CONTRIBUŢII PRIVIND ARHITECTURA ŞI FIABILITATEA SISTEMELOR INFORMATICE DISTRIBUITE ORIENTATE PE SERVICII WEB

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea "Politehnica" din Timişoara
în domeniul ŞTIINŢA CALCULATOARELOR
de către

ing. CEZAR-FLORIN TOADER

Conducător ştiinţific:	prof.univ.dr.ing. NICOLAE ROBU
Referenţi ştiinţifici:	prof.univ.dr.ing. MIRCEA PETRESCU
	prof.univ.dr.mat. VIOREL NEGRU
	prof.univ.dr.ing. VLADIMIR CREŢU

Ziua susţinerii tezei: 23.12.2011

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare şi tehnologia informaţiei

Universitatea „Politehnica” din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

CUVÂNT ÎNAINTE

Această parte a lucrării de faţă, numită „Cuvânt înainte”, îmi oferă un minunat prilej de a exprima gânduri alese de preţuire şi recunoştinţă pentru cei care, direct sau indirect, m-au susţinut şi încurajat de-a lungul pregătirii mele şi astfel, au contribuit la formarea şi finalizarea acestei teze de doctorat.

Am cunoscut la Facultatea de Automatică şi Calculatoare, din cadrul Universităţii „Politehnica” Timişoara, oameni deosebiţi care, de-a lungul mai multor ani, cu ocazia examenelor şi rapoartelor de cercetare pentru doctorat, fie mi-au fost examinatori, fie mi-au analizat rapoartele de cercetare, fie şi-au făcut timp şi au avut răbdarea de a discuta despre preocupările mele şi despre drumul ce ar trebui să-l urmez în pregătirea mea pentru doctorat.

Doresc să exprim calde mulţumiri şi gânduri sincere de preţuire coordonatorului meu, prof.univ.dr.ing. Nicolae Robu, Rectorul Universităţii „Politehnica” din Timişoara, cel care a văzut în mine un doctorand deschis către cunoaştere şi perfecţionare şi care, într-un moment mai dificil al evoluţiei mele profesionale, a acceptat să mă sprijine pe acest drum al pregătirii ştiinţifice de nivel doctorat şi, astfel, am ajuns şi la acest moment al finalizării tezei.

Domnului Decan al Facultăţii de Automatică şi Calculatoare, prof.univ.dr.ing. Octavian Proştean, îi mulţumesc pentru sprijinul acordat de-a lungul timpului şi pentru sugestiile pe care mi le-a dat în numeroasele noastre discuţii în legătură cu organizarea acestei teze.

În anii de început ai pregătirii mele de specialitate am avut ocazia de a cunoaşte, de a discuta şi de a susţine examene cu profesori din Departamentul de Calculatoare. Nu am uitat acea perioadă şi primele aprecieri şi încurajări primite de la prof.univ.dr.ing. Jurca Ioan, căruia îi mulţumesc pe această cale.

De asemenea, doresc să-i mulţumesc d-lui prof.univ.dr.ing. Ionel Jian, din Departamentul de Calculatoare, care mi-a fost şi examinator dar, mai cu seamă, a fost un om deschis care a înţeles efortul meu în acest domeniu ştiinţific şi a avut cuvinte de apreciere la susţinerea tezei în departament.

Pentru ajutorul primit şi pentru cuvintele de apreciere doresc să-i mulţumesc d-lui prof.univ.dr.ing. Daniel Curiac, iar pentru sfaturile utile mulţumesc d-lui prof.univ.dr.ing. Ioan Filip, ambii din cadrul Departamentului de Automatică şi Informatică Industrială.

Tuturor membrilor Facultăţii de Automatică şi Calculatoare, care de-a lungul anilor de pregătire m-au ajutat cu idei, sfaturi sau documentaţie de specialitate le mulţumesc acum.

Domnilor referenţi ştiinţifici le mulţumesc pentru că au acceptat să facă parte din comisie şi pentru încrederea acordată.

Finalizarea acestei teze nu ar fi fost posibilă fără sprijinul permanent al soţiei mele, Rita Toader, căreia îi mulţumesc pe această cale pentru răbdare şi devotament.

Cezar Toader

Familiei mele,

Toader, Cezar-Florin

CONTRIBUŢII PRIVIND ARHITECTURA ŞI FIABILITATEA SISTEMELOR INFORMATICE DISTRIBUITE ORIENTATE PE SERVICII WEB

Teze de doctorat ale UPT, Seria 10, Nr. 38, Editura Politehnica, 2011, 108 pagini, 39 figuri, 2 tabele.

ISSN: 1842-7707

ISBN: 978-606-554-421-5

Cuvinte cheie:
servicii Web, arhitecturi orientate pe resurse, ROA, principii REST, arhitecturi orientate pe servicii, SOA, fiabilitate, dependabilitate, toleranţă la defecte, replicare, mesaje, protocoale, securitate.

Rezumat
Această lucrare abordează probleme de arhitectură şi de fiabilitate din domeniul sistemelor distribuite. S-au analizat arhitecturi actuale ale aplicaţiilor Web şi concepte legate de adresabilitatea resurselor. A fost propusă o arhitectură modernă de aplicaţie Web care permite integrarea mai multor tipuri de resurse şi rezolvă problemele de acces uniform la resurse. Au fost analizate conceptele de toleranţă la defecte, securitate şi fiabilitate pentru sistemele distribuite bazate pe servicii Web. A fost modelat un sistem distribuit în care un proces manager coordonează procese executant aflate pe gazde diferite. Pe baza acestuia a fost propus un sistem tolerant la defecte, care foloseşte tehnica replicării proceselor pe gazde la distanţă. Au fost definite şi analizate modurile de operare şi protocoalele de funcţionare. Pe baza modelelor prezentate a fost definită o arhitectură de sistem fiabil cu replicare manageriată. Au fost definite componentele sistemului şi rolurile lor. S-au prezentat detalii de implementare legate de funcţionarea sistemului şi resincronizarea serviciilor. A fost analizată problema securizării mesajelor transmise pe reţea între manager şi executanţi. A fost propusă o tehnică de protejare cu lacăt informatic. Au fost analizate atacurile posibile şi s-a arătat că metoda propusă oferă nivelul de securitate dorit. A fost definit un parametru specific pentru a arăta influenţa replicării asupra timpului de răspuns. S-au prezentat detalii de implementare şi rezultate experimentale.

CUPRINS

1. INTRODUCERE 9
   1.1. Tematica abordată şi obiectivele propuse 9
   1.2. Organizarea tezei 9
2. ARHITECTURI, RESURSE ŞI SERVICII ÎN WEB-UL PROGRAMABIL 12
   2.1. Adresabilitatea resurselor Web 12
   2.2. Operaţii asupra resurselor 13
   2.3. Arhitecturi Web actuale 15
      2.3.1. Arhitecturi RESTful 15
      2.3.2. Arhitecturi stil RPC 16
      2.3.3. Arhitecturi mixte RESTful-RPC 17
   2.4. Servicii Web 18
   2.5. O scurtă taxonomie a serviciilor şi stărilor 19
   2.6. Implementări stateless, interfeţe stateful 20
   2.7. Structuri WS-Resource şi proprietăţile ACID 21
   2.8. Apelarea resurselor algoritmice 22
   2.9. Arhitecturi RESTful stratificate 24
   2.10. Adaptarea identificatorilor resurselor 28
   2.11. Concluzii şi contribuţii 30
3. FIABILITATEA SERVICIILOR WEB 32
   3.1. Disponibilitate, siguranţă, fiabilitate 32
   3.2. Tehnici tradiţionale de replicare 33
   3.3. Caracteristicile replicării optimiste 35
   3.4. Elementele replicării optimiste 36
   3.5. Concluzii şi contribuţii 37
4. MODELAREA UNUI SISTEM DISTRIBUIT CU MANAGEMENT CENTRALIZAT 39
   4.1. Procese şi conexiuni 39
   4.2. Localizarea şi rolul proceselor 40
   4.3. Problema acordului distribuit 41
   4.4. Procese şi operaţii 42
   4.5. Niveluri, componente şi evenimente 43
   4.6. Comunicaţia între procese 46
   4.7. Ordinea cauzală a mesajelor 47

4.8. Sistem distribuit cu management centralizat şi execuţie separată 49
4.9. Schimbul de mesaje între procesele client şi manager 51
4.10. Schimbul de mesaje între procesele manager şi executant 52
4.11. Concluzii şi contribuţii 54

5. MODEL DE REPLICARE CU SEPARAREA MANAGEMENTULUI DE EXECUŢIE 55
5.1. Introducere 55
5.2. Obiectivele sistemului cu replicare 56
5.3. Componentele sistemului cu replicare 57
5.4. Moduri de operare 60
5.5. Faze de execuţie şi protocoale 61
5.5.1. Protocolul componentei de execuţie 63
5.5.2. Protocolul componentei de management pentru operarea normală a sistemului 65
5.5.3. Protocolul componentei de management pentru operarea degradată a sistemului 67
5.5.4. Protocolul componentei de management pentru operarea minimală a sistemului 68
5.5.5. Protocolul pentru resincronizarea serviciilor 69
5.6. Concluzii şi contribuţii 70

6. ARHITECTURA UNUI SISTEM FIABIL CU REPLICARE MANAGERIATĂ 72
6.1. Introducere 72
6.2. Arhitectura sistemului 73
6.3. Ipoteze iniţiale 75
6.4. Modul normal de operare 75
6.5. Resincronizarea serviciilor în sistem 77
6.6. Securizarea comunicaţiilor între servicii 79
6.6.1. Stabilirea părţilor secretului 80
6.6.2. Reconstituirea secretului 80
6.6.3. Securizarea difuzării mesajelor 83
6.6.4. Protejarea cheii de decriptare 85
6.6.5. Protocolul pentru crearea mesajelor securizate 87
6.6.6. Protocolul pentru decriptarea mesajelor securizate 90
6.6.7. Modul de protecţie împotriva atacurilor 91
6.7. Abordări experimentale 92
6.8. Concluzii şi contribuţii 95

7. CONCLUZII ŞI CONTRIBUŢII PERSONALE 97

BIBLIOGRAFIE 101

LISTA FIGURILOR

Fig. 2.1. Simbolizarea unui serviciu Web 18
Fig. 2.2. Clasificarea serviciilor după modul de păstrare a stării 20
Fig. 2.3. O arhitectură posibilă a unei aplicaţii Web complexe 25
Fig. 2.4. Arhitectura RESTful Multilayer 27
Fig. 2.5. Modificarea URI-urilor de către Dispecerul de Resurse.....29
Fig. 4.1. Comunicaţia între procese.....39
Fig. 4.2. Un proces emiţător şi multe procese receptoare.....40
Fig. 4.3. Procese localizate pe gazde diferite 40
Fig. 4.4. Procesul S comunică cu mai multe procese.....40
Fig. 4.5. Procesul M are rol de manager pentru procesele executante X1, X2 41
Fig. 4.6. Etapele unei operaţii în cadrul unui proces dintr-un sistem distribuit 42
Fig. 4.7. Modelul de proces cu mai multe componente.....43
Fig. 4.8. Comunicaţii între nivelurile unui proces.....45
Fig. 4.9. Desfăşurarea în timp a comunicaţiei între două procese 47
Fig. 4.10. Ordinea cauzală a mesajelor.....48
Fig. 4.11. Sistem distribuit cu management centralizat şi execuţie separată 50
Fig. 4.12. Schimbul de mesaje între procesul client C şi procesul manager M 51
Fig. 4.13. Schimbul de mesaje între procesele manager şi executant.....53
Fig. 5.1. Succesiunea operaţiilor principale din sistem 58
Fig. 5.2. Componentele principale ale sistemului.....59
Fig. 5.3. Diagrama stărilor sistemului şi a tranziţiilor dintre stări 61
Fig. 5.4. Protocolul de operare normală a unui serviciu Worker 63
Fig. 5.5. Detalii despre protocolul de operare normală a unui serviciu Worker 65
Fig. 5.6. Protocolul pentru modul de operare normală a componentei Manager.....65
Fig. 5.7. Detalii despre protocolul de operare normală a componentei Manager 67
Fig. 5.8. Protocolul de operare degradată a componentei Manager 67
Fig. 5.9. Detalii despre protocolul de operare degradată pentru Manager.....68
Fig. 5.10. Protocolul de operare minimală a componentei Manager.....68
Fig. 5.11. Detalii despre protocolul de operare minimală pentru Manager 69
Fig. 5.12. Protocolul pentru resincronizare.....70

Fig. 6.1. Componentele sistemului distribuit. 73
Fig. 6.2. Arhitectura propusă (cazul cu 3 servicii *worker*) 74
Fig. 6.3. Difuzare de mesaje necriptate către serviciile worker 83
Fig. 6.4. Difuzare de mesaje securizate către serviciile worker 84
Fig. 6.5. Mesaj securizat difuzat de manager către un serviciu worker 85
Fig. 6.6. Schema protocolului de creare a mesajelor securizate....89
Fig. 6.7. Schema protocolului de decriptare a mesajelor securizate....91
Fig. 6.8. Procesare rapidă pentru formarea răspunsului sistemului 94
Fig. 6.9. Procesare sigură pentru formarea răspunsului sistemului....94

LISTA TABELELOR

Tabelul 6.1. Înregistrarea cererilor 76
Tabelul 6.2. Structura de date WS-Status....76

1. INTRODUCERE

1.1. TEMATICA ABORDATĂ ŞI OBIECTIVELE PROPUSE

În această teză sunt studiate tematici importante din domeniul sistemelor distribuite:

- Arhitecturi ale aplicaţiilor Web, probleme de adresabilitate a resurselor Web şi operaţii asupra acestora;
- Servicii Web, arhitecturi orientate pe servicii, resurse algoritmice şi arhitecturi stratificate, integrarea aplicaţiilor Web non-RESTful în arhitecturi RESTful;
- Disponibilitatea, siguranţa şi fiabilitatea serviciilor Web;
- Tehnici tradiţionale de replicare comparativ cu replicarea optimistă;
- Modelarea sistemelor distribuite cu management centralizat şi execuţie separată pe mai multe procese, comunicaţia între procese, ordinea cauzală a mesajelor, problema acordului distribuit;
- Modelarea sistemelor cu replicare coordonată centralizat, planificarea fazelor de execuţie şi stabilirea protocoalelor;
- Arhitectura sistemelor fiabile cu replicare manageriată.
- Securizarea comunicaţiilor între componentele sistemelor cu replicare.

Principalele obiective ale acestei teze sunt:

- Studiul tendinţelor actuale în organizarea resurselor aplicaţiilor Web, clarificarea arhitecturilor orientate pe resurse;
- Elaborarea unei arhitecturi RESTful stratificate capabilă să integreze şi aplicaţii Web non-RESTful;
- Studiul tendinţelor actuale în creşterea fiabilităţii sistemelor distribuite;
- Elaborarea unui model de sistem distribuit cu procese de execuţie coordonate separat de un subsistem de management;
- Elaborarea unei arhitecturi de sistem distribuit cu replicare manageriată şi precizarea modurilor de operare, a fazelor de execuţie şi a protocoalelor.
- Elaborarea unui protocol de securizare a mesajelor în sistemele distribuite cu replicare manageriată.

1.2. ORGANIZAREA TEZEI

Conţinutul acestei teze este organizat pe mai multe capitole care abordează probleme distincte şi prezintă studiile şi concluziile autorului.

În Capitolul 1, sunt prezentate cadrul general în care se încadrează teza, tematicile abordate, obiectivele principale şi modul de organizare a tezei.

În Capitolul 2, intitulat *Arhitecturi, resurse şi servicii în Web-ul programabil*, se analizează probleme actuale specifice dezvoltării aplicaţiilor Web şi tehnologiei identificatorilor de resurse, se analizează operaţiile efectuate asupra resurselor în cadrul aplicaţiilor Web şi se definesc trei tipuri de arhitecturi întâlnite în aplicaţiile

Web actuale. Scopul clarificării acestor arhitecturi este de a aduce Web-ului programabil un plus de claritate prin creşterea gradului de uniformitate în modul de apelare a resurselor, prin obţinerea unor avantaje maxime pe baza capabilităţilor protocolului HTTP, şi, în final, prin crearea posibilităţii de integrare a aplicaţiilor mai vechi existente în cadrul companiilor în noile arhitecturi ale aplicaţiilor Web.

