La sfârşitul anilor '90 a fost dezvoltat standardul XML (Extensible Markup Language) pentru a simplifica specificarea apelului peste web. Bazându-se pe acest standard, în 1998 Dave Winer a dezvoltat XML-RPC, primul tip de serviciu web, care era un framework mult mai simplu ce oferea suport pentru tipuri elementare de date. Avantajul major adus de XML-RPC este independenţa de limbajul de programare si utilizarea protocolului HTTP (Hypertext Transfer Protocol, protocol ce a fost dezvoltat dupa apariţia tehnologiilor bazate pe DCE/RPC) sau SMTP (Simple Mail Transfer Protocol) pentru transportul datelor. XML-RPC este un protocol simplu care se bazeaza pe modelul cerere-raspuns [7, 1].

În contradicţie cu simplitatea framework-ului XML-RPC a fost dezvoltat protocolul SOAP (Simple Object Access Protocol) care a vrut sa ofere suport pentru mai multe tipuri de date. În cadrul SOAP au fost numeroase propuneri de standarde pentru diferite categorii, cum ar fi: interoperabilitate, securitate, metadata, resurse şi altele. Astfel SOAP

devine un protocol foarte complex, care are însă avantajele sale, cum ar fi comunicarea SOAP care trece peste proxie-uri sau firewall-uri fară a fi nevoie să se modifice protocolul în sine [14]. Din cauza complexităţii sale SOAP nu foloseşte decât metoda POST a protocolului HTTP, deorece celălalte metode nu ofera informaţii suficiente. Ca avantaj faţă de framework-urile bazate pe DEC/RPC, SOAP foloseşte formatul XML pentru reprzentarea datelor. Astfel transferul de date poate fi verificat, validat şi procesat. Din punct de vedere teoretic, se consideră că XML-RPC face parte din protocolul SOAP. Ce aduce în plus protocolul SOAP este WSDL (Web Service Description Language) care reprezintă un contract al serviciului şi UDDI (Universal Description, Discovery and Integration), folosit pentru publicarea şi descoperirea serviciului web. O altă utilizare a standardului WSDL este generarea automată a unui client [7, 1].

Un alt standard dezvoltat în anul 2000 a fost REST. Acest tip de serviciu web a pornit de la teza de doctorat a lui Fielding, care propune să se folosească toate metodele oferite de HTTP (GET, PUT, POST şi DELETE). Aceste caracteristici deosebesc REST de SOAP, care foloseşte doar metoda POST a protocolului HTTP. Totodată REST devine mai uşor de folosit pentru utilizator, dar mai greu de procesat de către calculator. Dezavantajul serviciilor REST a fost, la început, lipsa unui standard pentru generarea clienţilor sau a unui contract al serviciului. De aceea a fost propus WADL (Web Application description Language) de către World Wide Web Consortium. WADL reprezintă o descriere bazată pe XML a serviciului REST. Totuşi WADL nu a fost standardizat, deoarece nu este suportat la scară largă [7, 13, 1].

În evoluţia actuală a serviciilor web se consideră că acestea se împart în servicii de tipul SOAP, care includ şi serviciile XML-RPC, şi servicii de tipul REST. Dintre cele trei categorii prezentate, cea mai populară pe piaţă este categoria serviciilor de tip REST. Acestea au ca avantaj faţă de serviciile de tip XML-RPC un suport pentru mai multe tipuri de date, iar avantajul principal faţă de serviciile de tip SOAP îl reprezintă simplitatea. Cu toate acestea, în cele mai multe cazuri transferul de date şi documente suportat de XML-RPC este suficient pentru a implementa funcţionalităţile dorite. De aceea lucrarea de faţăare ca subiect serviciile web de tip XML-RPC.

2.2 Caracteristicile unui serviciu web

Serviciile web reprezintă o categorie de tehnologii middleware de tip RPC (Remote Procedure Call) care permit apeluri peste web şi au ca scop principal interoperabilitatea oricăror două aplicaţii şi independenţa de platformă şi de limbajul de programare. Astfel serviciile web aduc eterogenitate în cadrul aplicaţiilor distribuite. W3C (World Wide Web

Consortium) a publicat în anul 2002 o serie de propuneri pentru a standardiza serviciile web. În această perioadă serviciile web au început să se dezvolte în diferite direcţii şi prin aceste propuneri W3C încearcă să definească arhitectura şi componentele serviciilor web, pentru a uniformiza modul de apel între sistemul care oferă serviciile şi cel care face cereri către aceste servicii. O altă parte a propunerilor reprezintă o standardizare a modului în care serviciile web sunt descoperite de sistemele care le apelează. În cadrul acestor propuneri W3C defineşte serviciul web ca un software proiectat astfel încât să ofere suport pentru interoperabilitate peste reţea [6, 5, 16].

Serviciile web au o serie de caracteristici care le conferă abilitatea de interoperabilitate, cea mai importantă dintre acestea fiind faptul că se bazează pe XML (Extensible Markup Language). XML este un standard propus de W3C pentru structurarea, stocarea şi transportul datelor care uniformizează practic transferul de date între aplicaţii. Se rezolvă astfel problema diferitelor reprezentări de date. Avantajul principal adus de XML este reprezentarea serializată a obiectelor din diferite limbaje de programare. De menţionat este că XML ajută atât la transferul de date, cât şi al documentelor indiferen de complexitatea acestora. Astfel XML asigură portabilitatea şi transparenţa în transferul de documente între client si serverul web [5, 18].

Un alt avantaj al serviciilor web este "cuplarea slabă" între componenta care oferă serviciile şi consumatorii serviciilor. Această caracteristică se referă la legătura dintre server si client, şi anume, serverul trebuie să aibe posibilitatea să modifice interfaţa serviciului fară să influenţeze abilitatea clientului de a apela serviciul (fară să fie nevoie să se facă modificări suplimentare pe partea clientului). În cazul serviciilor web integrarea serverului cu clientul este foarte simplă şi se poate întreţine uşor pe parcursul diferitelor modificări ce apar [5].

Un serviciu web poate fi atât sincron cât şi asincron. Un serviciu asincron îi oferă clientului avantajul că după apelul serviciului poate să efectueze alte operaţii până primeşte răspunsul de la serviciul web. Capacitatea unui serviciu de a fi asincron are un rol fundamental în cuplarea slabă a servului cu clientul [5].

Serviciile web oferă clienţilor posibilitatea să apeleze funcţii sau servicii ale unor obiecte remote utilizand un protocol bazat pe XML. Astfel serviciul web poate fie să ofere servicii proprii cu aceleaşi funcţionalităţi, fie să transforme invocaţiile primite în invocaţii către componente EJB sau .NET [5].

Scopul serviciilor web este ca, în final, să se ajungă la o integrare a business-urilor automată. Când se descoperă o componentă software nouă care oferă funcţionalitatea dorită, aceasta se accesează, se integrează şi apoi se pot invoca metodele oferite de noul software. Toate aceste procese trebuie să aibă loc autoamat fără intervenţie umană. Din acest punct de vedere serviciul web poate fi văzut ca o interfaţă care are implementate o

serie de funcţionalităţi şi care este accesibilă peste web. Astfel, serviciul web face legătura între furnizorul de servicii si clientul care doreşte să le apeleze.

2.3 Componentele şi arhitectura unui serviciu web

Un serviciu web se bazează pe tehnologii standard Internet pentru a furniza serviciile dorite. Astfel clientul cand are nevoie de o funcţionalitate apelează un serviciu web care oferă funcţionalitatea respectivă , iar acesta la rândul său acţionează pe post de interfaţă şi apelează aplicaţia care implementează serviciul respectiv. Deoarece comunicarea se efectuează pe Internet, localizarea celor trei componente nu este importantă, acestea putând fi localizate pe diferite maşini. Pentru fiecare nivel din stiva de protocoale serviciile web s-au dezvoltat în jurul unor protocoale, fie că acestea sunt protocoale standard folosite pentru comunicaţia peste web sau protocoale specializate pentru serviciile web [1]. Această stivă împreună cu protocoalele aferente se poate observa în figura 2.1.

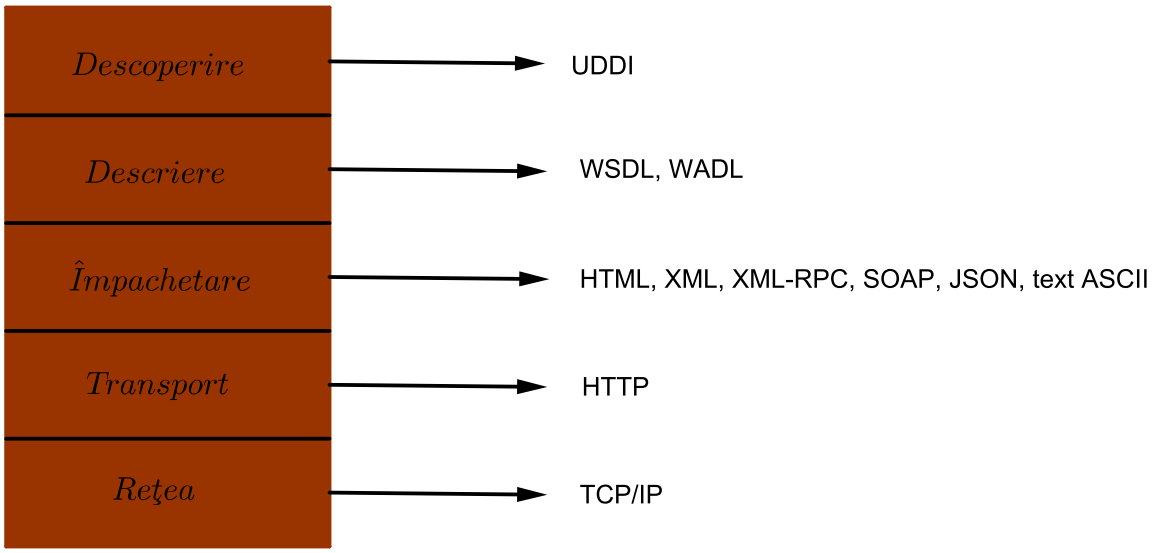

Figura 2.1: Stiva de protocoale pentru un serviciu web

Primul nivel al stivei se referă la descoperirea serviciului web de către clienţii care doresc să îl utilizeze. Serviciile web de tip SOAP folosesc în acest sens serviciile UDDI. Companii precum Microsoft, Google, IBM oferă servicii UDDI publice care reprezintă directoare globale ce conţin mai multe servicii oferite de diferiţi clienţi ai companiilor respective. Există de asemenea servicii UDDI private folosite de corporaţii bancare, organizaţii guvernamentale, consorţii, etc. Totuşi alte tipuri de servicii nu oferă acest nivel de desco-

perire, ceea ce implică faptul că un client care vrea să apeleze serviciile respective trebuie să cunoască detaliile serviciului prin alte mijloace, cum ar fi un contract direct sau o ofertă citeboian.