În acest capitol se prezintă, în continuare, conceptul de serviciu Web, caracteristicile acestor servicii, impactul lor asupra dezvoltării aplicaţiilor distribuite şi o clasificare a stărilor şi serviciilor. Se continuă cu prezentarea unei modalităţi de formare a resurselor de tip serviciu Web, care, astfel, devin componente *stateful* în aplicaţii distribuite. Acest tip de resurse vor fi folosite în cadrul tezei, într-un capitol ulterior. Capitolul continuă cu prezentarea conceptului de resurse algoritmice şi a arhitecturilor RESTful stratificate. În acest capitol autorul propune o arhitectură orientată pe resurse numită *Arhitectură RESTful Multilayer*.

Autorul organizează aplicaţiile Web şi resursele (în sens generalizat) pe trei niveluri, clarifică rolul entităţilor software de pe fiecare nivel şi arată modul cum interacţionează entităţile software aflate pe niveluri diferite. Această arhitectură stratificată respectă două principii: *principiul adresabilităţii resurselor* şi, respectiv, *principiul interfeţei uniforme pentru accesul la resurse*.

Capitolul 3 este intitulat *Fiabilitatea serviciilor Web*. În acest capitol se trece de la problemele privind organizarea aplicaţiilor distribuite, care au fost tratate în capitolul anterior, la probleme de fiabilitate a sistemelor distribuite. Se prezintă, mai întâi, o analiză a conceptelor actuale legate de disponibilitate, siguranţă, toleranţă la defecte şi fiabilitate, ca un concept integrator. Se continuă cu analiza replicării optimiste, tehnologie folosită în această teză într-un capitol ulterior. Se prezintă caracteristicile şi elementele replicării *optimiste*, comparativ cu replicarea *pesimistă*.

În Capitolul 4, intitulat *Modelarea unui sistem distribuit cu management centralizat*, se prezintă pe etape dezvoltarea unui model de sistem distribuit în care elementele abstracte primordiale sunt *procesul* şi *conexiunea (legătura) între procese*. Se tratează problema comunicaţiei între procese în cadrul sistemului distribuit, conceptul de *mesaj între procese*, şi se prezintă *problema acordului distribuit*. În continuare se trece la abstractizarea elementelor din cadrul unui proces, respectiv, niveluri, componente şi evenimente. Se prezintă importanţa conceptului de *timp logic* în cadrul unui sistem distribuit şi, pe baza acestuia, se defineşte conceptul abstract de *ordine cauzală a mesajelor*. Toate aceste concepte sunt necesare pentru definirea, în cadrul acestui capitol, a unui model de sistem distribuit în care unele procese au *rol de management* al operaţiilor desfăşurate de alte procese care au doar *rol de execuţie*. După prezentarea proceselor componente ale sistemului, în acest capitol se abstractizează *schimbul de mesaje între procesele* sistemului distribuit. În acest capitol se propune şi se analizează un model abstract de sistem distribuit în care sunt modelate procese executant controlate de un proces manager, care rulează la distanţă, pe o altă gazdă, faţă de procesele executant. Modelul prezentat în acest capitol este la baza dezvoltării, într-un capitol ulterior, a unui sistem distribuit tolerant la defecte, care foloseşte replicarea operaţiilor.

Capitolul 5 este intitulat *Model de replicare cu separarea managementului de execuţie*. Pornind de la modelul abstract de sistem distribuit prezentat anterior se propune în acest capitol un model de replicare a operaţiilor şi datelor pe structura unui sistem distribuit în care există mai multe procese de execuţie. După prezentarea unei liste de obiective pentru sistemul tolerant la defecte, se propune *separarea operaţiilor de management de operaţiile efective de execuţie şi stocare de date* în cadrul sistemului distribuit. În acest capitol se propune utilizarea unui proces central, special destinat pentru operaţii de coordonare şi control a altor procese

destinate operaţiilor de execuţie. Într-o primă etapă, acest sistem distribuit este analizat din punctul de vedere al stărilor şi, respectiv, al modurilor de operare. În a doua etapă, se prezintă fazele de execuţie şi protocoalele pentru componentele esenţiale ale sistemului cu replicare, respectiv, componenta de management şi componentele de execuţie. Aceste faze de execuţie sunt analizate diferenţiat pe modurile de execuţie ale sistemului ca întreg, moduri prezentate în acest capitol.

În Capitolul 6, intitulat *Arhitectura unui sistem fiabil cu replicare manageriată*, se prezintă un mod de implementare a modelului de replicare prezentat în capitolul anterior şi se arată în detaliu ipotezele de lucru, modul de operare al acestui sistem cu replicare coordonată de un proces separat, protocoalele de funcţionare, modul de resincronizare a replicilor şi alte detalii de implementare.

Arhitectura propusă este compusă, în esenţă, din componente cu următoarele roluri: înregistrarea cererilor clienţilor, execuţia operaţiilor cerute de client, coordonare şi control, înregistrarea stării serviciilor Web cu rol de execuţie şi altele. Pe baza informaţiilor primite de la subsistemele de înregistrare, managerul serviciilor Web este capabil să decidă în ce stare este fiecare serviciu Web şi când trebuie făcută resincronizarea sistemului. Pentru aplicaţia client acest sistem apare ca un software unitar, capabil să rezolve intern defectele de sistem apărute şi să dea un răspuns corespunzător la cererea făcută, chiar dacă uneori apare o întârziere datorată procesului de resincronizare a unor servicii replică.

Sunt prezentate în acest capitol aspecte ale implementării şi detalii despre rezultatele experimentale. S-a propus *un nou parametru al sistemelor distribuite tolerante la defecte* care să exprime influenţa fazei de replicare a serviciilor Web asupra timpului de răspuns, numit *Relative Delay*. Această mărime relativă permite evaluarea modificărilor performanţei sistemului faţă de valori de referinţă. Abordările experimentale au arătat o influenţă a fazei de replicare asupra performanţei globale a sistemului, dar valorile întârzierilor sistemului pot fi menţinute la valori acceptabile. Monitorizarea evoluţiei în timp a valorilor acestui parametru permite aprecierea scăderilor de performanţă a sistemului faţă de o stare iniţială şi deschide noi posibilităţi de control şi optimizare.

În acest capitol este abordată şi problema securităţii sistemului distribuit, ca o componentă importantă a conceptului de *fiabilitate*. Pentru arhitectura de sistem cu replicare manageriată se studiază problema difuzării mesajelor de către componenta de management către serviciile pe care le coordonează. În scopul protejării sistemului de atacuri informatice se analizează problema securizării mesajelor difuzate în cadrul acestui sistem cu replicare. Se propune în acest capitol un protocol pentru crearea de mesaje securizate folosind o tehnică de protejare a informaţiei sensibile cu un *lacăt informatic*, iar algoritmul folosit este bazat pe polinoame de interpolare Lagrange.

Capitolul 7 prezintă *Concluzii şi contribuţii personale* în cadrul acestei teze.

2. ARHITECTURI, RESURSE ŞI SERVICII ÎN WEB-UL PROGRAMABIL

2.1. ADRESABILITATEA RESURSELOR WEB

World Wide Web nu este o tehnologie nouă. De la apariţia sa, Web-ul a cunoscut o evoluţie constantă, dar mulţi autori susţin că sunt aşteptate noi schimbări. Potenţialul Web-ului pentru a susţine programarea distribuită este subiectul din atenţia specialiştilor.

Dezvoltarea unor arhitecturi complexe bazate pe servicii Web (similare cu DCOM şi CORBA) determină o îndepărtare de simplitatea iniţială a Web-ului [19].

Astăzi există voci care susţin că există pericolul ca arhitecturile complexe bazate pe servicii Web ar putea să modifice sau să ignore trăsăturile care au adus succes Web-ului încă de la început [45].

S-a dezvoltat treptat conceptul de *arhitecturi orientate pe resurse* (*ROA – Resource Oriented Architecture*), iar unii autori realizează diverse comparaţii între aceste arhitecturi şi cele bazate pe servicii sau cele bazate pe activităţi [49].

Arhitecturile orientate pe resurse se dezvoltă în jurul următoarelor concepte:

- Resurse;
- Identificarea resurselor;
- Reprezentarea resurselor;
- Legături între resurse.

Principiile de la baza tehnologiei URI, *Uniform Resource Identifier*, sunt descrise de Tim Berners-Lee încă din 1996 [4]. Tehnologia URI este fundamentală pentru Web pentru că interconectează resursele Web-ului.

Arhitecturile orientate pe resurse, ROA, respectă următoarele principii:

- *Adresabilitate* – o aplicaţie bazată pe această arhitectură expune un URI pentru fiecare informaţie pe care o poate servi;
- *Lipsa memorării stării* – fiecare cerere HTTP se petrece într-o izolare completă faţă de alte cereri anterioare sau ulterioare;
- *Existenţa legăturilor* – resursele trebuie să fie conectate unele cu altele, în reprezentarea lor şi, prin urmare, un serviciu Web poate fi pus în diferite stări prin apelarea diferitelor link-uri;
- *Existenţa unei interfeţe uniforme* – accesul la orice resursă Web se poate face prin metode HTTP standardizate: GET (pentru obţinerea reprezentării unei resurse), PUT (pentru crearea unei noi resurse cu un nou URI sau pentru modificarea unei resurse existente), POST (crearea unei noi resurse la un URI existent), DELETE (ştergerea unei resurse existente).

De exemplu, interacţiunea dintre client şi server în HTTP versiunea 0.9 era simplă. Este precizată aici pentru că trebuie evidenţiate o serie de lipsuri.

Cererea clientului: `GET /hello.txt`
Răspunsul serverului: `Hello, world!`

În această versiune de HTTP, clientul cere un anumit document indicând calea absolută către acesta. Serverul returnează conţinutul fişierului cerut.

Se observă aici două caracteristici importante: *modul de adresare a unei resurse*, precum şi *lipsa informaţiilor de stare* între cereri HTTP. Versiunea HTTP 0.9 a fost dezvoltată şi completată cu noi caracteristici din care unele sunt considerate astăzi inutile sau contraproductive [45].

Web-ul s-a dezvoltat prin crearea de noi straturi/niveluri de abstractizare, iar simplitatea iniţială a protocolului HTTP a fost o condiţie necesară. Dezvoltarea ulterioară a Web-ului şi a serviciilor Web a necesitat o atenţie deosebită acordată scalabilităţii şi adaptabilităţii.

În anul 2000, Roy Fielding a publicat teza sa de doctorat *„Architectural Styles and the Design of Network-based Software Architectures*" în care a definit, nu o arhitectură, ci un set de principii formând, practic, un stil de proiectare pentru arhitecturile bazate pe Web numit REST (engl. *Reprezentational State Transfer*) [23]. Arhitecturile care respectă aceste principiile sunt descrise de atributul *RESTful*.

Acest termen poate fi asemănat cu termenul *language-oriented* din ingineria software în ceea ce priveşte rolul pe care îl are în proiectarea unei arhitecturi Web.

2.2. OPERAŢII ASUPRA RESURSELOR

HTTP este un protocol de nivel Aplicaţie (în stiva OSI). Aplicaţiile comunică între ele prin *mesaje* HTTP. Clientul pune un document într-o anvelopă (ca un „plic") şi-l trimite serverului. Acesta pune datele care formează răspunsul într-o anvelopă şi le trimite clientului. HTTP respectă standarde stricte referitoare la anvelopă, dar nu se preocupă mult de conţinut. Versiunea actuală este HTTP/1.1.

De exemplu, dacă se apelează pagina de intrare (uzual, `/index.html`) de pe un server Web, cererea arată ca în exemplul de mai jos:

```
GET /index.html HTTP/1.1
Host: www.mycompany.ro
UserAgent:  Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; ...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;...
Accept-Language:  ro
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 300
Connection: keep-alive
```

Trebuie remarcat aici că există precizate următoarele elemente:

- Metoda HTTP — GET
- Calea absolută către resursă — /index.html
- Antetele din cerere — Host, User-Agent, Accept etc.
- Corpul mesajului HTTP — Este documentul cuprins în anvelopă

În cererea HTTP din acest exemplu nu există corpul mesajului, nu există documentul care ar trebui transmis, deci „plicul" este gol.

Răspunsul serverului Web la cererea făcută poate fi următorul:

```
HTTP/1.1 200 OK
Date: Fri, 21 Sep 2007 18:46:02 GMT
Server:  Apache
```

```
Content-Type:  text/html; charset=utf-8
Content-Encoding:  gzip
Content-Length:  26640
Connection: Keep-Alive
...

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title> ajax - Căutare Google</title>
...
<head>
<body>
...
<b>Ajax</b> (programming) - Wikipedia, the free encyclopedia ...
...
</body>
</html>
```

Mesajul HTTP de răspuns are următoarele elemente:

- Codul HTTP de răspuns 200
- Anteturile din răspuns Host, User-Agent, Accept etc.
- Documentul transmis de server El începe după ultimul antet.

HTTP este un element comun tuturor tehnologiilor bazate pe Web. De aceea, atunci când se foloseşte HTTP, trebuie precizate două chestiuni importante, precizate mai jos.

Prima se referă la modul în care o aplicaţie client transmite serverului ceea ce doreşte să obţină de la acesta, dându-i toate elementele care precizează exact operaţia dorită, fără confuzie. Aceasta este *informaţia despre operaţia dorită*.

O cale de a transmite serverului operaţia dorită este să se folosească metodele HTTP. Acestea sunt denumite astfel: GET, POST, HEAD, PUT şi DELETE. Aceste metode HTTP sunt standardizate, ceea ce constituie un mare avantaj. Dar informaţia despre operaţia dorită se poate transmite şi în corpul documentului cuprins în mesajul HTTP.

A doua chestiune se referă la modul cum clientul îi poate transmite serverului cu care parte din toate datele existente să opereze. Serverul trebuie să poată distinge clar asupra căror date să efectueze operaţia indicată. Aceasta este *informaţia despre domeniul de acţiune* a operaţiei dorite. Locul unde este plasată această informaţie este important.

O variantă pentru plasarea acestei informaţii este în URI. De exemplu, dacă browserul efectuează cererea HTTP http://www.google.ro/search?q=ajax , atunci operaţia dorită este GET, iar informaţia despre domeniul de acţiune este dată de şirul: /search?q=ajax .

O altă variantă este plasarea acestei informaţii în corpul mesajului HTTP, în documentul din anvelopă. Aşa lucrează în mod obişnuit serviciile Web bazate pe SOAP. Apelul către serverul `api.google.com` este dat de exemplul următor:

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <gs:doGoogleSearch xmlns:gs="urn:GoogleSearch">
    <q>ajax</q>
...
  </gs:doGoogleSearch>
</soap:Body>
</soap:Envelope
```

În exemplul de mai sus se observă că metoda HTTP folosită este POST, iar poziţia informaţiei despre domeniul de acţiune (respectiv: *ajax*) este între marcajele pereche `<q>` şi `</q>` şi, mai departe, este cuprinsă între marcajele `<gs:doGoogleSearch>` şi, respectiv, `</gs:doGoogleSearch>`.

Se constată diferenţe mari faţa de cazul anterior în sensul că *metoda* este altfel indicată (aici prin marcajul `<gs:doGoogleSearch>`, iar anterior prin metoda HTTP GET), iar *domeniul de acţiune* este aici indicat doar prin cuvântul *ajax* plasat în document, iar anterior a fost indicat în URI prin şirul de caractere */search?q=ajax*.

Modul în care a fost proiectat serviciul Web determină modul în care se va indica serverului informaţia despre metoda apelată şi informaţia despre domeniul de acţiune al metodei apelate.

Pe baza acestor considerente se pot identifica trei *tipuri de arhitecturi cu servicii Web*:

- Arhitecturi RESTful (*Reprezentational State Transfer*) orientate pe resurse
- Arhitecturi bazate pe XML-RPC (*Remote Procedure Call*)
- Arhitecturi mixte REST-RPC.

2.3. ARHITECTURI WEB ACTUALE

2.3.1. ARHITECTURI RESTFUL

Aceste arhitecturi cu servicii Web se evidenţiază prin faptul că *informaţia despre operaţia dorită* şi *informaţia despre domeniul de acţiune* al operaţiei indicate este în URI, astfel că prima linie a unei cereri HTTP către un asemenea serviciu Web arată ca în exemplul următor:

```
GET  /reports/students  HTTP/1.1
```

Din această primă linie a cererii HTTP se înţelege ce doreşte clientul. Restul cererii cuprinde detalii. Aceste arhitecturi sunt *orientate pe resurse*.

Dacă metoda HTTP nu se potriveşte cu operaţie dorită de client atunci serviciul nu este considerat *RESTful*, deci nu respectă principiile REST. Dacă informaţia despre domeniul de acţiune a operaţiei indicate nu este în URI, atunci serviciul nu este orientat pe resurse.

Câteva exemple de binecunoscute servicii Web de tip *RESTful* şi *resource-oriented*:

- Majoritatea serviciilor Web de la Yahoo (http://developer.yahoo.com/);
- Simple Storage Service (S3) de la Amazon (http://aws.amazon.com/s3);

- Serviciile care expun Atom Publishing Protocol [1]
- Serviciul GData de la Google (http://code.google.com/apis/gdata/);
- Majoritatea serviciilor Web de tip *read-only* care nu folosesc SOAP;
- Majoritatea aplicaţiilor Web de tip *read-only*.

2.3.2. ARHITECTURI STIL RPC

S-a arătat mai sus că un mesaj HTTP este ca o anvelopă pentru datele efective care trebuie să circule între aplicaţiile aflate în „dialog”, una având rol de client care efectuează cereri, iar cealaltă rol de server care răspunde la cereri.

Un serviciu Web în stilul RPC, Remote Procedure Call [46], acceptă de la client o anvelopă plină cu date şi, similar, trimite înapoi o anvelopă cu alte date. Informaţia despre *operaţia dorită* de client, precum şi informaţia despre *domeniul de acţiune al operaţiei* respective sunt păstrate în interiorul anvelopei conţinute în mesajul HTTP. Nu contează aici tipul de anvelopă. Cazul cel mai simplu este când datele sunt împachetate direct în mesajul HTTP. Există şi varianta în care mesajul HTTP împachetează un mesaj SOAP, iar acesta, la rândul său, este ca o anvelopă pentru datele efective care trebuie transmise.

Protocolul XML-RPC pentru servicii Web este un exemplu elocvent de aplicare a arhitecturii bazate pe RPC.

Cererea HTTP a clientului, împachetând ca o anvelopă datele în format XML ale clientului, arată ca în exemplul de mai jos:

```
POST  /rapoarte/rezultate  HTTP/1.1
Host: students.upt.ro
UserAgent:  Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; ...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;...
Accept-Language:  ro
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 300
Connection: keep-alive

<?xml version=”1.0” ?>
<methodCall>
  <methodName>RezultateStudent</methodName>
  <params>
    <param><value><sting>CAL5537</string></value></param>
  </params>
</methodCall>
```

Trebuie remarcat că anvelopa HTTP rămâne mereu la fel, dar documentul XML se schimbă, deoarece depinde de procedura apelată (în exemplul de mai sus, *RezultateStudent*) şi de parametrii care trebuie transmişi procedurii (de exemplu, numărul matricol al studentului). Nu contează ce procedură apelează clientul, metoda HTTP este mereu POST.

Se constată că un serviciu bazat pe XML-RPC nu foloseşte multe din facilităţile HTTP. Un serviciu Web bazat pe RPC expune un URI pentru fiecare procesor al documentelor, respectiv o entitate software care ştie să deschidă anvelopa din mesajul HTTP şi să convertească datele conţinute în comenzi software.

Pentru comparaţie, un serviciu RESTful, orientat pe resurse, care să facă acelaşi lucru ca cel de mai sus, ar trebui să primească o cerere HTTP GET astfel:

```
GET  /rapoarte/rezultate/student/CAL5537  HTTP/1.1
Host: students.upt.ro
UserAgent:  Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; ...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;...
Accept-Language:  ro
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 300
Connection: keep-alive
```

Operaţia dorită şi domeniul ei de acţiune sunt date în URI. Anvelopa HTTP este goală. Cererea HTTP GET de mai sus nu conţine un corp al mesajului, ci doar URI şi anteturi .

Exemple de servicii Web în stil RPC:

- Toate serviciile care folosesc XML-RPC;
- Marea majoritate a serviciilor bazate pe SOAP;
- Unele aplicaţii Web (considerate slab proiectate).

2.3.3. ARHITECTURI MIXTE RESTFUL-RPC

Acest termen descrie serviciile Web care se pot încadra între cele RESTful şi cele în stil RPC [45]. Un asemenea serviciu se apelează prin URI de forma:

```
http://students.upt.ro/services/absolventi?method=search&nume=pop
```

Mesajul HTTP al unui client Web care a cerut resursa indicată prin URI de mai sus poate arăta astfel:

```
GET  services/absolventi?method=search&nume=pop  HTTP/1.1
Host: students.upt.ro
UserAgent: ....
Accept: text/xml,application/xml,text/html ....
Accept-Language:  ro
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 300
Connection: keep-alive
....
```

De la început se constată că *informaţia despre operaţia dorită* este în URI, respectiv: *method=search*. La fel, *informaţia despre domeniul de acţiune al operaţiei* este în URI, respectiv *nume=pop*. S-ar putea concluziona că este vorba de un serviciu Web ce respectă arhitectura *RESTful*, orientată pe resurse.

Totuşi, cererea clientului nu foloseşte o metodă HTTP pentru a indica operaţia dorită (GET, PUT, DELETE etc.), aşa cum se procedează în arhitecturile RESTful. Practic, această cerere foloseşte mesajul HTTP ca o anvelopă în care se indică, într-un alt mod decât cel indicat de regulile REST, operaţia dorită. Aceasta înseamnă că arhitectura este în stil RPC.

Constatăm că varianta analizată mai sus este între cazul clar al arhitecturilor RESTful şi cel al arhitecturilor stil RPC. Este cazul *arhitecturilor mixte RESTful-RPC*, denumite şi *arhitecturi hibride* [45].

Mulţi programatori Web proiectează serviciile Web exact aşa cum proiectează aplicaţiile Web şi, astfel, ajung la aceste arhitecturi mixte.

2.4. SERVICII WEB

În cadrul consorţiului W3C (http://www.w3.org), grupul de lucru pentru arhitectura serviciilor Web (*W3C Web Services Architecture Working Group*) a elaborat în 2004 un document care a avut un impact extrem de puternic asupra dezvoltării de aplicaţii bazate pe servicii Web numit *Web Services Architecture* [5].

Conform acestui document, un serviciu Web este un sistem software destinat să sprijine interoperabilitatea între maşini care interacţionează pe o reţea, iar interfaţa sa este descrisă într-un format specific, destinat să fie procesat de un sistem automat (nu de utilizatorul uman). Un alt sistem software, numit uzual *client*, interacţionează cu serviciul Web într-o manieră prestabilită folosind *mesaje* care respectă protocolul SOAP [51], iar transportul efectiv pe reţea este realizat conform protocolului HTTP [32].

Un serviciu Web oferă anumite funcţionalităţi clienţilor săi, în sensul că aceştia pot solicita numite *operaţii specifice* serviciului respectiv. Clienţii determină operaţiile posibile ale unui anumit serviciu din descrierea interfeţei sale în limbaj WSDL [68].

Pentru a putea prezenta ulterior arhitecturi cu servicii Web trebuie stabilită o simbolizare pentru unitatea software numită *serviciu Web* (Fig. 2.1).

Fig. 2.1. Simbolizarea unui serviciu Web

Serviciile Web sunt din ce în ce mai mult folosite în sisteme distribuite, în aplicaţii şi servicii de importanţă majoră. Pentru sistemele distribuite, arhitectura orientată pe servicii, *SOA - Service Oriented Architecture*, care foloseşte serviciile Web, este acceptată deja ca fiind arhitectura capabilă să interconecteze aplicaţii care rulează pe diferite sisteme de operare şi facilitează interacţiuni complexe între sisteme autonome şi eterogene, fie în cadrul organizaţiilor, fie între organizaţii aflate în relaţii de tip B2B (business-to-business).

Serviciile Web permit aplicaţiilor software aflate în diferite organizaţii (companii) să interacţioneze unele cu altele, chiar dacă organizaţiile respective folosesc sisteme hardware diferite, sau sisteme de operare şi chiar limbaje de programare diferite. Serviciile Web sunt capabile să uniformizeze şi să eficientizeze activităţile de business pe Internet prin invocarea automată a unor operaţii care, altfel, ar trebui invocate manual de un operator uman. Deci, serviciile Web permit interacţiunile directe între calculatoare situate în organizaţii diferite.

Beneficiile aduse de serviciile Web sunt semnificative deoarece ele facilitează automatizarea unor activităţi în cadrul unor afaceri distribuite pe Internet între mai multe companii, precum şi colaborarea între organizaţii prin interconectarea proceselor care rulează în sistemele informatice ale acestora.

Standardele pentru servicii Web definesc sintaxa documentelor furnizate de

aceste servicii, precum şi formatul mesajelor şi metodele de a descrie şi de a găsi aceste servicii. Aceste standarde nu definesc mecanismele de implementare şi nici interfeţele de programare a aplicaţiilor, care rămân proprietatea companiilor furnizoare [5].

Diferiţi furnizori pot să implementeze în mod diferit infrastructurile de servicii Web. Astfel, standardele pentru servicii Web oferă interoperabilitate între servicii Web care sunt implementate pe diferite platforme hardware, sisteme de operare şi în diferite limbaje de programare, dar ele nu tratează problema portabilităţii programelor de aplicaţii de pe o platformă pe alta (hardware şi software).

Standardele de bază pentru servicii Web cuprind:

- *Limbajul XML (eXtensible Markup Language)*, care defineşte sintaxa documentelor furnizate de serviciile Web, astfel încât informaţiile din respectivele documente să fie auto-descriptive [69].
- *Protocolul SOAP (Simple Object Access Protocol)* pentru schimbul de mesaje XML şi maparea tipurilor de date astfel încât aplicaţiile să poată comunica între ele [51].
- *Limbajul WSDL (Web Services Description Language)* pentru descrierea unui serviciu Web, a numelui său, a operaţiilor care pot fi apelate, a parametrilor necesari operaţiilor, precum şi a adresei unde trebuie trimise cererile [68].
- *Standardul UDDI (Universal Description Discovery and Integration)* care precizează cum trebuie să procedeze furnizorii de servicii pentru a face publice serviciile oferite şi, de asemenea, cum trebuie să procedeze clienţii pentru a descoperi serviciile existente, pentru a le folosi [64].

2.5. O SCURTĂ TAXONOMIE A SERVICIILOR ŞI STĂRILOR

În ceea ce priveşte starea unui serviciu şi modul în care este ea stocată, se disting următoarele tipuri de servicii:

1) *Serviciul stateless* implementează schimbul de mesaje fără a folosi informaţii care nu sunt conţinute în mesaje. Un asemenea serviciu nu menţine informaţii de stare de la un mesaj la altul.
2) *Serviciul stateful* menţine informaţii de stare de la un mesaj la altul. Acţiunile unui asemenea serviciu depind de conţinutul mesajului-cerere, dar şi de starea serviciului în acel moment. Locul unde sunt depozitate informaţiile de stare determină două tipuri de servicii stateful:
    a. *servicii care încapsulează starea*, uzual în obiecte proprii;
    b. *servicii care folosesc resurse stateful*, respectiv, servicii care îşi păstrează informaţiile de stare în depozite externe de date persistente (de exemplu, baze de date sau fişiere). Un asemenea serviciu accesează şi manipulează un set de resurse situate fizic într-un sistem de stocare a datelor. Aceste resurse memorează starea curentă, pe baza mesajelor pe care serviciul le primeşte şi le trimite.

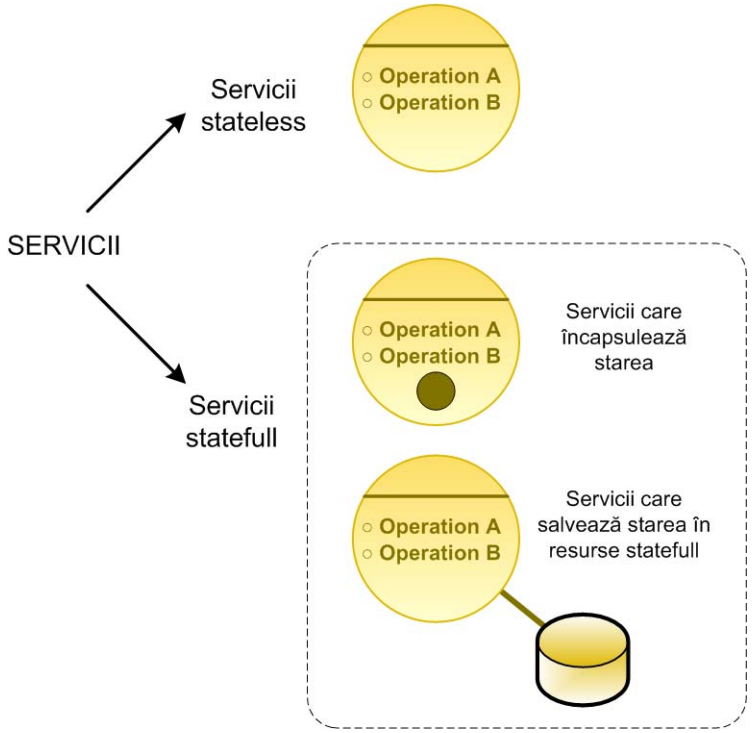

Fig. 2.2. Clasificarea serviciilor după modul de păstrare a stării

2.6. IMPLEMENTĂRI STATELESS, INTERFEŢE STATEFUL

Implementarea unui serviciu care foloseşte resurse stateful foloseşte obiecte dinamice care stochează valori care descriu starea curentă a serviciului, obiecte necesare în schimburile de mesaje dintre serviciul respectiv şi serviciile apelante. Un asemenea serviciu poate fi considerat *stateless*, dacă el deleagă responsabilitatea managementului obiectelor de stare unei alte componente ca, de exemplu, o bază de date.

Evitarea folosirii obiectelor de stocare a stării curente a serviciului determină creşterea fiabilităţii şi scalabilităţii. Un serviciu Web *stateless* poate fi repornit după o cădere fără a avea grija istoricului interacţiunilor anterioare. În plus, se pot crea noi copii ale unui serviciu Web *stateless* ca urmare a modificării nivelului de încărcare a sistemului. Astfel, nestocarea stării curente este considerată o bună practică inginerească în implementarea serviciilor Web.

O consecinţă a proprietăţii *statelessness* este că orice stare dinamică

necesară pentru execuţia unui anumit mesaj trebuie să îndeplinească următoarele condiţii:

- starea trebuie să fie precizată explicit în interiorul mesajului cerere fie direct, prin precizarea unor valori, fie indirect, prin referinţă.
- starea trebuie să fie menţinută şi actualizată în cadrul altor componente ale sistemului cu care serviciul Web poate interacţiona.

Pe de altă parte, implementarea unui serviciu Web poate să fie controlată de parametri de stare statici, respectiv referinţe pre-configurate către alte componente ale sistemului.

Modelul de serviciu care foloseşte resurse stateful permite ca o implementare *stateless* a unui serviciu Web să interacţioneze frecvent şi să actualizeze o stare dinamică menţinută în alte componente ale sistemului ca, de exemplu, o bază de date. În asemenea cazuri, identitatea elementelor de stare poate să fie transferată în mesajul cerere sau să fie menţinută sub forma unor date statice de către serviciul Web. Interfaţa oferită de un asemenea serviciu este, în mod clar, *stateful*, deci comportamentul serviciului este influenţat de starea respectivă.

2.7. STRUCTURI WS-RESOURCE ŞI PROPRIETĂŢILE ACID

O abordare în sensul modelării resurselor stateful într-un framework cu servicii Web este prezentată în [26]. Denumirea acestei noi unităţi software este: *WS-Resource*. Această structură oferă clientului atât *date* cât şi *operaţii* specifice.

O structură *WS-Resource* are câteva elemente definitorii:

- este o structură compusă dintr-un serviciu Web *stateless* (nu păstrează date despre stare) şi o resursă *stateful* (asigură stocarea datelor persistente);
- serviciul Web oferă clienţilor *operaţii*;
- resursa oferă date necesare pentru execuţia operaţiilor solicitate în mesaje;
- starea structurii WS-Resource poate fi interogată şi actualizată prin schimburile de mesaje la nivelul serviciului Web.

Acronimul ACID se referă la cele patru proprietăţi importante care trebuie respectate de resursele stateful din cadrul structurilor WS-Resource în contextul utilizării tranzacţiilor în sistemele tranzacţionale [26].

- *Atomicitatea***.** Actualizările unei resurse stateful în contextul folosirii tranzacţiilor trebuie făcute în maniera „totul sau nimic”.
- *Consistenţa***.** În urma executării tranzacţiilor şi chiar în urma eşecului acestora resursa trebuie să rămână într-o stare consistentă.
- *Izolarea***.** Actualizările parţiale ale unei resurse stateful în cadrul unei tranzacţii formate din mai multe operaţii nu trebuie să fie vizibile în afara tranzacţiei până la încheierea ei (controlul accesului concurent).
- *Durabilitatea***.** Actualizările unei resurse stateful trebuie să rămână permanente după încheierea tranzacţiei.

2.8. APELAREA RESURSELOR ALGORITMICE

O serie de preocupări actuale sunt orientate spre aplicaţii adaptive în care se pune accentul pe reutilizarea resurselor existente în definirea noilor aplicaţii [7].

În domeniul aplicaţiilor Web este foarte important modul cum se face adresarea resurselor, având în vedere tendinţele din ultimii ani de a folosi în principal URI-uri non-RESTful, bazate doar pe metodele HTTP GET şi POST [45].

Reanalizarea tendinţelor din ultimii ani în domeniul Web-ului programabil comparativ cu principiile arhitecturale iniţiale ale Web-ului a condus la arhitecturi moderne ale aplicaţiilor distribuite, orientate pe resurse şi pe reutilizarea acestora [9].

Proiectarea resurselor pentru o arhitectură orientată pe resurse presupune o serie de operaţii importante:

- identificarea tipurilor de resurse şi a rolului acestora în aplicaţie;
- crearea unor denumiri specifice pentru resurse;
- alegerea tipului de acces (public/privat) la resurse;
- identificarea operaţiilor dorite asupra acestor resurse;
- crearea / identificarea ierarhiei resurselor;
- identificarea resurselor algoritmice şi a metodelor acestora;
- identificarea colecţiilor de resurse, dacă există.

Resursele oferite de serviciile Web ale unei aplicaţii complexe sunt de mai multe tipuri, iar aceste tipuri sunt strâns legate de scopurile aplicaţiei respective. Mai mult, o problemă care trebuie rezolvată cu atenţie este problema tipului de acces la resurse, respectiv, *acces public sau privat*, acordat în urma unei autentificări a utilizatorului.

Resursele statice se apelează totdeauna la fel, dar *resursele algoritmice* se apelează prin indicarea metodei dorite şi a parametrilor care trebuie transmişi metodei respective. O resursă publică, de exemplu, pagina Web cu lista specializărilor unei facultăţi, se apelează simplu folosind metoda HTTP GET.

Utilizatorul nu ştie (şi nu este interesat) cum s-a format pagina Web din fereastra browserului său: fie prin citirea unui fişier static de pe server, fie în mod dinamic, prin rularea unei aplicaţii pe partea serverului, cu sau fără accesarea unor baze de date. Din punctul de vedere al ierarhiei orientate pe resurse contează doar faptul că informaţia respectivă este *read-only* pentru client, iar această citire se realizează cu metoda GET.

O mare atenţie trebuie acordată pentru *ierarhia resurselor*. Un model avantajos de a descrie această ierarhie este asemănător cu cel din programarea *object-oriented*, în care apar legături de tipul *ascendenţi – descendenţi*. Într-un identificator de resurse, elementul care arată trecerea spre un alt nivel al ierarhiei este separatorul de cale uzual pentru URL, caracterul „/”.

Resursele algoritmice se referă la resursele obţinute în urma aplicării unui algoritm asupra unui set de date. Algoritmul respectiv, de cele mai multe ori, trebuie să primească o serie de parametri. Aceşti parametri sunt pe acelaşi nivel ierarhic, deci ei nu trebuie separaţi de caracterul „/”, ci de un alt caracter care sugerează menţinerea nivelului ierarhic. Uzual, este caracterul ”,”.

Având convenite aceste elemente, o resursă algoritmică poate fi privită ca o metodă specifică a unui obiect, metodă care se apelează cu parametri. Astfel, trebuie indicate toate aceste elemente după regula:

```
obiect/metoda/valoare_par1,valoare_par2,valoare_par3...
```

Identificatorul de mai sus poate fi înţeles mai uşor dacă este scris astfel:

```
obiect/metoda?par1=valoare1, par2=valoare2, par3=valoare3...
```

Acest mod de apelare a unei metode cu parametri se va baza pe calea absolută spre resursa respectivă şi pe porţiunea *QueryString* dintr-un URI.

Câteva probleme ar putea ridica modul de adresare a colecţiilor de resurse şi modul de indicare a unui element dintr-o colecţie de resurse. Se foloseşte separatorul „/”. Se indică mai întâi numele colecţiei, urmează separatorul „/”, apoi se indică numele elementului căutat. Astfel, în adresa elementului respectiv apare:

```
colecţie/{element-colecţie}...
```

Regula rămâne valabilă şi dacă elementul primei colecţii este, la rândul său, tot o colecţie, din care trebuie indicat un element. În acest caz, în URI ierarhia apare sub forma:

```
colecţie_ascendent/colecţie_descendent/{element-colecţie}...
```

În continuare, elementul apelat poate fi privit ca un obiect (în sensul dat de programarea orientată pe obiecte). Acest obiect are metode proprii. Apelarea acestora de către o aplicaţie client poate determina execuţia unei operaţii din setul *CRUD (Create, Read, Update, Delete)* asupra obiectului respectiv. Uneori aceste operaţii se aplică unui obiect de tip *set de date*.

Pentru aplicaţia client denumirea metodelor unui obiect trebuie să fie sugestivă. De aceea, se vor folosi o altă terminologie, specifică aplicaţiei, mai sugestivă decât *Create, Read, Update, Delete*. De exemplu: *Adaugă, Înmatriculează, Afişează, Modifică, Şterge* etc.

Pe baza considerentelor de mai sus, se constată că există *identificatori ai unui obiect* şi *identificatori ai unor acţiuni*. Vom folosi, uzual, termenii, *URL-ul unui obiect* şi *URL-ul unei acţiuni* (*URL – Uniform Resource Locator*).

Regulile de adresare a resurselor, indicate mai sus, vor putea fi folosite în scopul organizării resurselor unei aplicaţii, astfel încât acestea să fie apelabile prin URL-uri conform principiilor REST.

Se constată că există o serie de elemente care sunt subordonate unul altuia, astfel: elementul *student* (membru al unei colecţii de studenţi) este subordonat elementului *specializare* (membru al unei colecţii de specializări) care este subordonat, la rândul său, elementului *facultate* (membru al unei colecţii de facultăţi).

Un URI poate fi explicit în precizarea subordonării elementelor, ca de exemplu:

```
facultate/specializare/an_studiu/nume_student
```

desemnează colecţia de studenţi cu numele indicat (`nume_student`), de la specializarea indicată (ex. `calculatoare`), anul de studiu indicat (ex. `4`), din cadrul facultăţii indicate (ex. `inginerie`), indiferent de celelalte atribute ale studenţilor, de exemplu: prenume, anul înmatriculării, bursier/nebursier, integralist/neintegralist etc.

Pe de altă parte, un URI de forma:

```
facultate/specializare/nr_matricol
```

desemnează, în mod unic, studentul cu numărul matricol indicat (care este unic) de la specializarea indicată, din cadrul facultăţii indicate.

Având clarificată calea către resursa dorită (o colecţie de studenţi sau un anumit student), în URI se indică în continuare metoda care trebuie aplicată resursei. Aici terminologia folosită este specifică activităţii de învăţământ. Operaţiile dorite pot fi indicate prin termeni ca de exemplu: *promovează, acordă_bursă, înscrie_licenţă, afişează_lista,* etc.

De exemplu, în cazul unei aplicaţii pentru managementul activităţii studenţilor şi raportare, utilizatorul *student* ar putea fi interesat să obţină accesul la următoarele resurse:

- lista disciplinelor din anul de studiu indicat, cu numărul de credite;
- rezultatele obţinute într-un anumit an universitar, etc.

În cazul când utilizatorul indică, o *colecţie de resurse*, aplicaţia trebuie să îl direcţioneze către o resursă cu rol de *catalog al colecţiei de resurse*, sau index al colecţiei. Acolo, utilizatorul trebuie să vadă URL-urile specifice către resursele conţinute în colecţie.

În cazul când utilizatorul indică, *un element dintr-o colecţie de resurse*, aplicaţia trebuie să îl direcţioneze către resursa respectivă, indicată într-un mod inconfundabil.

Exemplele de resurse de mai sus, care pot fi accesate de utilizatorul student sunt *resurse algoritmice*. Uzual, acestea sunt pagini Web cu conţinut dinamic care se formează în urma interogării unor resurse de tip baze de date sau fişiere statice aflate pe un nivel ierarhic inferior.

Utilizatorul uman accesează resurse aflate pe un nivel superior, iar aceste resurse sunt reprezentate într-un anumit format. Resursele de pe nivelul inferior sunt accesate de module software aflate pe un nivel ierarhic superior, iar fişierele conţinute în mesajele schimbate între aceste module pot fi în diferite limbaje (de exemplu, HTML sau XML) şi sunt destinate prelucrării de către entităţi software şi nu trebuie să fie neapărat lizibile utilizatorului uman.

O aplicaţie complexă de management al activităţii studenţilor poate opera cu mai multe colecţii de resurse, de exemplu: *cursuri, studenti*, *rezultate* etc. Unele colecţii sunt elemente ale altor colecţii, de exemplu colecţia *facultăţi* conţine sub-colecţia *specializări* care, la rândul său conţine sub-colecţia *cursuri*. Analog, se pot defini şi alte ierarhii ale resurselor, de tipul *colecţie/colecţie/colecţie/element*.

Toate aceste considerente de mai sus au influenţă în definirea şi clarificarea în capitolul următor a unei arhitecturi stratificate pentru aplicaţii Web.

2.9. ARHITECTURI RESTFUL STRATIFICATE

O aplicaţie modernă are o arhitectură care, în ceea ce priveşte resursele, ţine seama de principiile ROA, *Resource Oriented Architecture* [45]. Fiecare din aceste principii se referă la anumite aspecte organizatorice şi funcţionale ale aplicaţiei. Propun aici pentru analiză o arhitectură modulară şi stratificată, conform Fig. 2.3.

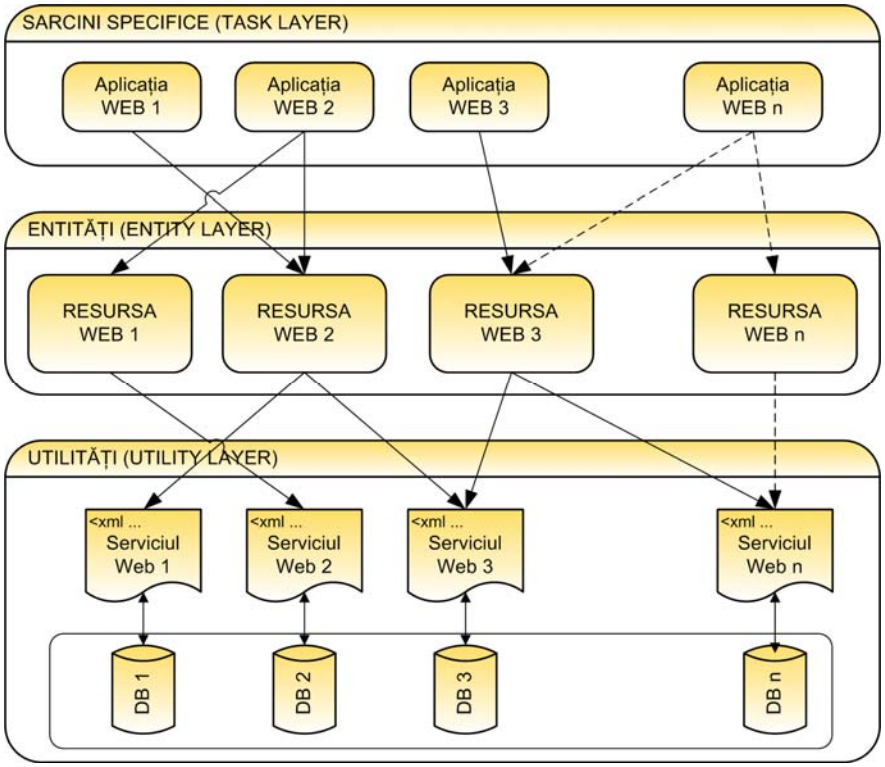

Fig. 2.3. O arhitectură posibilă a unei aplicaţii Web complexe

Nivelul superior al arhitecturii este *Task Layer*. El cuprinde o serie de aplicaţii Web apelabile de entităţi software externe, uzual aplicaţii tip browser Web. O aplicaţie de pe acest nivel este proiectată să execute o anumită sarcină (*task*).

Pentru îndeplinirea unei sarcini, aplicaţia Web apelată execută operaţiunile:

- analizează cererea clientului extern de acces la resurse de pe un nivel inferior;
- construieşte identificatorul resursei căutate conform ierarhiei stabilite (practic, construieşte un URI conform principiilor REST);
- efectuează cererea către nivelul ierarhic inferior, respectiv către dispecerul de resurse, de acces la resursa cu URI-ul stabilit anterior;
- acceptă redirecţionarea făcută de dispecer către o anumită resursă entitate de pe nivelul Entity Layer;
- transmite unei anumite entităţi să efectueze o anumită operaţie (citire, scriere, ştergere) asupra unui set de date, folosind pentru aceasta un URI în formatul specific entităţii respective, uzual un format stil RPC;
- preia de la entitatea solicitată o reprezentare a datelor cuprinse într-un mesaj de răspuns, sub forma unui fişier în format XML sau HTML;
- formează un răspuns în format HTML, conform unui şablon al aplicaţiei respective, şi îl returnează aplicaţiei client.

Aplicaţiile Web de pe nivelul superior sunt orientate pe sarcini specifice. Potenţialul de reutilizare pentru alte sarcini este relativ scăzut. În funcţie de cererea clientului, aplicaţiile Web de pe nivelul Task Layer accesează resurse diferite aflate pe nivelul Entity Layer.

Subordonat nivelului *Task Layer* este nivelul *Entity Layer*. Acesta cuprinde, în principal, entităţi specifice aplicaţiei respective care, practic, constituie resurse Web pentru nivelul superior.

O resursă Web furnizează, la cerere, date în format HTML sau XML. Trebuie precizat aici că, în această arhitectură, pornim de la ipoteza că resursele Web de pe acest nivel ştiu să interpreteze şi să răspundă la cereri formulate în stil RPC. Este cazul obişnuit al aplicaţiilor existente deja în cadrul organizaţiilor, aplicaţii dezvoltate de-a lungul timpului folosind diverse tehnologii.

O resursă Web de pe nivelul al doilea trebuie, de multe ori, să obţină seturi de date. Ea se poate adresa direct unui sistem de gestiune a bazelor de date (SGBD), dacă are suport software pentru aceasta. Datorită diversităţii de sisteme de gestiune a bazelor de date, accesul la bazele de date este dependent de platformă. Aplicaţiile moderne folosesc servicii Web cu scopul de a avea un acces unitar la datele din bazele de date, indiferent de platforma pe care rulează SGBD-ul şi de a obţine datele dorite într-un format înţeles de toate aplicaţiile Web. În mod obişnuit, serviciile Web oferă seturi de date în format XML.

Resursele Web de pe stratul *Entity Layer* solicită seturi de date serviciilor Web de pe nivelul ierarhic inferior numit *Utility Layer*, deoarece rolul acestui nivel este de a furniza diferite utilităţi nivelului superior.

Într-o aplicaţie cu asemenea arhitectură dezvoltată în timp folosind diverse tehnologii este foarte probabil ca resursele Web de pe nivelul *Entity Layer* să fie programate cu mai mulţi ani în urmă astfel încât cererile care trebuie adresate acestora să fie în stil RPC, nu REST. De aceea, aplicaţiile Web de pe nivelul ierarhic superior trebuie să folosească URI-uri care nu respectă principiile REST.

Problema care trebuie rezolvată aici este următoarea:

Cum trebuie modificată arhitectura aplicaţiei astfel încât aplicaţiile Web (aplicaţiile-client) să apeleze resursele folosind numai URI-uri stil REST, chiar dacă resursele existente anterior vor fi apelate, în continuare, în stil RPC, în interiorul sistemului ?

Soluţia pe care o prezentăm aici constă în crearea unei modalităţi de transformare a identificatorilor de resurse stil RPC în identificatori stil REST, respectând şi ierarhia existentă a resurselor organizaţiei.

Practic, vom considera un strat suplimentar de adaptare, plasat între Task Layer şi Entity Layer. Se formează o nouă arhitectură care respectă principiile REST şi este organizată pe mai multe niveluri. Consider că denumirea potrivită este: *Arhitectură RESTful Multilayer* (Fig. 2.4).

Scopul arhitecturii RESTful Multilayer Architecture este să adapteze la principiile REST aplicaţiile Web şi resursele existente deja în cadrul unei organizaţii, fără să fie necesară reprogramarea aplicaţiilor.

Această adaptare a cererilor formulate în stil REST, pe partea aplicaţiilor Web de pe nivelul superior *Task Layer*, cu cererile în stil RPC pe care le aşteaptă resursele Web existente pe nivelul *Entity Layer* se produce în cadrul modulului *Resource Dispatcher*. Din punct de vedere arhitectural, Resource Dispatcher constituie un nivel suplimentar al cărui nume, sugestiv, este *URI Adapting Layer*.

Pe nivelul URI Adapting Layer, în cadrul modulului Resource Dispatcher, se petrec următoarele operaţiuni:

- este recepţionată o cerere în stil REST către o resursă de pe un nivel inferior;
- URI-ul resursei solicitate este analizat pentru a se determina entităţile menţionate (colecţii, elemente din colecţii, metode);
- se construieşte o nouă cerere, dar în stilul RPC, potrivită cu modul de lucru existent deja al resursei respective;
- apelantul (*resource caller*) este redirecţionat folosind noul URI.

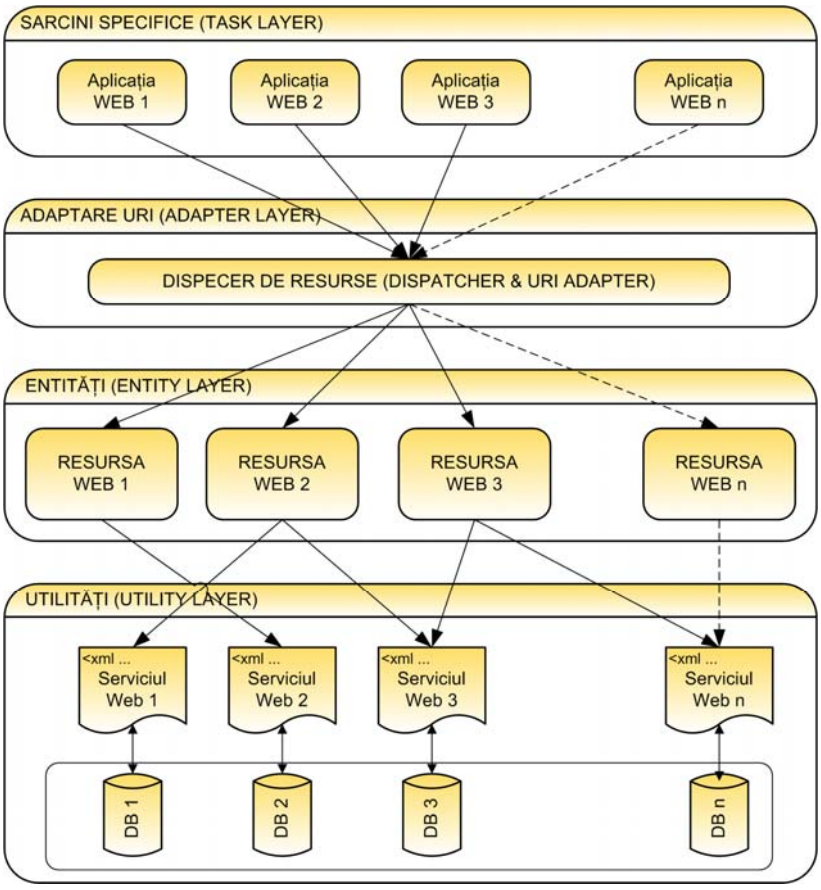

Fig. 2.4. Arhitectura RESTful Multilayer

Astfel, entitatea apelantă ajunge la resursa solicitată fără să cunoască faptul că, pe parcurs, URI-ul menţionat în cerere a fost transformat şi a existat o redirecţionare. Entitatea software apelantă consideră că mesajul HTTP primit este răspunsul firesc al cererii în stil REST pe care a făcut-o.

Operaţiile desfăşurate de *Resource Dispatcher* ţin cont de ierarhia logică a resurselor aplicaţiei distribuite, ceea ce este un mare avantaj. Resurse care trebuiau apelate, până în acest moment, în diferite stiluri, RPC, REST sau mixt RPC-REST, acum se pot apela după un principiu unitar de identificare.

Existenţa nivelului *URI Adapting Layer* permite punerea în practică a celor două principii ale ROA, *Resource Oriented Architecture,* respectiv [45]:

- principiul adresabilităţii resurselor;
- principiul interfeţei uniforme pentru accesul la resurse.

2.10. ADAPTAREA IDENTIFICATORILOR RESURSELOR

Problema adaptării identificatorilor de resurse se pune la interfaţa dintre *Task Layer* şi *Entity Layer*. Resursele de pe nivelul *Entity Layer* acceptă apeluri cu parametri, dar uzual aceştia sunt în porţiunea *QueryString* din URI. Se doreşte ca apelurile făcute de la nivelul superior, *Task Layer* să respecte principiile REST, de aceea modul de apelare a resurselor între aceste niveluri trebuie adaptat.

La nivelul *Task Layer* dezvoltatorii aplicaţiilor Web operează cu o ierarhie bine stabilită a resurselor aflate pe nivelul inferior. Avantajele introducerii nivelului *URI Adapting Layer*, reprezentat de modulul software *Resource Dispatcher*, sunt:

- devine posibilă constituirea unei ierarhii virtuale, logice, bine organizate, a resurselor întregii aplicaţii distribuite, indiferent de existenţa unor resurse şi utilităţi moştenite de la sistemele mai vechi, existente deja în reţeaua companiei;
- apare posibilitatea modificării după noi criterii a ierarhiei virtuale a resurselor aplicaţiei distribuite fără ca să fie necesară modificarea identificatorilor resurselor reale, aflate pe nivelurile inferioare;
- devine posibilă separarea dezvoltării aplicaţiilor de pe nivelul superior, *Task Layer*, de dezvoltarea resurselor şi utilităţilor de pe nivelurile inferioare, permiţând astfel echipe separate de programatori;
- devine posibilă implementarea principiilor REST la identificatorii de resurse folosiţi la nivelul superior, cu avantajele ce decurg de aici: obţinerea unui rang mai bun pentru resursele publice în bazele de date ale motoarelor de căutare, o mai mare uşurinţă pentru utilizatori în înţelegerea ierarhiei resurselor etc.

Adaptarea identificatorilor la nivelul modulului *Resource Dispatcher* este, practic, o mapare a URL-urilor RESTful cu care se lucrează la nivelul *Task Layer* la URI-urilor reale ale resurselor aflate pe nivelurile inferioare. Această mapare trebuie să fie modificabilă. Pentru aplicaţia ASP.NET această mapare conduce la modificarea URL-ului, deci poate fi considerată o rescriere a URL-ului. În lucrările de specialitate se întâlnesc ambii termeni: *URL Rewriting* [31] şi *URL Mapping* [40]. Termenul *rewriting* sugerează o analogie cu modulul *mod_rewrite* existent la serverul Web Apache şi lipsind la serverul Web IIS până la versiunea 6.

Adaptarea identificatorilor resurselor se poate face în mai multe moduri, fiecare bazat pe o altă metodă de modificare a URL-urilor.

O scurtă clasificare este următoarea:

- metode bazate pe rescrierea URL-urilor la nivelul serverului IIS [34], [35], [33];
- metode bazate pe folosirea metodei *Response.Redirect* în aplicaţia ASP.NET [42], [39];
- metode bazate pe folosirea metodei *Context.RewritePath* la nivelul aplicaţiei ASP.NET [31], [66].

Pe platforma ASP.NET se poate folosi metoda *RewritePath()* a obiectului *Context*, din clasa *HttpContext*. Această metodă permite programatorului să modifice în mod dinamic calea scrisă într-un URL care trebuie modificat şi, ca urmare, mediul ASP.NET va continua execuţia folosind URL-ul modificat. Această variantă de implementare este prezentată în [60].

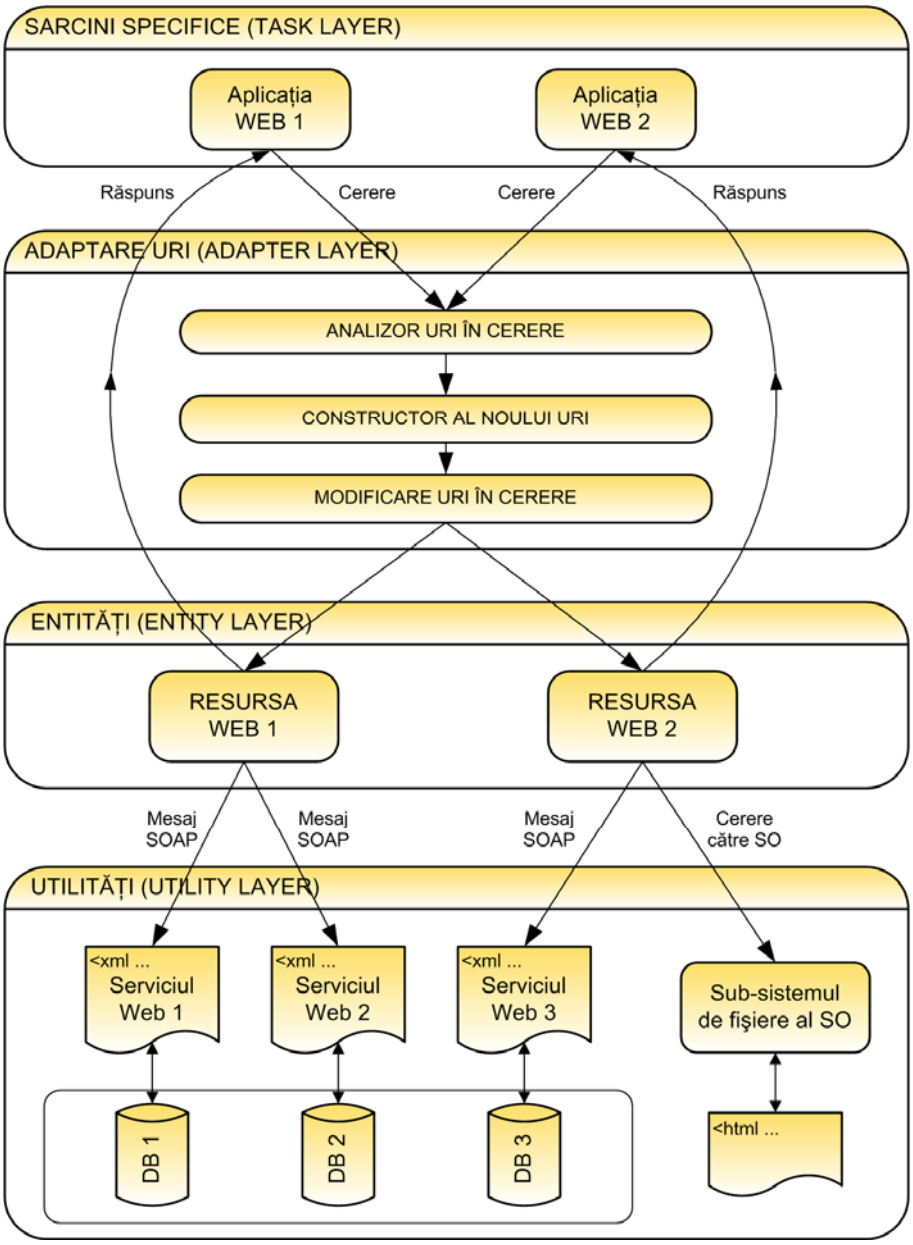

Fig. 2.5. Adaptarea identificatorilor resurselor

Modul de adaptare a URL-urilor propus aici serveşte unui obiectiv stabilit de la început şi anume *implementarea unei arhitecturi RESTful*, orientată pe resurse, în care să se folosească identificatori de resurse conform principiilor REST. Metodele existente deja, care folosesc metoda *RewritePath*, au ca scop optimizarea URL-urilor pentru motoarele de căutare şi se referă, în principal, la metoda HTTP GET.

În principiu, această operaţie de modificare trebuie efectuată în timpul metodei asociate evenimentului *BeginRequest*. Aceasta deoarece modificarea URL-urilor trebuie să fie făcută înainte de rularea *modulelor* HTTP.

La nivelul metodei care tratează evenimentul *BeginRequest* propun efectuarea următoarele operaţiuni (Fig. 2.5):

- analizarea URL-ului RESTful de intrare, care a declanşat crearea paginii ASP.NET pentru identificarea următoarelor elemente: metoda HTTP, calea către resursa solicitată prin ierarhia logică a resurselor (de forma: colecţie/colecţie/element);
- construirea unui nou URL (non RESTful) către resursa solicitată din cadrul resurselor Web existente deja în aplicaţia distribuită;
- rescrierea căii către o resursă reală prin folosirea metodei *RewritePath* a obiectului *Context*, care va determina obţinerea conţinutului paginii Web de la o resursă reală, de pe stratul *Entity Layer*.

Astfel, aplicaţia care a solicitat o resursă folosind un identificator virtual, care respectă principiile REST, primeşte ca răspuns conţinutul unei resurse reale, chiar dacă pe parcurs a fost efectuată o adaptare între URL-uri RESTful şi URL-uri non-RESTful.

Aplicaţiile ASP.NET pot conţine un fişier opţional numit *Global.asax*, în care se poate scrie cod pentru a particulariza evenimentele de nivel aplicaţie sau sesiune create de ASP.NET sau de modulele HTTP.