Pentru a introduce un serviciu web intr-un director de servicii folosit de UDDI, trebuie ca acesta să aibă o descriere. Protocoalele folosit pestru a descrie un serviciu web sunt WSDL şi WADL. WSDL e folosit, de obicei, de serviciile de tip SOAP şi conţine o descriere a serviciului care îi oferă clientului toate informaţiile necesare pentru a apela serviciul. În această descriere serviciul este definit ca o colecţie de porturi. Deoarece se bazează pe XML, WSDL poate fi procesat automat de către anumite programe software [1, 17]. Serviciile de tip REST folosesc ca şi descriptor WADL care se bazează de asemenea pe XML. WADL descrie serviciul sub forma unei mulţimi de resurse care au parametrii si metode care descriu forma cererii şi a răspunsului unei resurse. Deşi există două standarde pentru descrierea unui serviciu web, sunt şi servicii web care nu au fisier de descriere [1, 15]. De obicei serviciile web de tip XML-RPC nu oferă o astfel de descriere, deoarece nu este obligatorie pentru ca un client să poată apela serviciul cu succes. În lucrarea de faţă vom prezenta un descriptor pentru serviciile de tip XML-RPC. Acesta, deşi nu este încă larg răspândit şi nici standardizat, este foarte util.

Pentru a menţine caracteristica de interoperabilitate a serviciilor web cel mai important nivel din stiva prezentată este cel de impachetare a datelor. Astfel cererile şi răspunsurile unui serviciu web trebuie să aibă un format standard, independent de platformă. De aceea pentru reprezentarea cererilor şi a răspunsurilor se foloseşte de obicei XML, dar se poate folosi şi JSON, HTML(sau XHML) sau text ASCII. Apoi pentru împachetarea acestora se folosesc cele trei standarde de servicii web: XML-RPC, SOAP şi REST [1].

Nivelul de transport se ocupă cu modalitatea în care cererea gata împachetată este trimisă la serviciul web, iar apoi raspunsul gata impachetat este trimis înapoi la client. Cel mai des folosit protocol pentru transportul peste web este HTTP (poate fi folosit şi alt protocol de transfer al resurselor) [1].

La nivelul reţea, comunicarea între două calculatoare se face de obicei prin Internet cu ajutorul protocolului bazat pe TCP/IP (Transmission Control Protocol/ Internet Protocol) [1].

Cu ajutorul primelor două nivele ale stivei de protocoale, descoperirea şi descrierea, clientul află toate informaţiile necesare pentru a putea apela metode ale serviciului. Aceste informaţii nu se schimbă de la un apel la altul. De aceea ar fi redundant, atât pentru viteza de reacţie a serviciului, cât şi pentru cantitatea de informaţii trimisă prin reţea, ca acestea să fie trimise de serviciu către client la fiecare apel de metodă. În acest scop se foloseşte un proxy la client care salvează toate datele necesare, astfel încât, la al doilea

apel către serviciu, clientul sare peste etapele de descoperire si descriere a serviciului. Etapele de împachetare, transport şi reţea rămân însă valabile pentru fiecare apel făcut de client. Acestea au rolul de a urmări cererea pe traseul său de la client la server unde se transformă într-un răspuns, iar apoi răspunsul până în momentul în care ajunge la client [1].

3. Servicii Web de tip XML-RPC

3.1 Generalităţi

XML-RPC oferă programelor posibilitatea de a apela funcţii sau proceduri stocate pe un alt calculator prin reţea. XML-RPC este cel mai simplu dintre cele trei tipuri de servicii web, în primul râpentru că refoloseşte infrastructura existentă pentru comunicarea peste web : HTTP şi XML. Datorită utilizării XML datele pot fi procesate de calculator şi în acelaşi timp pot fi citite şi înţelese de oameni. Astfel dezvoltatorii nu mai trebuie să se concentreze pe proiectarea unor interfeţe pentru utilizatori, cum era cazul până acum la toate aplicaţiile web [4, 9].

XML-RPC are anumite limitări datorită tipurilor restrânse de date acceptate sau a utilizării protocolului HTTP. Dezavantajul folosirii protocolului HTTP este limitarea vitezei de răspuns, a mărimii mesajelor transmise şi a utilizării serviciului la scară largă (în cazul unor aplicaţii care trimit informaţii foarte des şi îpentru care timpul de raspuns este vital). De asemenea, comunicarea prin HTTP este stateless, iar starea sistemului trebuie memorată în logica aplicaţiei. Cu toate acestea, avantajul major adus de acest tip de servicii web este simplitatea. Datorită simplităţii sale integrarea sistemelor de tipuri diferite devine mult mai simplă decâîn cazul celorlalte servicii web. De asemenea nu apar dificultăţi în implementarea protocolului şi testarea interoperabilităţii se face foarte uşor. Deşi XML-RPC oferă suport pentru un numar mic de tipuri de date, are însăun nivel de granularitate suficient pentru a exprima informaţiile necesare. Astfel XML-RPC este folosit atâde dezvoltatori care integrează sisteme distribuite, cât şi de dezvoltatori care creează servicii publice. În cel de=al doilea caz, avantajul dezvoltatorului este că folosind XML-RPC creează o interfaţă prin care serviciul este accesibil utilizatorilor, iar funcţionalităţile le pot implementa în orice limbaj de programare. Dezvoltatorii au control total asupra serverului, dar nu pot controla modul în care clienţii apelează serviciile oferite [4, 9, 1].

Deoarece respectă paradigma RPC, protocolul XML-RPC se bazează pe un mecanism simplu de cerere - răspuns. Pentru a asigura comunicarea, implementarea protocolului nu

trebuie să aibă detalii despre arhitectura şi infrastructura clientului care apelează serviciul, ci trebuie doar să publice o interfaţă care poate fi apelată peste web. Deoarece XML-RPC se bazează pe schimburi de informaţii prin apeluri procedurale, nu este obligatoriu să respecte arhitectura client-server. XML-RPC suportă atât comunicarea peer-to-peer, cât şi comunicarea client-server. De asemenea, un program poate include în acelşi timp o parte de server XML-RPC şi o parte de client XML-RPC [4, 9].

Dezvoltatorii serviciilor web XML-RPC nu au nevoie să cunoască toate detaliile interne despre funcţionarea protocolului. De cele mai multe ori se folosesc librării care se ocupă cu detaliile de implementare ale protocolului, cum ar fi împachetarea şi transportul datelor. Singurul efort pe care dezvoltatorii este nevoie să îl facă este să integreze aplicaţia cu librăriile folosite, care de obicei oferă un API uşor de utilizat şi de înţeles. De asemenea XML-RPC le oferă posibilitatea dezvoltatorilor de a folosi un standard în crearea protocolului pentru integrarea diferitelor sisteme. Acest standard oferă posbilitatea refolosirii funcţionalităţilor implementate. Acest lucru nu era posibil până la aparitXML-RPC, deoarece fiecare dezvoltator concepea alt protocol pentru ca două sisteme să comunice între ele, iar două astfel de protocoale diferite nu puteau comunica între ele. Astfel standardizarea salvează timp şi efort prin refolosirea implementărilor unor funcţionalităţi XML-RPC.

O caracteristică a serviciilor web de tip XML-RPC care paote fi considerată în acelaşi timp un avantaj şi un dezavantaj este trecerea de firewall-uri cu ajutorul HTTP. Avantajul este, evident, faptul că serviciul poate fi utilizat fă ră ca utilizatorii să fie nevoiţi să modifice politica de filtrare a pachetelor pentru a putea apela funcţionalităţiile oferite de un serviciu web. Dezavantajul îl reprezintă problemele de securitate datorate uşurinţei cu care apelurile de tip cerere-răspuns către serviciul web trec de firewall. Astfel calculatoarele pe care se afla o interfaţă pentru un serviciu web XML-RPC sunt vulnerabile în faţa unor atacuri peste web [9].

Specificaţiile XML-RPC nu sunt foarte complexe, deoarece scopul protocolului este să aibă un format simplu, uşor extensibil. De asemenea standardul a fost gândit astfel încât să fie uşor de implementat şi să se poată adapta repede la alte medii de programare sau la alte sisteme de operare [11].

3.2 Protocolul XML-RPC

Un apel către un serviciu XML-RPC implică doi actori principali: un client şi un server, reprezentat printr-un URL (Uniform Resource Locator). Documentul XML folosit în cererea şi răspunsul XML-RPC are o sintaxă simplă şi bine definită de DTD (Document Type Definition), după cum se observă în figura 3.1 [1].

```
<!ELEMENT methodCall(methodName, params)|methodResponse(params|fault) >
<!ELEMENT methodName(#PCDATA) >
<!ELEMENT params(param*) >
<!ELEMENT param(value) >
<!ELEMENT value(i4|int|boolean|string|double|dateTime.iso8601|base64|struct|array) >
<!ELEMENT i4(#PCDATA) >
<!ELEMENT int(#PCDATA) >
<!ELEMENT boolean(#PCDATA) >
<!ELEMENT string(#PCDATA! >
<!ELEMENT double(#PCDATA) >
<!ELEMENT dateTime.iso8601(#PCDATA) >
<!ELEMENT base64(#PCDATA) >
<!ELEMENT array(data) >
<!ELEMENT data(value*) >
<!ELEMENT struct(member*) >
<!ELEMENT member(name, value) >
<!ELEMENT name(#PCDATA) >
<!ELEMENT fault(value) >
```

Figura 3.1: XML-RPC.dtd

Paşii urmaţi în cadrul apelului sunt următorii:

1. Clientul XML-RPC pregăteşte apelul către server, în care specifică metoda pe care vrea să o apeleze împreună cu parametrii necesari şi serverul care implementează metoda.

2. Clientul XML-RPC creează un document XML care conţine numele metodei şi parametrii pe care îl trimite către server printr-o cerere HTTP de tip POST.

3. Serverul XML-RPC primeşte cererea POST şi trimite fişierul XML mai departe la un listener XML-RPC.

4. Listener-ul XML-RPC parseaza fişierul XML primit pentru a obţine numele metodei şi parametrii cu care clientul vrea să apeleze metoda. Apoi apeleaza local metoda cu parametrii respectivi.

5. Componenta XML-RPC listener primeşte răspunsul de la metodă şi îl împacheteaza sub forma unui document XML.

6. Serverul web trimite răspunsul la cererea POST cu corpul format din documentul XML.

7. Clientul XML-RPC primeşte răspunsul de la server şi parsează documentul XML pentru a-i trimite clientului răspunsul la apelul făcut în forma cerută.

8. Clientul primeşte valoarea doritășşi poate continua procesarea datelor.

Din paşii prezentaţi observăm că folosirea protocolului HTTP înseamnă că XML-RPC este stateless şi sincron, deoarece răspunsul trebuie să fie primit pe aceeaşi conexiune HTTP ca şi cererea. Se pot implementa de asemenea sisteme XML-RPC asincrone, însă pentru aceasta este nevoie de mai multe cicluri cerere - răspuns [1, 9].

Specificaţiile tehnice XML-RPC conţin trei părţi principale, şi anume:

- modelul de date XML-RPC
- structura cererii XML-RPC
- structura răspunsului XML-RPC

Fiecare dintre acestea vor fi descrise pe scurt în următoarele trei subcapitole.

3.2.1 Modelul de date XML-RPC

Specificaţiile XML-RPC definesc şase tipuri simple de date şi două tipuri complexe. Deşi acestea reprezintă un număr foarte mic din toate tipurile de date existente în diferite limbaje, acestea sunt cele mai des utilizate şi de obicei sunt suficiente pentru a exprima informaţiile dorite. Cererea poate avea mai mulţi parametrii, toţi exprimaţi prin aceste tipuri de date, iar răspunsul poate returna o singură valoare din aceaşi listă de tipuri de date. Tipurile simple de date sunt reprezentate prin elemente simple XML care reprezintă tipul de date şi al căror conţinut este valoarea. Acest element este inclus într-un element de tipul `<value> </value>` [1, 9, 4].

Numerele întregi reprezintă prima categorie din cadrul tipurilor de date simple. Se pot reprezenta numere pe 32 biţi cuprinse între -2147483648 (-2^{31}) şi 2147483647 ($2^{31} - 1$). Pentru reprezentarea numerelor întregi există două posibilităţi echivalente, şi anume:

```
<value><i4>n</i4></value>
```

sau

```
<value><int>n</int></value>
```

Elementul $< i4 >$ este denumit astfel datorită reprezentării numărului întreg pe 32 biţi, adică 4 octeţi. Conţinutul tag-ului nu trebuie să conţina alte caractere în afara de prefixul $-$ sau $+$ şi cifre. Acestea sunt câteva exemple de numere întregi reprezentate conform specificaţiilor XML-RPC:

```
<value><i4>13</i4></value>
<value><int>1546</int></value>
<value><int>-348</int></value>
```

Numerele cu virgulă flotantă pot fi reprezentate binar pe 64 biţi conform standardului IEEE 754. Numerele cu virgulă flotantă care pot fi reprezentate au modulele cuprinse între 10^{-324} şi 10^{308}. Pentru a marca valori nedefinite standardul IEEE 754 propune utilizarea valoarii NaN(not a number), dar aceasta nu are o reprezentare echivalentă în XML-RPC şi nu poate fi folosită într-o cerere sau un răspuns. Aceste numere se reprezintă în standardul XML-RPC prin următorul element:

```
<value><double>n</double></value>
```

Exemple de astfel de numere ar fi:

```
<value><double>13.1234646</double></value>
<value><double>-23434.345667</double></value>
```

Caracterele valide în reprezentarea unui număr cu virgulă flotantă sunt similare cu cele de la numerele întregi, dar în plus apare punctul zecimal.

Elementele de tip boolean pot avea ca valori doar 1, care reprezintă adevărat, sau 0, care reprezintă fals. În standardul XML-RPC aceste valori se reprezintă dupa cum urmează:

```
<value><boolean>b</boolean></value>
```

Având în vedere limitarea valorilor posibile pentru un element boolean, singurele reprezentări posibile sunt următoarele:

```
<value><boolean>0</boolean></value>
<value><boolean>1</boolean></value>
```

Pentru elementele de tip string există două reprezentări echivalente. Acestea pot fi incluse doar în tag-ul `value` sau în tag-ul `string` care este inclus la rândul său într-un tag `value`. În elemente de tip string se poate folosi orice tip de caracter, dar caracterele speciale pentru XML, cum ar fi $<$ $>$ & se pot introduce astfel : &lt;, &gt;, respectiv &amp;. Ca exemplu, un string poate fi reprezentat prin cele două metode astfel:

```
<value>Elaine\ \&amp;\ Co.</value>$ (av\ai nd ca valoare \sh irul $Elaine\ \&\ Co.)
```

sau

```
<value><string>Elaine\ \&amp;\ Co.</string></value>
```

Ca şi reprezentare se poate folosi atât coldul ASCII, care este cel mai des întâlnit, cât şi UNICODE cu standardul ISO 8859-1.

Elementele de tip dată calendaristică se reprezintă conform unuia dintre profilele standardului ISO 8601, iar formatul folosit este an lună zi oră minut secundă. De aceea se foloseşte tag-ul `dateTime.iso8601`:

```
<value><dateTime.iso8601>AAAALLZZTHH:MM:SS/dateTime.iso8601></value>
```

Un exemplu de dată calendaristică reprezentată după standardul XML-RPC este:

```
<value><dateTime.iso8601>20130903T13:15:00</dateTime.iso8601></value>
```

Specificaţia XML-RPC menţionează că deţinătorul server-ului unde este implementat serviciul trebuie să specifice în documentaţie fusul orar. De aceea se recomandă să se folosească GMT, umrând ca aplicaţiile să convertească valoarea primită la ora locala în funcţie de fusul orar local.

Pentru a putea trimite prin XML-RPC caractere care sunt speciale în XML se foloseşte sistemul de codificare base 64. Acest sistem de codificare se bazează pe următoarele idei:

- Şirul de octeţi este prelungit, dacă este cazul, cu zerouri, astfel încât lungimea şirului să fie un multiplu de trei octeţi.
- Fiecare grupa de trei octeţi este împărţit în patru grupe de câte 6 biţi.
- Fiecare grup de 6 biţi poate fi reprezentat ca un număr întreg între 0 şi 63. Astfel se poate înlocui cu caracterul de pe poziţia corespunzătoare din lista celor 64 de caractere (26 litere mari, 26 litere mici, 10 cifre, caracterul +, caracterul /)
- În cazul în care grupul de 6 biţi este alcătuit doar din zerouri, acesta se reprezintă prin caracterul =.

Dacă luăm ca exemplu şi rul ASCII "Baza", acesta se reprezintă conform sistemului de codificare astfel:

```
<value><base64>QmF6YQ==</base64></value>
```

Constanta nulă se reprezintă prin tag-ul $< nil/ >$

Toate tipurile de date simple prezentate [11, 1, 9, 4] sunt sumarizate în figura 3.2.

int sau i4	întregi pe 32-biti	$< int > 5 < /int >$ $< i4 > 13 < /i4 >$
double	numere în virgulă flotantă pe 64-biti	$< double > -3.456 < /double >$
Boolean	adevărat (1) sau fals(0)	$< boolean > 0 < /boolean >$
string	text ASCII sau Unicode	$< string > Hello < /string >$
dateTime.iso8601	date in formatul: AAAALLZZTHH:MM:SS	$< dateTime.iso8601 > 20130903T13 : 11 : 05$ $< /dateTime.iso8601 >$
base 64	informație binară encodată în baza 64, după cum e definit in RFC 2045	$< base64 > date\ encodate\ in\ baza\ 64$ $< /base64 >$

Figura 3.2: Tipuri de date simple XML-RPC

Pentru a putea reprezenta informaţii mai complexe protocolul XML-RPC propune două tipuri de date compuse: tablou (array) şi structură. Conform specificaţiilor, tipurile de date compuse pot conţine una sau mai multe valori, care pot fi la rândul lor de tip simplu sau de un tip compus deja definit.