O variantă de implementare pentru aplicaţii ASP.NET a acestei tehnici de adaptare a resurselor a fost prezentată în [60].

Arhitecturile RESTful oferă acces la resurse folosind un anumit mod de scriere a identificatorilor acestor resurse. Pentru a putea accesa aceste resurse atât pentru citire, dar şi pentru scriere, aplicaţiile client trebuie să fie capabile să trimită mesaje HTTP care folosesc toate cele 4 metode: GET, POST (uzuale) dar şi PUT, DELETE. Cu aceste metode HTTP aplicaţiile-client pot efectua corect operaţiile *CRUD* (*Create, Read, Update, Delete*).

2.11. CONCLUZII ŞI CONTRIBUŢII

World Wide Web a cunoscut în timp o evoluţie importantă şi impactul asupra modului de realizare a aplicaţiilor distribuite a fost semnificativ încă de la apariţia acestuia. Deoarece la nivelul protocolului HTTP se pot transmite se pot transmite diferite tipuri de date, Web-ul este considerat un mediu puternic pentru susţinerea programării distribuite. El este de mulţi ani în atenţia dezvoltatorilor de aplicaţii şi specialiştii apreciază că sunt aşteptate noi evoluţii în dezvoltarea aplicaţiilor distribuite care folosesc Web-ul ca mediu de transport.

În acest capitol se prezintă conceptele de *resurse*, *identificarea resurselor*, *reprezentarea resurselor*, şi, respectiv, *legături între resurse*, care sunt concepte esenţiale în dezvoltarea aplicaţiilor Web. Pe baza acestora şi a tehnologiei *Uniform Resource Identifier* s-a definit conceptul de *arhitectură orientată pe resurse (ROA)*.

În continuare s-au prezentat principiile esenţiale ale arhitecturilor orientate pe resurse şi s-a analizat în detaliu modul de aplicare a tehnologiei *URI (Uniform Resource Identifier)* care stă la baza definirii unor stiluri în proiectarea aplicaţiilor, astfel încât sunt identificate arhitecturi *RESTful*, *RPC* şi, respectiv, *mixte*.

În acest capitol s-au prezentat detalii tehnice despre aceste arhitecturi Web şi s-a evidenţiat tendinţa actuală de organizare a resurselor aplicaţiilor Web conform principiilor REST (*Representational State Transfer*).

Scopul clarificării acestor arhitecturi este de a face Web-ul programabil mai bine organizat şi mai clar prin:

- O structurare mai bună a resurselor;
- Creşterea gradului de uniformitate în modul de apelare a resurselor;
- Obţinerea avantajelor maxime pe baza capabilităţilor protocolului HTTP.

Principiile care diferenţiază arhitecturile Web analizate în acest capitol vor fi aplicate în această teză într-un capitol ulterior, în care se propune o arhitectură de aplicaţii Web.

În continuare în acest capitol se prezintă problematica organizării resurselor unei aplicaţii Web conform principiilor REST, se abordează problema organizării logice a resurselor unei aplicaţii Web, a integrării în sistemele noi a resurselor existente anterior.

În finalul capitolului se propune o variantă de implementare a arhitecturii orientate pe resurse, denumită în această lucrare *Arhitectură RESTful Multilayer*.

Contribuţiile aduse în acest capitol sunt următoarele:

- S-a efectuat un studiu asupra problemei adresabilităţii resurselor şi o analiză a evoluţiei modului de adresare a resurselor într-o aplicaţie Web. S-a prezentat o clasificare a arhitecturilor de aplicaţii pe baza modului de adresare a resurselor cu evidenţierea elementelor specifice şi a avantajelor acestora.
- S-a propus un mod de proiectare a ierarhiei resurselor algoritmice ale unei aplicaţii Web pentru a implementa principiile arhitecturale REST (*Representational State Transfer*), cu scopul de a obţine o structurare mai bună a resurselor, o creştere a gradului de uniformitate a modului de adresare a resurselor şi la utilizarea cu avantaje maxime a protocolului HTTP.
- S-a propus o tehnică de adaptare a ierarhiei virtuale a resurselor, utilizată la nivelul superior al aplicaţiilor, cu ierarhia reală a resurselor de pe nivelurile inferioare, care pot fi resurse nou create, sau pot fi moştenite din sistemele informatice existente anterior. Prin aceasta, s-a aplicat principiul păstrării compatibilităţii cu sistemele existente în cazul dezvoltării sistemelor complexe.
- S-a propus şi s-a definit o arhitectură orientată pe resurse, pentru aplicaţii distribuite şi stratificate, în care fiecare strat are un rol său specific în organizarea ierarhică şi adresabilitatea resurselor utilizate într-o aplicaţie Web, toate acestea având la bază principiul interfeţelor uniforme pentru accesul la resurse. Această arhitectură a fost denumită *Arhitectură RESTful Multilayer*.

Aspectele evidenţiate în acest capitol, concluziile la care a ajuns autorul, precum şi variantele de implementare adoptate pentru punerea în practică a principiilor RESTful şi a Arhitecturii RESTful Multilayer au fost publicate în [57], [58] şi [60].

3. FIABILITATEA SERVICIILOR WEB

3.1. DISPONIBILITATE, SIGURANŢĂ, FIABILITATE

Serviciile Web sunt din ce în ce mai mult folosite în sisteme distribuite, în aplicaţii şi servicii de importanţă majoră. Pentru sistemele distribuite, arhitectura orientată pe servicii, *SOA - Service Oriented Architecture*, care foloseşte serviciile Web, este acceptată deja ca fiind arhitectura capabilă să interconecteze aplicaţii care rulează pe diferite sisteme de operare şi facilitează interacţiuni complexe între sisteme autonome şi eterogene, fie în cadrul organizaţiilor, fie între organizaţii aflate în relaţii de tip B2B (*business-to-business*)

Serviciile Web permit aplicaţiilor software aflate în diferite organizaţii (companii) să interacţioneze unele cu altele, chiar dacă organizaţiile respective folosesc sisteme hardware diferite, sau sisteme de operare şi chiar limbaje de programare diferite. Serviciile Web sunt capabile să uniformizeze şi să eficientizeze activităţile de business pe Internet prin invocarea automată a unor operaţii care, altfel, ar trebui invocate manual de un operator uman. Deci, serviciile Web permit interacţiunile directe între calculatoare situate în organizaţii diferite.

Beneficiile aduse de serviciile Web sunt semnificative deoarece ele facilitează automatizarea unor activităţi în cadrul unor afaceri distribuite pe Internet între mai multe companii, precum şi colaborarea între organizaţii prin interconectarea proceselor care rulează în sistemele informatice ale acestora.

Eşecurile şi avariile din aplicaţiile Web pot conduce la procesare eronată sau chiar la căderi de sistem în afaceri de tip e-commerce, e-banking sau alte sisteme bazate pe tranzacţii.

Una din cauzele importante ale întreruperilor serviciilor este „căderea” serverelor. Prin urmare, sunt din ce în ce mai importante tehnicile care asigură toleranţa la erori şi furnizarea neîntreruptă a serviciilor pe Internet chiar în condiţiile unor avarii ale serverelor.

Pentru multe sisteme distribuite bazate pe servicii Web există un număr mare de aplicaţii client deja instalate şi este foarte complicată modificarea tuturor acestor aplicaţii. Acesta este motivul pentru care furnizorii de servicii caută anumite soluţii tolerante la erori, numite *client-transparent*, care să nu necesite nici o acţiune specială din partea aplicaţiilor client şi nici modificarea acestora. Această transparenţă pe partea aplicaţiei-client este o cerinţă importantă atât pentru aplicaţia propriu-zisă, cât şi pentru sistemul de operare respectiv.

În aplicaţiile complexe, serviciile Web trebuie să se conecteze la alte servicii cu scopul de a forma servicii Web compozite şi arhitecturi complexe bazate pe servicii (SOA). Dacă o componentă din lanţul de servicii nu este disponibilă sau nu este fiabilă, atunci întreg sistemul este afectat.

Un *serviciu corect* este cel care implementează funcţia sistemului respectiv. Atunci când serviciul furnizat către client diferă de serviciul corect înseamnă că este vorba de un *eşec al sistemului*. Această abatere înseamnă că serviciul nu respectă specificaţiile sale bine-precizate. O eroare este acea parte din starea sistemului care poate determina un eşec. Cauza erorii este un *defect*. Un defect poate fi activ sau nu. Când este activ, defectul produce o *eroare în sistem*. Erorile grave duc la *eşecul sistemului* (căderea sistemului) [3].

Serviciile Web pot să introducă noi probleme în sistemele informatice ale organizaţiilor, de exemplu:

- Defectele existente în sistemul informatic al unei organizaţii pot afecta negativ sistemul informatic al unei organizaţii partenere;
- Consistenţa datelor, integritatea şi confidenţialitatea lor sunt mai greu de menţinut;
- Lipsa disponibilităţii, a fiabilităţii şi securităţii datelor pot deteriora relaţiile dintre o companie şi clienţii, furnizorii şi partenerii săi.

Aceste probleme devin tot importante şi mai solicitante pe măsură ce activităţile din sistemul informatic necesare desfăşurării afacerilor devin tot mai automatizate, pe măsură ce serviciile Web apelează alte servicii Web, iar activităţile de afaceri necesită tot mai mulţi paşi.

Disponibilitatea (engl. *availability*) unui sistem se referă la capacitatea unui sistem de „a fi gata de a oferi un serviciu corect".

Încrederea/siguranţa (engl. *reliability*) se referă la capacitatea unui sistem de „a putea furniza continuu un serviciu corect".

Fiabilitatea (engl. *dependability*) este un concept mai cuprinzător care integrează mai multe elemente: disponibilitate, încredere/siguranţă, securitate, confidenţialitate, integritate, mentenabilitate [2].

Un mijloc foarte important de a atinge fiabilitatea este *toleranţa la erori*. Aceasta se referă la tehnicile implementate într-un sistem care îi dau capacitatea de „a furniza un serviciu corect chiar în prezenţa unor erori".

În sistemele considerate *fiabile*, replicarea este o tehnică larg acceptată, ce permite evitarea căderilor de sistem. Astfel, arhitecţii de sistem implementează un serviciu într-un sistem distribuit, folosind un grup de servere, independente din punct de vedere fizic, astfel încât dacă o parte din acestea încetează să funcţioneze, serverele rămase au capacitatea de a furniza clienţilor serviciul respectiv [15].

Replicarea protejează o aplicaţie, ce rulează pe un server, împotriva defectelor, astfel încât, dacă o replică devine inoperantă, altă replică este disponibilă să ofere acel serviciu clienţilor. Strategiile de replicare cele mai folosite sunt clasificate astfel: pasive, active şi semi-active. O prezentare a acestor strategii se găseşte în [41].

3.2. TEHNICI TRADIŢIONALE DE REPLICARE

Replicarea datelor constă în menţinerea mai multor copii ale datelor, numite *replici*, pe calculatoare separate. Replicarea este o tehnologie importantă în domeniul serviciilor distribuite. Replicarea îmbunătăţeşte disponibilitatea datelor prin faptul că permite accesul utilizatorilor la date duplicate din vecinătatea apropiată, chiar şi atunci când unele copii ale datelor solicitate nu sunt accesibile.

Replicarea îmbunătăţeşte performanţa sistemului prin reducerea timpilor de aşteptare atunci când unui utilizator i se oferă date situate în duplicatele din vecinătatea imediată evitând, astfel, accesul la distanţă mare. Mai mult, există şi o creştere a performanţei datorită replicării datelor, care se manifestă prin deservirea simultană a mai multor aplicaţii client.

Tehnicile tradiţionale de replicare urmăresc menţinerea unei consistenţe

între pachetul principal de date şi o singură copie, iar aplicaţiile client „văd" un singur set de date cu grad ridicat de disponibilitate. Conceptul de bază este următorul: se blochează accesul la datele replicate până când acestea sunt actualizate, iar această actualizare este dovedită în cadrul aplicaţiei. De aceea, aceste tehnici se numesc „pesimiste".

Algoritmii de lucru folosesc cel puţin două replici ale datelor. În multe cazuri se alege una din aceste două replici care primeşte *rolul de replică principală*. După modificarea unor date, replica principală transmite modificările celorlalte replici şi numai după această sincronizare a datelor se permite aplicaţiei client să efectueze noi operaţii asupra datelor.

În cazul în care replica principală devine indisponibilă, dintre replicile rămase trebuie aleasă una care să devină replica principală. Aceste tehnici ale replicării pesimiste se pot folosi cu succes în reţele locale, unde vitezele de transfer sunt relativ mari, latenţa este relativ mică şi căderile de sistem sunt relativ rare.

Entitatea replicată poate fi de mai multe tipuri:

- obiect;
- fişier;
- structură de date (în sensul bazelor de date);
- serviciu.

De aceea, tehnicile de replicare pot să fie diferite pentru a putea opera cu elemente specifice. Există cel puţin două categorii de tehnici de replicare:

- tehnici destinate replicării bazelor de date;
- tehnici destinate replicării obiectelor şi proceselor din sistemele distribuite.

Tehnicile din cele două categorii de mai sus prezintă multe asemănări, dar şi deosebiri importante [67].

Datorită progresului continuu al tehnologiilor Internet apare şi tendinţa de a aplica algoritmii specifici replicării pesimiste la reţele de largă întindere geografică (WAN). Totuşi, în acest caz nu sunt previzibile aceleaşi performanţe şi aceeaşi disponibilitate a datelor ca şi în cazul reţelelor locale datorită unor motive precizate în cele ce urmează.

În primul rând, Internetul este relativ lent şi nu oferă aceeaşi fiabilitate şi aceeaşi disponibilitate a datelor ca reţelele locale [16]. Aceste probleme sunt mai evidente odată cu folosirea din ce în ce mai mult a calculatoarelor mobile cu conectivitate intermitentă.

Un algoritm de replicare pesimistă dacă încearcă o sincronizare cu un site devenit indisponibil (inaccesibil) va ajunge într-o stare de blocare (blocat în aşteptarea unui eveniment extern). Mai mult, există chiar şi posibilitatea coruperii datelor pentru că, în cazul unei întreruperi a reţelei, timpul de revenire nu este predictibil [12] şi, astfel, este imposibilă alegerea corectă a unei replici principale [25].

În al doilea rând, algoritmii de replicare pesimistă nu sunt uşor de scalat pentru a fi folosiţi în reţele mari, sau în reţele de largă întindere geografică (WAN). Este dificilă construirea pe un WAN a unui sistem de replicare pesimistă, de dimensiune mare, în care apar frecvente actualizări ale datelor deoarece, odată cu creşterea numărului de site-uri deservite, parametri importanţi de performanţă se degradează: timpii de răspuns şi disponibilitatea datelor [75]. De aceea, multe servicii Internet adoptă tehnologii de replicare optimistă.