Un tablou este o secvenţă de valori XML-RPC. Aceste valori nu sunt obligatoriu toate de acelaşi tip. Tabloul poate fi văzut ca o listă de elemente nenumerotate, care nu pot fi accesate printr-un index ca în alte limbaje de programare. Elementul tablou este indicat de tag-ul `array` care conţine la rândul său un tag `data` [1, 11, 9]. Conform standardului XML-RPC, un element de tip tablou se reprezintă în forma următoare:

```
<value>
   <array>
      <data>
         <value>valoare</value>
         ...
         <value>valoare</value>
      </data>
   </array>
</value>
```

Un exemplu simplu de tablou este:

```
<value>
   <array>
      <data>
         <value><string>cantitate</string></value>
         <value><int>1</int></value>
         <value><string>pret</string></value>
         <value><double>12.01</double></value>
      </data>
   </array>
<value>
```

Se pot reprezenta tablouri multidimensionale prin includerea unui tablou în alt tablou. Un exemplu de tablou multidimensional este următorul:

```
<value>
   <array>
      <data>
         <value>
            <array>
               <data>
                  <value><string>linia 1, coloana 1</string></value>
                  <value><string>linia 1, coloana 2</string></value>
               </data>
            </array>
         </value>
         <value>
            <array>
               <data>
                  <value><string>linia 2, coloana 1</string></value>
                  <value><string>linia 2, coloana 2</string></value>
               </data>
            </array>
         </value>
      </data>
   </array>
</value>
```

Elementul de tip structură este format din perechi de forma nume-valoare. Numele sunt de tip `string`, iar valoarea poate fi de orice tip valid în XML-RPC. O structură are

următoarea formă:

```
<value>
   <struct>
      <member>nume_1</member>
      <value>valoare_1</member>
      ...
      <member>nume_n</member>
      <value>valoare_n</value>
   </struct>
</value>
```

Specificaţia XML-RPC nu impune ca numele elementelor sa fie unice în cadrul unei structuri. Cu toate acestea, deoarece de obicei implementările XML-RPC convertesc automat datele de tip structură din XML-RPC în date de tip dicţionar în limbajul respectiv, se recomandă ca numele să fie unice [1, 9]. Următorul element este un exemplu de structură în care numele membrilor nu se repetă:

```
<value>
   <struct>
      <member>Nume</member>
      <value>Pop</value>
      <member>Prenume</member>
      <value>Vasile</value>
      <member>Locul na\sh terii</member>
      <value>Sibiu</value>
      <member>Cet\aaa \tz enie</member>
      <value>Rom\ai n\aaa</value>
   </struct>
</value>
```

Un element de tip structură poate de asemenea să conţină alte elemente compuse, adică structuri sau tablouri.

3.2.2 Formatul cererii

Cererea XML-RPC reprezintă o cerere HTTP prin metoda POST care are ca şi corp un document XML. Antetul cererii conţine o serie de atribute, dintre care o parte sunt obligatorii, şi anume:

- `Method` - specifică faptul că metoda folosită în cerere este POST

- `User-Agent` - reprezintă ce implementare de XML-RPC se foloseşte, de exemplu ws-xmlrpc de la Apache
- `Host` - indică adresa maşinii server care va trata cererea
- `Content-Type` - trebuie să aibă valoarea `text/xml`, deoarece corpul cererii este un document XML
- `Contetn-Length` - indică lungimea corpului cererii

Corpul cererii este format dintr-un document XML care respectă sintaxa definită în schema 3.1. Astfel rădăcina documentului este elementul `methodCall` care conţine două elemente `methodName` şi `params`. Tag-ul `methodName` conţine numele metodei pe care utilizatorul vrea să o apeleze. Elementul `params` conţine lista de parametrii cu care se apelează funcţia. Fiecare parametru este specificat printr-un element de tipul `param` şi conţine un element de tipul `value` cu valoarea parametrului. În cazul apelului unei metode fără parametrii coprul cererii trebuie să conţină totuşi elementul `params`. Un exemplu de cerere validă XML-RPC (considerând că lungimea cererii îbytes corespunde), care conţine atât headere-ul HTTP, cât şi documentul XML care reprezintă corpul cererii, este următorul:

```
POST /xmlrpc HTTP 1.0
User-Agent: unClientXMLRPC/1.0
Host: 192.168.124.5
Content-Type: text/xml
Content-Length: 235

<?xml version="1.0"?>
<methodCall>
   <methodName>sum\aaa</methodName>
   <params>
      <param><value><int>13</int></value></param>
      <param><value><int>23</int></value></param>
      <param><value><int>10</int></value></param>
   </params>
</methodCall>
```

3.2.3 Formatul răspunsului

Răspunul XML-RPC este, ca şi în cazul cererii de tip HTTP şi conţine un header HTPP şi un corp al cererii în format XML. Atributele din header-ul HTTP care sunt obligatorii în acest caz sunt următoarele:

- `Server` - indică server-ul care a procesat răspunsul XML-RPC
- `Content-Type` - trebuie să aibă valoarea `text/xml`, deoarece corpul răspunsului este un document XML
- `Content-Length` - indică lungimea corpului răspunsului

Corpul răspunsului este format, ca şi în cazul răspunsului, dintr-un document XML care respectă sintaxa definită în schema 3.1. Elementul rădăcină al răspunsului este tag-ul `methodResponse`. Acesta conţine un alt fiu, care în funcţie de procesarea cererii cu succes sau nu poate fi `params` sau `fault`.

Dacă funcţia apelată u a dat nici o eroare, atunci este prezent elementul `params` care conţine un alt element `param` cu valoarea rezultatului funcţiei apelate. Dacă rezultatul trebuie sa returneze mai multe valori, acestea trebuie să fie înglobate într-un tip de dată compus, deoarece, spre deosebire de cerere, în cazul răsunsului elementul `params` poate avea un singur fiu `param`. Următorul este un exemplu de răspuns întors de server, în cazul unui apel cu succes al funcţiei dorite, la cererea dată ca exemplu anterior, :

```
<?xml version="1.0">
<methodResponse>
   <params>
      <param><value><int>46<int></value></param>
   </params>
</methodResponse>
```

În cazul în care apelul metodei returnează vreo eroare (metoda nu este găsită, apelul metodei nu s-a executat corect, metoda nu returnat un rezultat valid), protocolul HTTP oferă un mecanism de aruncare de excepţii. În acest caz este prezent tag-ul `fault` care conţine un tag `value`. Valoarea este reprezentată sub forma unei structuri cu doi membrii. Primul membru indică `faultCode` printr-un număr întreg, iar al doilea membru conţine uns tring ce reprezintă `faultString`. Un exemplu de răspuns în caz de eroare este următorul:

```
<?xml version="1.0">
<methodResponse>
```

```
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value>103</value>
            </member>
            <member>
               <name>faultString</name>
               <value><string>No such method.</string></value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

Un dezavantaj al standardului XML-RPC este că nu include o standardizare a codurilor şi a mesajelor `fault`. Astfel acestea diferă în funcţie de implementarea XML-RPC şi nu pot fi tratate automat de către aplicaţia clientului. Faptul că răspunsurile sunt trimise prin HTTP, nu ajută la detectarea eventualelor erori, deoarece codul de răspuns HTTP va fi `200 OK` chiar dacă corpul răspunsului conţine un element de tip `fault`.

După ce răspunsul a fost trimis se închide conexiunea cu clientul. Dacă după aceea acesta face încă o cerere se deschide o nouă conexiune XML-RPC.

4. Descriptorul XRDL

4.1 Scopul descriptorilor de servicii web

Dintre cele trei tipuri de servicii web, SOAP este singurul care are în mod obligatoriu un fişier de descriere al serviciului bazat pe standardul WSDL. REST propune ca format pentru descriptorul serviciului WADL, dar existenţa acestui fişier de descriere la publicarea serviciului este opţională. În contrast cu celălalte două tipuri de servicii, standardul XML-RPC nu conţine nici un format pentru descriptorul serviciului, iar acest fişier de descriere nu este nici obligatoriu. Faptul că descrierea serviciului nu ese obligatorie este considerat de mulţi cercetători ca un avantaj pentru că îi conferă standardului simplitate [19]. Cu toate acestea, lipsa fişierului de descriere al unui serviciu este un inconvenient pentru diferite tool-uri care utilizeaza descriptorul unui serviciu web.

Scopul principal este de a uniformiza serviciile web. Acest lucru presupune două idei principale : generarea automată uniformizată a diferitelor tipuri de servicii web şi a unor proxy-uri pentru clienţi şi compararea serviciilor web. Pentru generarea automată a unui serviciu web este nevoie de fişierul de descriere al acestuia, deoarece fără acel fişier generarea automată devine foarte complicată sau chiar imposibilă [2]. De asemenea compararea a două servicii web nu este posibilă fară existenţa fişierelor de descriere a celor două servicii web, indiferent de abordările propuse [8, 3].

Având în vedere utilizările fişierelor de descriere XML-RPC, consider că existenţa unui limbaj de descriere al serviciului XML-RPC este foarte importantă. În cele ce urmează voi prezenta XRDL ca limbaj de descriere pentru XML-RPC, care deşi nu este standardizat este un pas important pentru descrierea serviciilor de acest tip.

4.2 Structura documentului XRDL

XRDL (XML-RPC Description Language) este un proiect open-source creat ca înlocuitor pentru WSDL în cazul serviciilor de tip XML-RPC. XRDL este definit printr-o schema XSD ([20]) care descrie structura unui fişier XRDL valid:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="service">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="types">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="type" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="member" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType mixed="true">
                        <xs:attribute name="type" type="xs:string" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element><!-- types -->
        <xs:element name="methods">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="method" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType mixed="true">
                        <xs:attribute name="type" type="xs:string" use="required" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="result" type="xs:string" />
                  <xs:attribute name="name" type="xs:string" />
```

```
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    </xs:sequence>
    <xs:attribute name="url" type="xs:string" />
    <xs:attribute name="ns" type="xs:string" />
    <xs:attribute name="name" type="xs:string" />
  </xs:complexType>
 </xs:element><!-- service -->
</xs:schema>
```

După cum se observă în schema XSD, documentul XRDL are două mari secţiuni:

- o secţiune în care se definesc tipurile de date complexe folosite de serviciul respectiv
- o secţiune care descrie metodele serviciului web

A doua secţiune defineşte metodele oferite de serviciul web împreună cu parametrii şi rezultatul returnat. Tipul de date pentru parametrii şi rezultat se specifică cu ajutorul atributului `type` care are ca şi valoare un element de tip string. Teoretic pentru elementul `type` orice valoare care reprezintă un tip de date valid în XML, va genera un document XML valid. Cu toate acestea, pentru a descrie un serviciu XML-RPC este nevoie doar de datele simple din standardul XML-RPC, şi anume: int sau i4, double, boolean, string, dataTime.iso8601 sau base64, iar tipurile de date compuse din XML-RPC trebuie specificate în prima secţiune a documentului XRDL. De aceea restricţionez valorile luate de atributul `type` într-un document XRDL la cele 6 valori simple din standardul XML-RPC sau la o valoare complexă definită în prima parte a documentului.

4.3 Validitatea XRDL ca limbaj de descriere pentru XML-RPC

Proiectul open-source XRDL defineşte structura documentului XRDL şi propune câteva aplicaţii (generarea automată a documentului XRDL pentru servicii şi clienţi scrişi în PHP sau C++/Qt) [20]. Ceea ce lipseşte însă din documentaţia proiectului este o demonstraţie a faptului că XRDL este un limbaj de descriere valid pentru XML-RPC. Acest lucru presupune demonstrarea unei echivalenţe, şi anume că orice serviciu web de tip XML-RPC

poate fi descris printr-un document XRDL şi orice document valid XRDL descrie un serviciu web XML-RPC valid [10].

4.3.1 Existenţa unui document XRDL pentru orice serviciu XML-RPC

Prima parte a echivalenţei constă în demonstrarea următoarei teoreme:

Teorema 4.3.1 *Orice serviciu XML-RPC poate fi caracterizat printr-un document XRDL.*

Pentru a caracteriza un serviciu web trebuie descrise metodele implementate. Pentru ca utilizatorul să ştie cum să apeleze metodele are nevoie de numele funcţiei, datele de intrare necesare şi rezultatul pe care îl returnează. După cum se observă din schema XSD a documentului XRDL orice metodă poate fi descrisă printr-un mecanism simplu. Rămâne însă de demonstrat că orice tip de date are echivalent în XRDL. Pentru tipurile simple de date acest lucru este evident. Mai trebuie arătat că tipurile compuse din XML-RPC pot fi definite în XRDL. În paragrafele următoare se tratează cele douătipuri de date compuse: tablou şi structură.

Dupa cum am vazut în secţiunea 3.2.1, un element de tip tablou poate conţine atât elemente de tip simplu, cât şi elemente de tip compus. Astfel putem defini un tablou unidimensional ca un tablou care conţine doar elemente de tip simplu, iar dacă conţine cel puţin un element de tip compus, atunci tabloul va fi multidimensional. Voi dmeonstra mai întâi următoarea lemă:

Lema 4.3.2 *Un tablou unidimensional cu n elemente poate fi definit ca tip de date în XRDL,* $\forall n \in \mathbb{N}$.

Voi demonstra lema 4.3.2 folosind inducţia matematică. Pentru cazul $n = 1$ tabloul cu un element va avea ca formă generală următoarea structură:

```
<array>
   <data>
      <_tipSimpluDeDate>_valoare</_tipSimpluDeDate>
   </data>
</array>
```

`_tipSimpluDeDate` va fi înlocuit cu cu oricare dintre tipurile: string, int/i4, double, boolean, dateTime.iso8601 sau base64, iar `_valoare` va fi înlocuit cu o valoare corespunătoare tipului de date ales. Corespunzătorul acestui tablou în XRDL va fi următorul tip de date complex:

```
<type name="_tablou_1_elem">
   <member type="_tipSimpluDeDate"> _membru_1</member>
</type>
```

_tablou_1_elem este numele ales pentru noul tip de date definit, _tipSimpluDeDate este acelaşi tip definit mai sus, iar _membru_1 este numele dat membrului din tablou (Obs. Numele date tipului customizat de date definit sau a membrilor tipului sunt irelevante). Se observă că acest tip customiza de date din XRDL este echivalent cu cel anterior în XML.

Pentru pasul inductiv presupunem că un tablou unidimensional cu k elemente poate fi definit ca tip complex de date în XRDL. Rămâne de demonstrat că un tablou cu k+1 elemente poate fi de asemenea definit în XRDL. Fie tabloul cu k+1 elemente următorul:

```
<array>
   <data>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
      <value>
         <_tipSimpluDeDate_2>_valoare_2</_tipSimpluDeDate_2>
      </value>
      ...
      <value>
         <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
      </value>
      <value>
         <_tipSimpluDeDate_k+1>_valoare_k+1</_tipSimpluDeDate_k+1>
      </value>
   </data>
</array>
```

Conform presupunerii putem defini în XRDL un tip de date complex care să conţina primele k elemente ale tabloului. Fie acest tip definit astfel:

```
<type name="_tablou_k_elem">
   <member type="_tipSimpluDeDate_1">_ membru_1</member>
   <member type="_tipSimpluDeDate21">_ membru_2</member>
   ...
   <member type="_tipSimpluDeDate_k-1">_ membru_k-1</member>
   <member type="_tipSimpluDeDate_k"> _membru_k</member>
```

```
</type>
```

În cele ce urmează se poate defini un tablou similar prin extinderea acestuia cu un element:

```
<type name="_tablou_k+1_elem">
   <member type="_tipSimpluDeDate_1">_ membru_1</member>
   <member type="_tipSimpluDeDate_2"> membru_2</member>
   ...
   <member type="_tipSimpluDeDate_k-1">_ membru_k-1</member>
   <member type="_tipSimpluDeDate_k"> membru_k</member>
   <member type="_tipSimpluDeDate_k+1">_ membru_k+1</member>
</type>
```

Se oservă că elementul `_tablou_k+1_elem` este chiar echivalentul din XRDL al taboului din XML cu k+1 elemente. În concluzie, rezultăcă ipoteza indcuţie este adevărată pntru orice $n \in \mathbb{N}$.

În cele ce urmeazăse tratează cazul tablourilor multidimensionale (tablouri ce conţin cel puţin un element de tip tablou sau structură). Fie următoarea lemă:

Lema 4.3.3 *Un tablou n-dimensional poate fi definit ca tip de date în XRDL,* $\forall n \in \mathbb{N}$.

Un exemplu de tablou bidimensional este următorul:

```
<array>
   <data>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
      <value>
         <array>
            <data>
               <_tipSimpluDeDate_2>_valoare_2</_tipSimpluDeDate_2>
               <_tipSimpluDeDate_3>_valoare_3</_tipSimpluDeDate_3>
            </data>
         </array>
      </value>
   </data>
</array>
```

Înainte însă de a demonstra lema 4.3.3 trebuie tratat şi celălalt tip de date complex, structura. După cum am văzut în sectţ3.2.1 structura poate fi de asemenea unidimensională sau multidimensională. Prima lemă tratează structurile unidimensionale:

Lema 4.3.4 *O strucutră unidimensională cu n perechi de elemente nume-valoare poate fi definităca tip de date în XRDL,* $\forall n \in \mathbb{N}$.

Pentru cazul particular din cadrul inducţiei se tratează o strucutra cu o pereche nume-valoare:

```
<struct>
   <member>
      <name>_membru_1</name>
      <value>
         <_tipSimpluDeDate>_valoare_1</_tipSimpluDeDate>
      </value>
   </member>
</struct>
```

_tipSimpluDeDate va fi înlocuit cu unul din tipurile: string, int/i4, double, boolean dateTime.iso8601 sau base64. Tipul de date echivalent în XRDL va fi:

```
 <type name="structura_1_elem">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate">_valoare_1</member>
</type>
```

Pentru pasul inductiv se presupune că o structură unidimensională cu k elemente poate fi definită ca tip complex de date în XRDL. Rămâne de demonstrat că o structură unidimensională cu k+1 elemente poate fi definită în XRDL. Fie structura cu k+1 elemenete următoarea:

```
<struct>
   <member>
      <name>_membru_1</name>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
   </member>
   ...
   <member>
```

```
      <name>_membru_k</name>
      <value>
         <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
      </value>
   </member>
   <member>
      <name>_membru_k+1</name>
      <value>
         <_tipSimpluDeDate_k+1>_valoare_k+1</_tipSimpluDeDate_k+1>
      </value>
   </member>
</struct>
```

Dacă luăm în considerare doar primele k perechi ale structurii, acestea formează o altă structură cu k perechei nume-valoare. Această structură cu k elemente poate fi definită în XRDL conform presupunerii inductive astfel:

```
<type name="structura_k_elem">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1">_valoare_1</member>
   ...
   <member type="string">_membru_k-1</member>
   <member type="_tipSimpluDeDate_k-1">_valoare_k-1</member>
   <member type="string">_membru_k</member>
   <member type="_tipSimpluDeDate_k">_valoare_k</member>
</type>
```

Pornind de la această strutură se poate construi un alt tip complex care îl extinde pe acesta cu încă doi membrii (o pereche de tipul membru-valoare):

```
<type name="structura_k_elem">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1 ">_valoare_k</member>
   ...
   <member type="string">_membru_k</member>
   <member type="_tipSimpluDeDate_k">_valoare_k</member>
   <member type="string">_membru_k+1</member>
   <member type="_tipSimpluDeDate_k+1">_valoare_k+1</member>
</type>
```

Elementul `structura_k_elem` este echivalentul în XRDL al structurii cu k+1 elemente, ceea ce concluzionează inducţia. Astfel ipoteza inducţiei este adevărată pentru $\forall n \in \mathbb{N}$.