În al treilea rând, unele activităţi ale utilizatorilor umani necesită partajarea datelor. În multe domenii inginereşti şi, mai ales, în ingineria software, specialiştii

lucrează deseori pe probleme punctuale, bine delimitate şi specificate, într-o anumită izolare până la finalizarea lucrării respective. De acea, este mai bine să se permită actualizarea independentă a datelor în depozitele centralizate de date ale organizaţiilor şi rezolvarea ulterioară a diferenţelor şi conflictelor apărute, decât să se blocheze accesul la un anumit set de date până când un anumit operator uman termină de editat datele respective [65].

3.3. CARACTERISTICILE REPLICĂRII OPTIMISTE

Replicarea optimistă semnifică un set de tehnici pentru partajarea eficientă a datelor în medii mobile sau în medii de lucru de dimensiune mare. *Trăsătura esenţială* care diferenţiază algoritmii pentru replicarea optimistă de cei pentru replicarea pesimistă este modul în care se abordează *controlul accesului concurent*.

Algoritmii pesimişti coordonează sincronizarea între replici în timpul accesului şi blochează noile cereri ale utilizatorilor pe durata unei actualizări în desfăşurare.

Algoritmii optimişti permit accesul la date chiar dacă nu s-a realizat încă sincronizarea replicilor pe baza ipotezei optimiste că nu vor apare conflicte decât rar, sau deloc. Actualizările sunt propagate în fundal către replici, iar conflictele ocazionale sunt rezolvate atunci când apar. Acest mod optimist de a vedea lucrurile nu este nou, dar s-a răspândit pe larg pe măsură ce Internetul şi tehnologiile mobile au devenit din ce în ce mai folosite.

Comparativ cu tehnicile tradiţionale de replicare pesimistă, replicarea optimistă promite un grad ridicat de performanţă a sistemelor şi de disponibilitate a datelor, dar acceptă pe o durată limitată diferenţe între datele principale şi duplicatele lor, această inconsistenţă urmând să fie rezolvată în timp util.

Algoritmii optimişti oferă mai multe avantaje faţă de cei pesimişti în domeniul aplicaţiilor distribuite pe Internet. În primul rând, aplicaţiile progresează chiar dacă legăturile de reţea sau site-urile sunt nesigure.

În al doilea rând, algoritmii optimişti sunt flexibili în ceea ce priveşte lucrul în reţea, în sensul că tehnicile folosite propagă operaţii în mod fiabil către toate replicile, chiar dacă graficul legăturilor de reţea este necunoscut şi variabil.

În al treilea rând, aceşti algoritmi optimişti se scalează uşor în cazul unui număr mare de replici, deoarece ei necesită un efort mai mic de sincronizare între site-uri.

În al patrulea rând, algoritmii optimişti permit site-urilor şi utilizatorilor să rămână autonomi, adică este posibilă adăugarea unei replici la cele existente fără să fie nevoie de modificări ale site-urilor existente.

În al cincilea rând, algoritmii optimişti permit colaborarea asincronă între utilizatori, cum este cazul sistemelor CVS [65].

În final, mai trebuie precizat că algoritmii optimişti dau un răspuns rapid aplicaţiilor client pentru că ei realizează actualizările de date imediat ce au fost solicitate.

Aceste avantaje aduse necesită un cost. Orice sistem distribuit se confruntă cu problema compromisului între consistenţa datelor şi disponibilitatea acestora [44]. În fazele de lucru unde algoritmii pesimişti aşteaptă atingerea unei anumite stări a sistemului, algoritmii optimişti permit continuarea operaţiilor cu aplicaţiile client dar, în fundal, desfăşoară alte operaţii de sincronizare.

Algoritmii optimişti trebuie să facă faţă situaţiilor în care există replici care diferă între ele, ceea ce poate conduce la conflicte între operaţii concurente. De aceea, ei sunt potriviţi numai la aplicaţiile care pot tolera conflicte ocazionale şi date inconsistente un timp limitat.

3.4. ELEMENTELE REPLICĂRII OPTIMISTE

Orice sistem de replicare operează cu conceptul de *unitate minimă de replicare*. Aceste unităţi de replicare sunt, practic, *obiecte*. O *replică* este o copie a unui obiect stocată pe un site, într-un calculator. Replicile mai multor obiecte se stochează pe un *site*. Fiecare obiect este gestionat independent conform algoritmilor de replicare folosiţi.

Unii algoritmi fac distincţie între site-uri care au permisiunea de a modifica obiectele conţinute, numite *site-uri master*, şi site-uri care doar stochează replici accesibile numai pentru citire. Uzual, se notează cu *R numărul total de replici* şi cu *M numărul de replici master ale aceluiaşi obiect*. Un caz obişnuit este cel în care M=1 şi R = M, adică există un singur site master ce conţine o singură replică master pentru fiecare obiect.

Alte elemente importante ale replicării se referă la *operaţii* şi la *modul de propagare a operaţiilor* pe sistemele replică. Un sistem de replicare optimistă trebuie să permită accesul la obiectele replicate chiar dacă unul din sistemele replicate este nefuncţional sau deconectat.

O actualizare a unui obiect din cadrul sistemului poartă numele de *operaţie*. În cazul replicării optimiste, operaţiile diferă de actualizările tradiţionale din bazele de date, deoarece operaţiile sunt propagate şi aplicate în background, deseori după ce a fost trimis un răspuns aplicaţiei client care a iniţiat cererea.

O operaţie poate fi privită ca o precondiţie pentru detectarea conflictelor combinată cu o comandă de actualizare a unui obiect. Natura operaţiilor diferă de la un sistem la altul. Multe sisteme suportă doar actualizări ale întregului obiect vizat. Acestea sunt numite *sisteme bazate pe transferul stării* (engl. *state-transfer systems*). Alte sistem permit o detaliere a operaţiilor de actualizare şi sunt numite *sisteme bazate pe transferul operaţiilor* (engl. *operation-transfer systems*) [47].

Pentru a actualiza un obiect, utilizatorul solicită o anumită operaţie pe un anumit site. Acesta efectuează operaţia la nivel local şi apoi permite utilizatorului să continue să lucreze bazându-se pe faptul că operaţia respectivă a fost efectuată. Site-ul respectiv apelează în background alte servere şi solicită operaţii la distanţă, pentru a propaga solicitarea primită. Aceste sisteme oferă *consistenţă eventuală* pentru că starea replicilor va converge, eventual, către consistenţă.

Această garanţie a consistenţei este, practic, foarte slabă. Ea poate fi considerată suficientă pentru multe aplicaţii de replicare optimistă dar, totuşi, unele sisteme oferă garanţii mai bune ca, de exemplu, cele în care starea unei replici nu este lăsată neactualizată mai mult de un anumit interval de timp.

O operaţie trimisă de un utilizator este înregistrată cu scopul de a fi propagată ulterior pe alte servere replică.

Datorită propagării în background, operaţiile nu sunt întotdeauna primite în aceeaşi ordine pe toate site-urile. Fiecare site trebuie să-şi reordoneze operaţiile primite pentru a produce rezultate echivalente cu cele oferite de celelalte replici din sistem. De aceea, o operaţie este considerată iniţial o *tentativă*.

O replică trebuie să-şi reordoneze în mod repetat operaţiile până când ajunge la o concordanţă cu celelalte replici din sistem asupra ordinii finale a operaţiilor. Termenul folosit pentru politica de ordonare a operaţiilor este cel de *planificare* (engl. *scheduling*).

Fără o coordonare a site-urilor, dinainte stabilită, este posibil ca mai mulţi utilizatori să actualizeze acelaşi obiect aproximativ în acelaşi timp. O soluţie relativ simplă ar fi alegerea arbitrară a unei modificări cerute de un utilizator şi ignorarea celorlalte. Totuşi, o asemenea politică de tratare a solicitărilor concurente determină apariţia de *actualizări pierdute*. Aceste actualizări pierdute nu sunt de dorit în multe aplicaţii distribuite.

O soluţie mai bună pentru tratarea acestor probleme constă în *detectarea operaţiilor care sunt în conflict* şi *rezolvarea* lor prin renegocierea succesiunii lor.

Un *conflict* apare când condiţiile necesare unei operaţii sunt încălcate, chiar dacă operaţiile s-au desfăşurat după politica de planificare din sistem.

3.5. CONCLUZII ŞI CONTRIBUŢII

În acest capitol se prezintă, mai întâi, aspecte care evidenţiază impactul semnificativ pe care îl au serviciile Web în dezvoltarea aplicaţiilor distribuite. Arhitectura orientată pe servicii, *SOA - Service Oriented Architecture*, care foloseşte serviciile Web, este acceptată deja ca fiind arhitectura capabilă să interconecteze aplicaţii care rulează pe diferite sisteme de operare şi facilitează interacţiuni complexe între sisteme autonome şi eterogene, fie în cadrul organizaţiilor, fie între organizaţii aflate în relaţii de tip B2B (business-to-business)

Serviciile Web permit aplicaţiilor software aflate în diferite organizaţii (companii) să interacţioneze unele cu altele, chiar dacă organizaţiile respective folosesc sisteme hardware diferite, sau sisteme de operare şi chiar limbaje de programare diferite.

Serviciile Web sunt capabile să uniformizeze şi să eficientizeze activităţile de business pe Internet prin invocarea automată a unor operaţii care, altfel, ar trebui invocate manual de un operator uman, ceea ce înseamnă o probabilitate mai mare de eroare.

Beneficiile aduse de serviciile Web sunt semnificative deoarece ele facilitează automatizarea unor activităţi în cadrul unor afaceri distribuite pe Internet între mai multe companii, precum şi colaborarea între organizaţii prin interconectarea proceselor care rulează în sistemele informatice ale acestora.

Contribuţiile aduse în acest capitol sunt următoarele:

- S-a studiat problema fiabilităţii serviciilor Web în contextul utilizării lor pentru dezvoltarea unor aplicaţii bazate pe principiile arhitecturilor orientate pe servicii (*SOA – Service Oriented Architectures*).
- S-au analizat şi evidenţiat conceptele de *disponibilitate*, *siguranţă* şi *fiabilitate* şi s-au prezentat detalii despre taxonomia termenilor folosiţi în domeniul fiabilităţii sistemelor: *serviciu corect*, *eroare*, *eşec al sistemului*, *defecte*, *toleranţă la defecte*.
- S-a prezentat tehnica de *replicare* folosită cu scopul de a realiza sisteme tolerante la defecte şi s-au analizat tehnicile tradiţionale de replicare cu caracteristicile lor.

- S-a analizat conceptul de *replicare optimistă* şi s-au prezentat caracteristicile sale.

Studiile şi analizele făcute în acest capitol vor folosi ulterior în această teză în definirea unei arhitecturi de sistem distribuit, tolerant la defecte, bazat pe servicii Web.

4. MODELAREA UNUI SISTEM DISTRIBUIT CU MANAGEMENT CENTRALIZAT

4.1. PROCESE ŞI CONEXIUNI

Abstractizarea sistemului trebuie să înceapă cu abstractizarea infrastructurii fizice pe care o va folosi sistemul. Definirea modelului de sistem presupune, în primul rând, descrierea elementelor relevante cu proprietăţile lor specifice şi precizarea modului în care aceste elemente interacţionează.

În această lucrare sunt folosite două elemente abstracte care permit reprezentarea infrastructurii fizice a sistemului:

- *procesul;*
- *conexiunea (între procese)*.

În cadrul unui program distribuit, *procesul* abstractizează o entitate activă care îşi desfăşoară activitatea după un anumit algoritm şi realizează o anumită procesare a unor date specifice. Procesul poate reprezenta un calculator, un procesor al unui calculator sau un fir de execuţie al unui procesor.

Procesele trebuie să poată coopera pentru îndeplinirea unor sarcini comune. De aceea ele trebuie să schimbe *mesaje* între ele. Mesajele între procese sunt posibile numai dacă există o anumită legătură fizică între procese.

Conexiunea abstractizează la nivel fizic (şi la nivel logic) legătura între procese şi stă la baza *comunicaţiei între procese*. Organizate într-un anumit mod, legăturile formează *reţele*.

Comunicaţia între procese presupune următoarele elemente (Fig.4.1):

- *mesajul – m;*
- *emiţătorul mesajului (sender) – procesul S*
- *receptorul mesajului (receiver) – procesul R*.

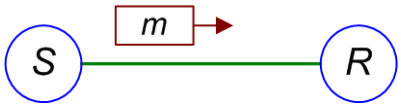

Fig. 4.1. Comunicaţia între procese

Descrierea unui sistem distribuit presupune o multitudine de proprietăţi ale acestor procese şi conexiuni, precum şi modul în care aceste elemente operează (sau încetează să opereze) în anumite condiţii ale mediului de lucru.

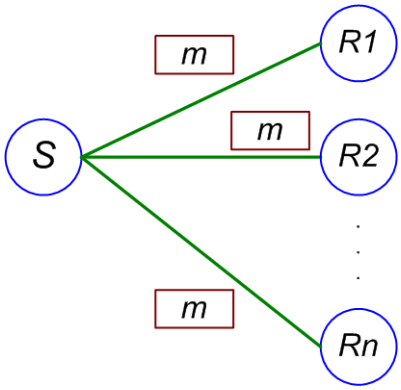

Fig. 4.2. Un proces emiţător şi multe procese receptoare

4.2. LOCALIZAREA ŞI ROLUL PROCESELOR

În multe cazuri este importantă *localizarea proceselor*, respectiv elementele care găzduiesc procesele respective şi le asigură condiţii de desfăşurare. Pentru aceasta se foloseşte elementul abstract numit *gazdă* (engl. *host*).

Un caz uzual este cel în care gazda este calculatorul pe care se desfăşoară un anumit proces (alături de alte procese în cadrul sistemului de operare). Importanţa unei gazde este dată de faptul că asigură anumite condiţii de mediu de lucru pentru procesul pe care-l găzduieşte, iar desfăşurarea procesului este influenţată de condiţiile oferite de gazdă (Fig.4.3).

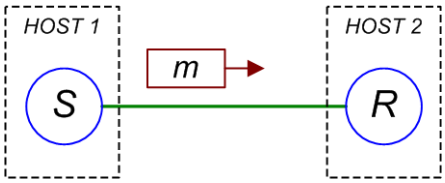

Fig. 4.3. Procese localizate pe gazde diferite

Folosind elementele abstracte precizate până acum (proces, conexiune, gazdă) se pot modela sisteme distribuite mai complexe în care un proces comunică prin mesaje cu mai multe procese, unele localizate pe aceeaşi gazdă, altele localizate pe gazde diferite (Fig.4.4).

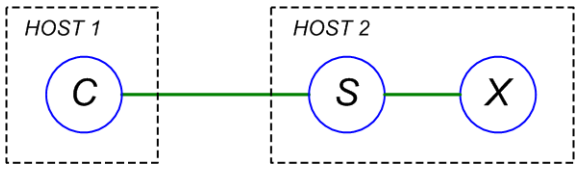

Fig. 4.4. Procesul S comunică cu mai multe procese

Se pot construi sisteme în care fiecare proces să aibă un anumit *rol* în cadrul sistemului distribuit, iar pentru a-şi îndeplini rolul, procesul trebuie să fie capabil să desfăşoare *activităţi specifice*. În aceste cazuri este importantă *natura mesajelor* primite şi transmise între procese.

De exemplu, se pot construi sisteme distribuite în care un proces C, cu rol de *client*, trimite un mesaj către un proces S, cu rol de *server*, prin care îi cere să execute anumite operaţii şi să-i dea un mesaj de răspuns. Procesul S are o parte de procesare internă, dar poate să trimită un mesaj către un alt proces X, cu rol de *executant*, solicitându-i operaţii specifice. În acest caz, procesul S trebuie să comunice atât cu procesul C, cât şi cu procesul X (Fig.4.4).

În cazul unor sisteme distribuite complexe, se pot defini mai multe procese cu rol de *executant*, notate, de exemplu, X1, X2 ş.a.m.d., situate pe gazde diferite, care desfăşoară activităţi specifice în urma recepţionării unor mesaje specifice de la un proces cu rol de *manager*, notat, de exemplu, cu M (Fig.4.5).

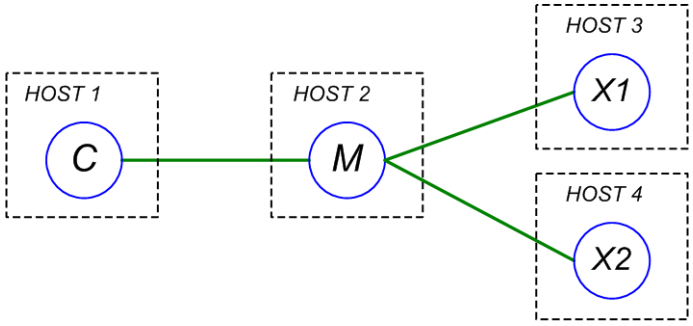