Revenim acum la demonstraţia lemei 4.3.3. Pentru cazul particular se observă că un tablou bidimensional trebuie să conţină cel puţin un element compus unidimensional de tip tablou sau structură şi eventual unul sau mai multe elemente de tip simplu de date. Dar lema 4.3.2 şi lema 4.3.4 demonstrează deja că orice tip de date compus unidimensional are un tip de date complex echivalent în XRDL. Forma generală a unui tablou bidimensional este:

```
<type name="_tablou_bidim_m_elem">
   <member type="_tip_1">_membru_1</member>
   <member type="_tip_2">_membru_2</member>
   ...
   <member type="_tip_m">_membru_m</member>
</type>
```

`_tip_i` reprezintă fie un tip simplu de date, fie un tip unidimensional de date compuse, $\forall i \in 1, 2, ..., m$, şi cu proprietatea că există cel puţin un $j \in 1, 2, ..., m$ pentru care `_tip_j` este un tip compus de date. Cazul particular pentru lema 4.3.3 este evident adevărat deoarece se referă la tablouri unidimensionale, adică ceea ce a fost demonstrat în lema 4.3.2. Pentru pasul inductiv presupunem că orice tablou k-dimensional poate fi definit în XRDL ca un tip complex de date. Se observă că un tablou (k+1)-dimensional conţine elemente de dimensiuni k sau mai mici. Din presupunerea inductivă deducem că orice element al tabloului (k+1)-dimensional poate fi definit în XRDL ca un tip de date complex. În concluzie putem defini tabloul (k+1)-dimensional folosind elemente de acele tipuri de date definite, iar lemma 4.3.3 este demonstrată.

Pentru structuri multi-dimensionale se defineşte următoarea lemă:

Lema 4.3.5 *O structură n-dimensională poate fi definită ca tip de date în XRDL,* $\forall n \in \mathbb{N}$.

Lema 4.3.5 se demonstrează prin acelaşi raţionament ca şi lemma 4.3.3, cu observaţia că unul din membrii perechii structurii (care reprezintă numele perechii respective) va fi în totdeauna de tip string.

4.3.2 Existenţa unui serviciu XML-RPC valid pentru orice document XRDL

Această secţiune va conţine demonstraţia următoarei teoreme:

Teorema 4.3.6 *Orice document XRDL descrie un serviciu XML-RPC valid.*

Din secţiunile anteriaore se observă că structura unui document XRDL permite descrierea unei metode de serviciu web şi, dacă este cazul, a unor tipuri de date compuse. În XRDL metodele serviciului web sunt descrise prin nume, datele de intrare şi datele de ieşire ale metodei. Putem deduce astfel că orice metodă descrisă în XRDL este o metodă validă XML-RPC dacă datele de intrare şi de ieşire sunt date valide XML-RPC. Având în vedere restricţia impusă asupra XRDL, toate tipuri de date simple din XRDL sunt date valide în XML-RPC. Rămâne de deomnstrat următoarea lemă:

Lema 4.3.7 *Orice tip compus de date din XRDL este un tip de date valid în XML-RPC.*

Pentru a demonstra această lemă şi pentru a păstra corespondeţele între aceleaşi tipuri de date, presupunem că XRDL respectă următoarele convenţii de denumire:

Propoziţia 4.3.8 *Tipurile compuse de date din documentul XRDL care descriu date XML-RPC de tip structură vor reflecta perechile structurii printr-o convenţie de nume: unul din membrii, care este de tip string şi reprezintă numele perechii va avea o denumire care începe cu* `_membru`*, iar celălalt membru care poate fi de orice tip valid de date va avea o denumire care începe cu* `_valoare`*. Tipruile compuse de date din documentul XRDL care descriu date XML-RPC de tip tablou vor conţine doar membrii a căror denumire începe cu* `_membru`*.*

Această propoziţie asigură faptul că tipurile de date compuse definite în XRDL pot fi diferenţiate în funcţie de corespondetul lor în XML-RPC. Fără această convenţie tipurile de date compuse din XRDL arată la fel, deoarece toate se definesc printr-o enumerare de date membre, fără vreo diferenţă de structură între ele.

Luând în considerare propoziţia 4.3.8, demonstraţia lemei 4.3.7 se poate împărţii în două părţi, tratând datele compuse care conţmembrii denumiţi cu `_membru` şi `_valoare` separat de celălalte tipuri compuse de date.

Se va trata mai întâcazul datelor compuse ce conţin membrii denumiţi cu `_membru` şi `_valoare`. Fie următoarea lemă:

Lema 4.3.9 *Un tip compus de date din documentul XRDL care conţine membrii denumiţi cu* `_valoare` *şi care are doar membrii de tipuri simple de date poate fi transcris în XML-RPC într-un element de tip structură.*

Din ipoteza lemei 4.3.9 deducem că elementul de tip compus va avea forma generală:

```
<type name="_elementCompus_n">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1">_valoare_1</member>
   ...
   <member type="string">_membru_n</member>
   <member type="_tipSimpluDeDate_n">_valoare_n</member>
</type>
```

_tipSimpluDeDate_i, $\forall i = \overline{1,n}$, va fi înlocuit cu unul din tipurile de date: string, int/i4, double, boolean, dateTime.iso8601 sau base64.

Pentru demonstraţia lemei vom folosi inducţia matematică după $n \in \mathbb{N}$. Pentru cazul de bază n=1 se obţine un element compus de forma:

```
<type name="_elementCompus_1">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1">_valoare_1</member>
</type>
```

Acesta poate fi transcris în XML-RPC printr-un element de tip structură:

```
<struct>
   <member>
      <name>_membru_1</name>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
   </member>
</struct>
```

Pentru pasul inductiv presupunem că lema este adevărată pentru $n = k$, adică orice tip compus de date cu $2 * k$ membrii poate fi transcris ca un element de tip structură în XML-RPC. Fie acest tip compus de date:

```
 <type name="_elementCompus_k">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1">_valoare_1</member>
   ...
   <member type="string">_membru_k</member>
   <member type="_tipSimpluDeDate_k">_valoare_k</member>
</type>
```

şi elementul din XML-RPC echivalent:

```
<struct>
   <member>
      <name>_membru_1</name>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
   </member>
   ...
   <member>
      <name>_membru_k</name>
      <value>
         <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
      </value>
   </member>
</struct>
```

Se observă că pentru pasul k+1 tipul compus de date va fi:

```
<type name="_elementCompus_k+1">
   <member type="string">_membru_1</member>
   <member type="_tipSimpluDeDate_1">_valoare_1</member>
   ...


   <member type="string">_membru_k</member>
   <member type="_tipSimpluDeDate_k">_valoare_k</member>
   <member type="string">_membru_k+1</member>
   <member type="_tipSimpluDeDate_k+1">_valoare_k+1</member>
</type>
```

Se observă că se poate extinde structura echivalentă cu tipul compus cu $2 * k$ membrii cu încă un membru pentru a obţine echivalentul pentru acest tip compus cu $2 * (k + 1)$ membrii. Astfel se pote transcrie sub forma:

```
<struct>
   <member>
      <name>_membru_1</name>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
```

```
   </member>
   ...
   <member>
      <name>_membru_k</name>
      <value>
         <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
      </value>
   </member>
   <member>
      <name>_membru_k+1</name>
      <value>
         <_tipSimpluDeDate_k+1>_valoare_k+1</_tipSimpluDeDate_k+1>
      </value>
   </member>
</struct>
```

Această structură este elemntul echivalent din XML-RPC căutat, ceea ce concluzionează demonstraţia lemei 4.3.9.

Ca o generalizare a lemei 4.3.9 avem:

Lema 4.3.10 *Un tip compus de date din documentul XRDL care conţine membrii denumiţi cu* ***_valoare*** *poate fi transcris în XML-RPC într-un element de tip structură.*

Se observă că această lemă nu mai are restricţia ca membrii să aibă doar tipuri simple de date. Acestă lemă tratează şi cazurile tipurilor compuse de date ce conţin alte tipuri compuse de date.

Înainte de a demonstra lema 4.3.10 trebuie tratate tipurile de date compuse care au ca echivalent în XRLD elemente de tip tablou. Se consideră următoarea lemă:

Lema 4.3.11 *Un tip compus de date dintr-un document XRDL care conţine doar membrii a căror denumire incepe cu* ***_membru*** *şi care are doar membrii de tipuri simple de date poate fi transcris în XML-RPC ca un element de tip tablou.*

Din ipoteza lemei deducem că forma generală a elementelor compuse descrise este:

```
<type name="_elementCompusTablou_n">
   <member type="_tipSimpluDeDate_1">_membru_1</member>
   ...
   <member type="_tipSimpluDeDate_n">_membru_n</member>
</type>
```

_tipSimpluDeDate_i, $\forall i = \overline{1,n}$, va fi înlocuit cu unul din tipurile de date: string, int/i4, double, boolean, dateTime.iso8601 sau base64. Această lemă se demonstrează cu ajutorul unei inducţii matematice după n. Pentru cazul particular n=1 tipul compus de date are forma:

```
<type name="_elementCompusTablou_1">
   <member type="_tipSimpluDeDate_1">_membru_1</member>
</type>
```

Acesta poate fi transcris în XML-RPC ca următorul tablou:

```
<array>
   <data>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
   </data>
</array>
```

Pentru pasul inductiv presupunem că un element compus cu k memebrii pote fi descris în XML-RPC ca un element de tip tablou. Rămâne de demonstrat că un element compus cu k+1 membrii poate fi de asemenea descris în XML-RPC. Fie elementul compus cu k membrii următorul:

```
<type name="_elementCompusTablou_k">
   <member type="_tipSimpluDeDate_1">_membru_1</member>
   ...
   <member type="_tipSimpluDeDate_k">_membru_k</member>
</type>
```

şi echivalentul său în XML-RPC:

```
<array>
   <data>
      <value>
         <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
      </value>
      ...
      <value>
         <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
      </value>
   </data>
```

```
</array>
```

Trebuie deomnstrat că tipul de date compus cu k+1 membrii are de asemenea un echivalent în XML-RPC. Fie acest tip compus reprezentat prin:

```
<type name="_elementCompusTablou_k+1">
  <member type="_tipSimpluDeDate_1">_membru_1</member>
  ...
  <member type="_tipSimpluDeDate_k">_membru_k</member>
  <member type="_tipSimpluDeDate_k+1">_membru_k+1</member>
</type>
```

Se observă că dacă se iau în considerare doar primii k membrii se poate construi un element de tip tablou în XML-RPC pe baza presupunerii inductive. Acest element tablou se poate extinde uşor adăugând un nou membru pentru a obţine un tablou echivalent cu tipul compus de date cu k+1 membrii:

```
<array>
  <data>
    <value>
      <_tipSimpluDeDate_1>_valoare_1</_tipSimpluDeDate_1>
    </value>
    ...
    <value>
      <_tipSimpluDeDate_k>_valoare_k</_tipSimpluDeDate_k>
    </value>
    <value>
      <_tipSimpluDeDate_k+1>_valoare_k+1</_tipSimpluDeDate_k+1>
    </value>
  </data>
</array>
```

Acest raţionament concluzionează demonstraţia lemei 4.3.11.

Revenind la demonstraţia lemei 4.3.10, trebuie demonstrat că tipuri compuse de date care conţin membrii denumiţi cu `_value` şi care conţin la rândul lor elemente compuse au echivalente în XML-RPC sub forma unor elemente de tip structură.

Se observă că un element compus poate include recursiv alte elemente compuse ca şi membrii. În exemplul următor `_elementCompusTablou1_1` conţine un membru de tipul `_elementCompusTablou2_1`.

```
<type name="_elementCompusTablou1_1">
```

```
   <member type="_elementCompusTablou2_1">_membru_1</member>
</type>
<type name="_elementCompusTablou2_1">
   <member type="_tipSimpluDeDate_1">_membru_1</member>
</type>
```

Se defineşte dimensiunea unui tip compus de date ca numă rul de tipuri compuse de date (inclusiv tipul compus în sine) incluse recursiv în structura tipului respectiv de date. În exemplul anterior elementul `_elementCompusTablou2_1` are dimensiunea 1, iar elementul `_elementCompusTablou1_1` are dimensiunea 2.

Dmonstraţia lemei 4.3.10 poate fi văzută ca o inducţie după $n \in \mathbb{N}$, unde n reprezintă dimensiunea tipului compus de date. Se observă că lema 4.3.9 reprezintă cazul particular cu n=1 pentru lema 4.3.10. Astfel primul pas pentru inducţie este demonstrat. În continoare utilizăm faptul că XML-RPC poate defini recursvi elemente de tip structură pentru orice dimensiune finită. Luând în considerare acest lucru, pasul inductiv devine evident, iar demonstraţia lemei 4.3.10 este în cheiată.

Pentru a trata şi ultimul tip de date compuse posibil în XRDL se mai defineşte următoarea lemă:

Lema 4.3.12 *Un tip compus de date dintr-un document XRDL care conţine doar membrii a căror denumire incepe cu* `_membru` *poate fi transcris în XML-RPC ca un element de tip tablou.*

Această lemă generalizează elementele de tip tablou în acelaşi sens ca lema 4.3.10 în cazul elementelor de tip structură. Pentru demonstraţia acestei leme se aplicaă un raţionament analog având în vedere că în XML-RPC se pot defini tablouri de orice dimensiune finită.

5. Aplicaţie

5.1 Motivarea şi scopul aplicaţiei

În capitolele anterioare am prezentat avantajele şi diferitele aplicaţii ale fişierelor de descriere a serviciilor web. Astfel, acestea se pot folosi pentru generarea automată de schelete pentru un serviciu web sau pentru un client care apelează metode ale serviciului web. De asemenea fără aceste fişiere de descriere nu este posibilă compararea serviciilor web. Un pas important către acceptarea generală a fişierlelor de descriere pentru servicii web este asigurarea faptului că definirea acestor fişiere nu îngreunează procesul de definire şi implementare al unui serviciu web.

Serviciile de tip SOAP şi REST care definesc ca standarde pentru fişierul de descriere al serviciului WSDL, respectiv WADL, au deja o serie de tool-uri implementate care automatizează crearea fişierului de descriere al serviciului. Astfel, implementarea serviciului în sine nu este complicată de existenţa fişierului de descriere. Mai mult decât atât, cu ajutorul fişierului de descriere al serviciului se pot generera părţi ale serviciului sau părţi ale clientului care apelează serviciul.

XML-RPC nu defineşte un standard pentru fişierul de descriere al serviciului, dar această lucrare prezintă formatul XRDL ca limbaj de descriere al serviciilor XML-RPC. După cum se observă din caracteristicile serviciului de tip XML-RPC prezentate în capitolele anterioare, avantajul principal al acestora este simplitatea. De accea obiectivul tool-ului de generare automată a fişiereului XRDL este ca dezvoltatorul să nu fie nevoit să creeze manual acest fişier.

Proiectul open source XRLD ([20]) conţine un set de tool-uri care automatizează generarea pentru servicii web scrise în C++/Qt şi PHP. Aplicaţia implementată de mine completează setul acesta cu un tool care generează fişierul XRDL pentru servicii web scrise în Java. În cele ce urmează voi prezenta structura aplciaţiei şi modul de integrare al acesteia cu un serviciu web XML-RPC.

5.2 Specificarea aplicaţiei

Aplicaţia dezvoltată generează o descriere în format XRDL a unui serviciu web implementat în Java. Pentru implementarea nivelului de logică al aplicaţiei am folosit cât mai mult principiul unei singure responsabilităţi, pentru a obţine un sistem slab cuplat (loosely coupled) şi uşor extensibil. Astfel, crearea documentului XRDL este împărţită în trei etape:

- Iniţializarea obiectelor necesare pentru crearea unui document XML
- Crearea corpului documentului XRDL, care conţine definirea tipurilor customizate de date şi a metodelor serviciului web
- Finalizarea şi salvarea documentului XML

Etapa de generare a corpului documentului XRDL procesează pe rând fiecare metodă oferită de serviciul web şi cu ajutorul unor funcţii private adaugă metodele în fişierul de descriere, împreună cu tipurile de date customizate, unde este cazul. În figura următoare se observă diagrama de clasă a generatorului de documente XRDL:

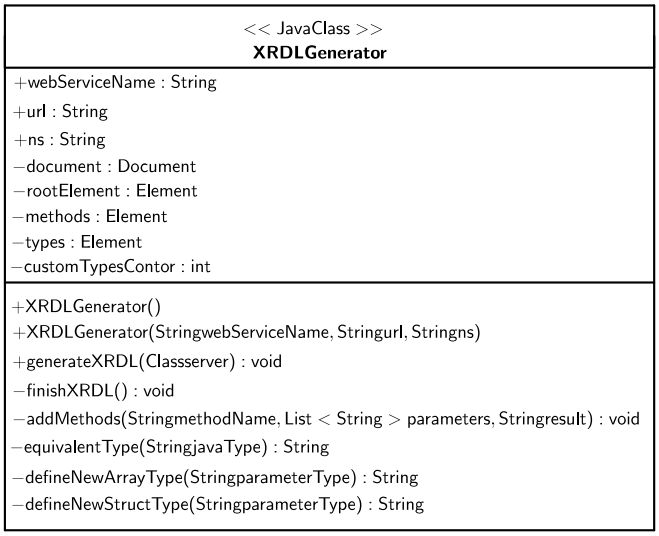

Figura 5.1: XRDLGenerator - diagrama de clasă

Singurele funcţii publice care pot fi apelate de un serviciu web sunt constructorul clasei XRDLGenerator şi metoda `generateXRDL`. Constructorul iniţializează elementele necesare pentru crearea unui document XML. Acesta poate fi apeltat cu sau fără parametrii, dar este preferabil să fie apelat cu parametrii pentru a oferi informaţii despre numele serviciului web, URL-ul la care este pus la dispoziţie şi namespace-ul ın care se află.

Funcţia `generateXRDL` primeşte ca parametru clasa obiectului de tip serviciu web XML-RPC şi pe baza informaţiilor obţinute prin reflecţie din acea clasă creează descrierea serviciului. Mai întâi se creează elementul rădăcină al documentului XRDL cu detaliile oferite prin parametrii constructorului (numele, URL-ul şi namespace-ul serviciului web). Apoi se parcurg pe rând toate metodele oferite de serviciul web şi se definesc în formatul specificat de XRDL. Dacă există parametrii sau rezultate ale metodelor care nu sunt de tip simplu de date, atunci se defineşte un tip customizat de date. Acest tip nou de date se defineşte în prima secţiune a documentului XRDL, după cum s-a specificat în capitolul 4.2.

După ce funcţia `generateXRDL` termină de procesat toate metodele serviciului web, se apelează funcţia `finishXRDL` care finalizează documentul şi îl salvează.

Funcţia `generateXRDL` foloseşte pentru procesarea metodelor un set de alte funcţii private ale generatorului de XRDL. Funcţia `addMethod` primeşte ca parametrii numele metodei serviciului web, lista tipurilor de parametrii ai metodei care trebuie definită şi tipul rezultatului returnat. În cazul în care metoda nu returnează nimic, atunci atributul `result` nu va apărea deloc în definirea metodei în XRDL, deoarece în formatul standard nu se pot specifica date de tip nul sau vid. Următoarea mostră de cod arată cum se prelucrează pe rând toţi parametrii metodei serviciului web. Pe parcurs ce se prelucrează un parametru se adaugă elementul `member` corespunzător în formatul XRDL:

```
for (String param : parameters) {
      Element parameterElement = document.createElement("member");
      parameterElement.setAttribute("type", param);
      parameterElement.appendChild(document.createTextNode("param_"+contor));
      contor++;
      methodElement.appendChild(parameterElement);
}
methods.appendChild(methodElement);
```

Funcţia `equivalentType` primeste ca parametru un tip de date din Java şi returnează tipul de date din XRDL echivalent tipului respectiv. În codul sursă de mai jos se pot identifica echivalenţele între tipurile de date din Java şi cele din XRDL:

```
switch(javaActualType ) {
case "Integer":  XMLtype = "int";
                 break;
case "int":      XMLtype = "int";
                 break;
```

```
case "String" :  XMLtype = "string";
                 break;
case "Double" :  XMLtype = "double";
                 break;
case "Date" :    XMLtype = "dateTime.iso8601, ";
                 break;
case "byte" :    XMLtype = "base64";
                 break;
case "Boolean" : XMLtype = "boolean";
                 break;
case "List" :    XMLtype = "array";
                 break;
case "Map" :     XMLtype= "struct";
                 break;
}
```

Cele două metode defineNewArrayType şi defineNewStructType primesc ca parametru tipul de date al elementelor tabloului, respectiv structurii şi definesc un tip customizat de date. Aceste tipuri de date se adaugă în prima parte a documentului XRDL, în tag-ul `types` care conţine toate tipurile de date compuse definite.

Deşi generarea documentului XRDL foloseşte diferite funcţii, dezvoltatorul unui serviciu web nu trebuie să aibă cunoştiinţă de acestea, iar utilizarea generatorului automat este foarte simplă. În cele ce urmează voi prezenta un exemplu de serviciu web pentru a arăta cum se integrează generatorul automat de XRDL cu serviciul web XML-RPC.

5.3 Integrarea aplicaţiei cu servicii web existente

Se prezintă în cele ce urmează un exemplu de serviciu web folosind una din distributiile Java pentru XML-RPC, şi anume Apache XML-RPC. Vom prezenta mai întâi serviciul web fără generarea fişierului de descriere, urmând ca apoi să arătăm cum se poate integra generatorul de XRDL cu serviciul web. Acelaşi procedeu se va putea urma pentru toate serviciile web existente, generând astfel fişierul de descriere fără a fi nevoie de foarte mult modificări.

5.3.1 Exemplu de serviciu web XML-RPC folosind Apache XML-RPC

blaablaa

5.3.2 Generarea documentului XRDL pentru serviciul web

Pentru a genera documentul XRDL nu trebuie modificată implementarea serviciului web, acesta putând fi făcută după pornirea serviciului, care se face de obicei prin metoda `start`. Se creează mai întâi un obiect de tip XRDLGenerator care se iniţializează cu detaliile serviciului web: numele, URL-ul la care este disponibil şi namespace-ul în care se află. Apoi se apelează metoda `generateXRDL` a obiectului creat. La apelul metodei se transmite ca parametru clasa serviciului web obţinută prin reflecţie. Astfel, se adaugă următoarea bucată de cod:

```
String name = "TestXML-RPC";
String ns = "";
String url = "http://localhost:1002/";
XRDLGenerator generator = new XRDLGenerator(name, ns, url);
Class class1 = this.getClass();
generator.generateXRDL(class1);
```

Utilizând acest mecanism simplu, pentru modelul de serviciu web prezentat, s-a generat următorul fişier de descriere XRDL: blaablaa

5.4 Posibile extinderi

Aplicaţia implementată tratează doar cazul serviciilor web scrise în Java. Proiectul open source XRDL conţine tool-uri similare pentru generarea fişierelor de descriere a servciilor implementate în PHP sau C++/Qt. Setul acestor tool-uri de generare automată a descriptorului XRDL poate fi extins pentru servicii web implementate în alte limbaje, cum ar fi C# sau Python.

De asemenea aplicaţia prezentată poate fi extinsă pentru a suporta mai multe tipuri de date. Unele implementări de XML-RPC oferă opţiunea de a extinde standardul pentru a folosi diferite tipuri de date. De exemplu, distribuţia folosită în exemplul anterior, Apache XML-RPC, oferă opţiunea `enabledForExtensions` prin care mai multe tipuri de date devin valide în XML-RPC. Aceste tipuri suplimentare de date sunt specificate prin namespace-ul `ns` care face referire la `http://ws.apache.org/xmlrpc/namespaces/`

`extensions`. Câteva exemple de tag-uri XML din acestă extensie sunt: ex:nil, ex:float, ex:bigdecimal, etc [21].

6. Concluzii

Serviciile web reprezintă tehnologia cea mai dezvoltată pentru implementarea aplicaţiilor distribuite. Acest lucru se datorează în primul rând principalului avantaj: interoperabilitatea. Cele trei tipuri de servicii web utilizate în prezent sunt: SOAP, REST şi XML-RPC, care se dezvoltă pe direcţii diferite. Principala componentă care ajută la uniformizarea serviciilor web este fişierul de descriere al serviciului, însă dintre cele trei tipuri de servicii, XML-RPC nu propune un standard pentru fişierul de descriere. De aceea, pentru a contribui la uniformizarea serviciilor, în această lucrare am tratat formatul XRDL de descriere al serviciilor de tip XML-RPC.

XRDL este un proiect open source ce conţine o serie de aplicaţii pentru serviciile sau clienţii implementaţi în C++/Qt sau PHP. Ceea ce îi lipseşte acestui proiect este o demonstraţie a echivalenţei dintre formatul XRDL şi standardul XML-RPC. Acesta este motivul pentru care, în această lucrare am prezentat o demonstraţie a faptului că XRDL este un limbaj de descriere valid pentru serviciile XML-RPC.

Ca parte aplicativă am completat setul de aplicaţii existente în proiectul open source cu un tool care generează un fişier de descriere XRDL pentru un serviciu XML-RPC implementat în Java. Avantajul soluţii implementate îi revine din faptul că este independentă de distribuţia XML-RPC folosită. Utilizând această aplicaţie se vor putea genera automat fişierele de descriere ale serviciilor XML-RPC fără a modifica implementarea serviciului sau a scrie manual descrierea serviciului. Din exemplul prezentat se poate observa uşurinţa utilizării tool-ului de generare a documentului XRDL.

În concluzie, contribuţiile personale din această lucrare ajută la uniformizarea serviciilor web şi la generarea automată a descrierii serviciilor web. Pe viitor, doresc să extind aplicaţia pentru a trata mai multe tipuri de date, dar şi servicii web implementate în alte limbaje de programare. De asemenea, pornind de la fişierul de descriere generat, se vor putea implementa scheleturi pentru servicii sau clienţi care apelează serviciile web.

Bibliografie

[1] F. Boian, Servicii Web; Modele, Platforme, Aplicaţii, editura Albastră , Cluj-Napoca, 2011

[2] F. Boian, B. Jancso, Uniform Solutions for Web Services, Studia Univ. Babe-Bolyai, vol. 57, no. 3, 2012, pg 13-23

[3] F.M. Boian, A. Ploscar, R.F. Boian, Web Service Matching, Knowledge Engineering Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2013 Cluj-Napoca (România), July 4-6, 2013, sub tipar

[4] E. Cerami, Web Services Essentials, O'Reilly Media, 2002

[5] D.A. Chappel, T. Jewell, Java Web Services, O'Reilly Media, 2002

[6] P. Festa, W3C defines Web services, http://news.cnet.com/2100-1001-965887.html

[7] M. Kalin, Java Web Services up and running, O'Reilly Media, 2009

[8] R. R. Khorasgani, E. Stroulia, O. R. Zaiane, Web service matching for RESTful web services, In Proceedings of Web Systems Evolution (WSE), 2011, pg 115-124

[9] S. St. Laurent, J. Johnston, E. Dumbill, Programming Web Services with XML-RPC, O'Reily Media, 2001

[10] D. Troancă, F. Boian, XRDL: A VALID DESCRIPTION LANGUAGE FOR XML-RPC, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2013 Cluj-Napoca (România), July 4-6, 2013, sub tipar

[11] D. Winer, XML-RPC Specification, http://xmlrpc.scripting.com/spec.html

[12] Distributed Computing Environment,

http://en.wikipedia.org/wiki/Distributed_Computing_Environment

[13] History of Web Services, http://www.servicestack.net/docs/framework/historyof-webservices

[14] SOAP, http://en.wikipedia.org/wiki/SOAP#Advantages

[15] Web Application Description Language, http://en.wikipedia.org/wiki/Web_Application_Description

[16] Web service, http://en.wikipedia.org/wiki/Web_service

[17] WSDL, http://ro.wikipedia.org/wiki/WSDL

[18] XML Introduction - What is XML?, http://www.w3schools.com/xml/xml_whatis.asp

[19] XML-RPC, http://en.wikipedia.org/wiki/XMLRPC

[20] xrdl - XML-RPC Description Language, http://code.google.com/p/xrdl/

[21] Data Types, http://ws.apache.org/xmlrpc/types.html