Fig. 4.5. Procesul M are rol de manager pentru procesele executante X1, X2

4.3. PROBLEMA ACORDULUI DISTRIBUIT

Un model de sistem distribuit abstractizează interacţiunile din cadrul sistemului. Acestea se referă la cooperarea dintre procese. Această cooperare poate fi modelată ca o *problemă de acord distribuit* (engl. *distributed agreement*).

Între procesele unui sistem distribuit trebuie să existe un *acord* referitor, de exemplu, la: modul de interpretare a unui anumit set de date de intrare, modul de exprimare a identităţii proceselor, efectuarea unei anumite succesiuni de operaţii din mai multe variante posibile, respectiv realizarea unui consens, etc.

Într-un sistem distribuit este posibil ca să existe un acord între procesele participante referitor la o anumită operaţie, care trebuie să aibă loc numai dacă sunt îndeplinite mai multe condiţii exprimate de date diferite existente în procesele participante. În cazul când nu sunt îndeplinite *toate condiţiile stabilite*, atunci procesele participante sunt „de acord” ca operaţia respectivă să nu aibă loc. Această formă de acord între procesele unui sistem distribuit este folosită în cazul *tranzacţiilor*.

Într-un sistem distribuit, procesele participante într-un program distribuit trebuie să fie de *acord asupra operaţiilor* care trebuie efectuate, dar trebuie să existe şi un *acord asupra ordinii în care trebuie efectuate operaţiile*. Această formă de acord distribuit este una din tehnicile de bază din sistemele care folosesc

replicarea proceselor în scopul realizării toleranţei la defecte (engl. *total order broadcast*).

4.4. PROCESE ŞI OPERAŢII

Pentru un proces considerăm elementul abstract numit *operaţie*. În cadrul unui sistem distribuit un proces participant transmite şi primeşte mesaje de la celelalte procese participante. Natura mesajelor poate determina executarea unor operaţii diferite în cadrul procesului.

O operaţie constă din următoarele etape:

- recepţionarea unui mesaj trimis de un alt proces;
- executarea unor calcule la nivel local;
- trimiterea unui mesaj către un alt proces din sistem.

Recepţionarea şi transmiterea mesajelor sunt *evenimente globale* în sistem deoarece la ele participă mai multe procese, cel puţin două. Executarea calculelor la nivel local nu implică o participare directă a altor procese, deci sunt *evenimente interne* ale procesului respectiv.

Procesul poate avea mai multe module, fiecare putând participa la o etapă a unei operaţii: recepţionarea unui mesaj, executarea unor calcule, trimiterea unui mesaj către un alt proces. Aceste module au roluri specifice şi cooperează între ele.

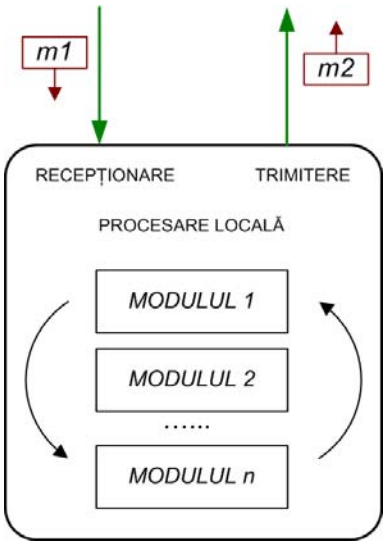

Fig. 4.6. Etapele unei operaţii în cadrul unui proces dintr-un sistem distribuit

Procesul poate avea o organizare pe mai multe niveluri, un modul fiind plasat, din punct de vedere al logicii de programare, pe un anumit nivel. Un proces complex este cel în care trebuie să se execute o multitudine de operaţii şi, de aceea, la nivel software, procesul are o structură stratificată numită şi *stiva de componente software*.

4.5. NIVELURI, COMPONENTE ŞI EVENIMENTE

La fiecare proces, stiva de componente software conţine câte o componentă pentru fiecare nivel din structura logică a procesului. Nivelul superior este numit nivelul Aplicaţie, în timp ce nivelul inferior este nivelul Reţea. În modelul de sistem distribuit, elementele care abstractizează programarea distribuită sunt la mijloc în stiva de niveluri.

Componentele software situate pe niveluri (*layers*) diferite în aceeaşi stivă de componente comunică între ele prin *evenimente* (Fig.4.7).

Fiecare componentă este, la un moment dat, într-o anumită *stare*. Recepţionarea unui eveniment declanşează o *tranziţie* către o altă stare a componentei.

Evenimentul este o grupare de informaţii aşezate într-o structură bine stabilită, definită anterior şi cunoscută de componentele care recepţionează şi/sau transmit acel eveniment. Regulile de comunicare dintre componente determină anumite *tipuri de evenimente*.

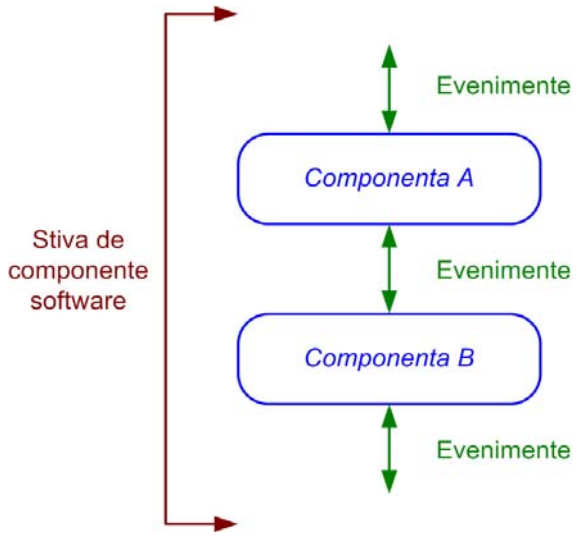

Fig. 4.7. Modelul de proces cu mai multe componente

Astfel, fiecare eveniment are următoarele caracteristici:

- este de un anumit *tip*;
- are o anumită *sursă* (componenta care a transmis evenimentul);
- are o serie de *atribute*;
- are o *destinaţie* (transportă anumite informaţii pentru alte componente).

Notaţia pentru un eveniment folosită în această lucrare începând de aici este:

$$\{ comp, \text{EvType} \mid attr1, attr2 \ldots \} \quad (1)$$

În notaţia de mai sus simbolurile folosite sunt:

- *comp* este componenta sursă a evenimentului;
- EvType este tipul evenimentului;
- *attr1*, *attr2* … etc. sunt atributele evenimentului.

Observaţii:

- mai multe componente pot folosi evenimente de acelaşi tip;
- mai multe componente pot folosi acelaşi eveniment;
- un eveniment recepţionat de o componentă poate fi retransmis fără modificări către alte componente, păstrând sau nu informaţia despre sursa iniţială.

Un eveniment recepţionat de componenta destinaţie este procesat de o entitate software special destinată acestei operaţii, numită *handler*. În pseudo-codul care prezintă ce operaţii trebuie efectuate pe baza recepţionării evenimentului se va folosi o instrucţiune special destinată acestui lucru, care să indice evenimentul şi instrucţiunile care trebuie executate datorate apariţiei evenimentului respectiv: *upon event*.

Procesarea unui eveniment are o serie de caracteristici:

- Un eveniment declanşat de o componentă este procesat numai dacă procesul din care fac parte toate componentele respective se desfăşoară corect.
- Procesarea unui eveniment poate declanşa crearea de noi evenimente de către aceeaşi componentă sau de componente diferite.
- Componenta destinaţie a unui eveniment poate filtra în mod explicit evenimentele. Acest lucru se va indica în pseudo-cod prin clauza *such that*.
- Evenimentele aceleiaşi componente sunt procesate în care au fost declanşate.
- Evenimentele schimbate între componentele din cadrul aceleiaşi stive de componente se ordonează la modul FIFO (*first-in-first-out*).

Interfeţele componentelor conţin două *tipuri de evenimente*: cereri (*requests*) şi indicaţii (*indications*).

Evenimentele de tip *request* sunt folosite de o componentă în următoarele scopuri:

- pentru a invoca un serviciu oferit de o altă componentă;
- pentru a semnaliza o condiţie altei componente.

Din punctul de vedere al modului de tratare la nivelul componentei software, evenimentele de tip *request* sunt *evenimente de intrare* (*input*).

Evenimentele de tip *indication* sunt folosite de o componentă în următoarele scopuri:

- pentru a livra o informaţie;
- pentru a semnaliza o condiţie altei componente.

Din punctul de vedere al modului de tratare la nivelul componentei software, evenimentele de tip *indication* sunt *evenimente de ieşire* (*output*).

Considerăm un proces care conţine mai multe componente software, fiecare fiind situată pe un anumit nivel (*layer*) în structura logică a procesului.

Fiecare nivel comunică cu nivelul superior (dacă există) printr-o serie de evenimente de tip *request* şi *indication*. Pe de altă parte, fiecare nivel comunică cu

nivelul inferior (dacă există) printr-o altă serie de evenimente de tip *request* şi *indication*. Propagarea evenimentelor între nivelurile procesului, cererile de sus în jos şi, respectiv, indicaţiile de jos în sus este prezentată în Fig. 4.8.

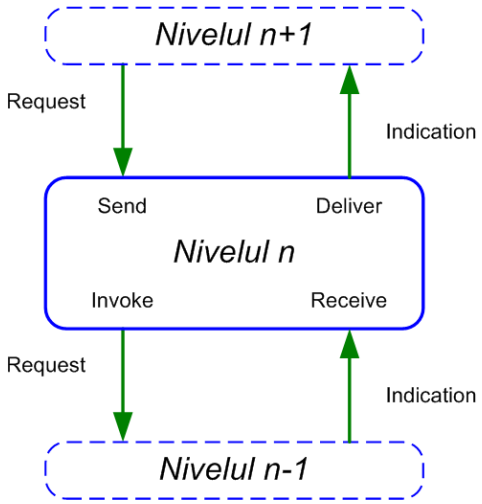

Fig. 4.8. Comunicaţii între nivelurile unui proces

La un anumit *Nivel n*, execuţia constă în următoarele etape:

- Mesajul *Request* difuzat de componenta software de pe nivelul superior, nivelul n+1, este recepţionat la nivelul n unde se declanşează procedura *Send* pentru trimitere a mesajului mai departe spre nivelurile inferioare din stivă.
- Componenta de pe nivelul n execută procedura *Invoke* pentru a invoca servicii de pe nivelul ierarhic inferior, nivelul n-1, folosind pentru aceasta evenimente *Request* specifice nivelului inferior.
- Mesajele de tip *Indication* trimise de nivelul inferior sunt recepţionate la nivelul n în procedura *Receive*. Ele trebuie transmise mai departe spre nivelul superior.
- Mesajul de tip *Indication*, dacă îndeplineşte condiţiile necesare pentru siguranţa procesului, sunt livrate nivelului superior în procedura *Deliver*.

Mesajele de tip *Request* sau de tip *Indication* care sunt transmise între niveluri nu au întotdeauna o încărcătură de date. Uneori pot să indice doar condiţii pentru sincronizarea nivelurilor. De exemplu, nivelul superior poate să difuzeze un mesaj specializat către celelalte niveluri pentru a semnala că o anumită fază de procesare s-a încheiat şi că se va trece la faza următoare.

4.6. COMUNICAŢIA ÎNTRE PROCESE

Considerăm două procese S (*sender*) şi R (*receiver*) între care există stabilită o conexiune. Pe această conexiune procesele îşi transmit mesaje (Fig.4.1).

Pentru modelarea comunicaţiei dintre procese vom considera conexiunea *punct-la-punct* dintre două procese ca o entitate software (un obiect) *conn*, o instanţă a unei clase denumită *PointToPointConnection*.

La nivelul obiectului *conn* au loc două evenimente distincte, unul la capătul dinspre procesul sursă *S*, iar celălalt la capătul dinspre procesul destinaţie *R*.

Evenimentul care modelează *trimiterea mesajului m* de către procesul-sursă *S* este:

$$\{ conn, \text{Send} \mid R, m \} \tag{2}$$

Evenimentul care modelează *recepţionarea mesajului m* către procesul *R* este:

$$\{ conn, \text{Receive} \mid S, m \} \tag{3}$$

În relaţia (2), atributele evenimentului sunt: *destinatarul R* şi *mesajul m*. În relaţia (3), atributele evenimentului sunt: *emiţătorul S* şi *mesajul m*.

În modelul de sistem distribuit analizat în această lucrare, corectitudinea comunicaţiei dintre procese presupune respectarea următoarelor *proprietăţi ale comunicaţiei sigure dintre procesele corecte ale sistemului distribuit*:

- **P1.** *Livrare fiabilă a mesajelor*. Dacă procesul *S*, care se desfăşoară corect, transmite un mesaj *m* către procesul *R*, care, de asemenea, se desfăşoară corect, atunci procesul *R* eventual recepţionează mesajul *m*.
- **P2.** *Mesajele nu sunt duplicate la sursă*. Un proces corect, într-o anumită fază a execuţiei sale, transmite un mesaj o singură dată.
- **P3.** *Mesajele nu sunt create la destinatar*. Dacă un mesaj *m* a ajuns la procesul destinaţie *R* din partea procesului *S*, atunci mesajul *m* a fost, cu siguranţă, transmis anterior procesului *R* de către procesul *S*.
- **P4.** *Mesajele sunt recepţionate în ordinea strictă a sosirii lor (FIFO)*. Dacă procesul *R* a recepţionat din partea altui proces *S* mai întâi mesajul *m1* şi ulterior mesajul *m2*, atunci funcţionarea corectă a procesului *R* înseamnă preluarea în ordine a mesajelor: mai întâi mesajul *m1* şi apoi mesajul *m2*.
- **P5.** *Se respectă un acord (agreement) la nivelul sistemului distribuit*. Dacă un mesaj *m* a fost livrat cu succes unui proces *R*, care se desfăşoară corect, atunci mesajul *m* este eventual livrat cu succes oricărui proces corect din sistem.

Corectitudinea proceselor şi comunicaţiei stă la baza bunei funcţionări a sistemului distribuit.

Modelarea succesiunii mesajelor nu poate fi făcută cu doar cu modelul iniţial prezentat în Fig. 4.1 pentru că trebuie luată în considerare o nouă coordonată: *timpul logic al proceselor*.

Lamport a propus în 1978 un model pentru timpul logic tocmai pentru a putea prezenta corect ordinea evenimentelor şi mesajelor în cadrul unui sistem

distribuit [37]. În abordarea sa, execuţia unui proces este modelată ca o succesiune de evenimente atomice, fiecare din ele necesitând pentru execuţie o unitate de timp logic.

În modelul lui Lamport trimiterea şi recepţionarea mesajelor sunt considerate *evenimente* în cadrul proceselor (Fig. 4.9).

Considerăm două procese corecte, notate P şi Q, care îşi desfăşoară activitatea în timp. La nivelul procesului P are loc evenimentul, notat aici *Send(m)*, de trimitere a mesajului *m* către procesul Q.

La nivelul procesului Q au loc două evenimente: unul de recepţionare a mesajului *m* , *Receive(m)*, şi altul de livrare a mesajului, *Deliver(m)*. către componentele procesului Q menite să prelucreze mesajul *m*.

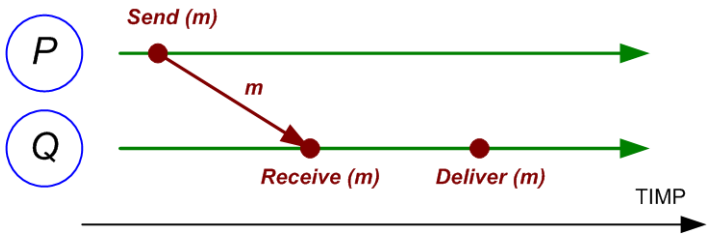

Fig. 4.9. Desfăşurarea în timp a comunicaţiei între două procese

Modelarea schimbului de mesaje conform Fig.4.9 permite *evidenţierea succesiunii evenimentelor*. De exemplu, la nivelul procesului Q, mai întâi are loc evenimentul de recepţionare a mesajului *m* şi ulterior are loc evenimentul de livrare a mesajului spre alte componente ale procesului.

Separarea evenimentelor de recepţionare de cele de livrare a mesajelor permite extinderea modelului abstract al sistemului cu protocoale pentru recepţionarea mesajelor, care pot să conţină o serie de activităţi legate de mesaj înainte ca acesta să fie livrat altor componente ale procesului. Aici trebuie evidenţiate câteva situaţii posibile:

- nu toate mesajele trimise vor fi recepţionate (probleme ale reţelei);
- nu toate mesajele recepţionate trebuie livrate componentelor de prelucrare din cadrul procesului (pot fi mesaje incorecte sau cu alt destinatar).

Având un model de sistem distribuit în care execuţia la nivelul proceselor şi comunicaţia între procese sunt prezentate în legătură cu timpul logic al sistemului, se poate trece la modelarea ordinii mesajelor pe baza relaţiilor de cauzalitate.

4.7. ORDINEA CAUZALĂ A MESAJELOR

În funcţionarea sistemului distribuit procesele trimit mai multe mesaje către alte procese şi, la rândul lor, recepţionează mai multe mesaje.

Transmiterea mesajelor între alte procese presupune faza de transmitere (difuzare) a mesajului pe conexiune şi, ulterior, faza de recepţionare şi cea de livrare a mesajului către procesul destinatar. Transmiterea mesajelor respectă proprietatea denumită *ordine cauzală a mesajelor*. Aceasta se referă la faptul că

mesajele sunt livrate cu respectarea relaţiei *cauză-efect*.

Un mesaj trebuie să aibă o proprietate *happened-before* care să arate această ordonare cauzală (după relaţia *cauză-efect*).

Considerăm două procese şi un mesaj *m1* transmis de la un proces emiţător P către un proces receptor Q. Dacă, la nivelul procesului Q, recepţionarea mesajului *m1* cauzează transmiterea unui alt mesaj *m2* către un alt proces R, atunci între mesajele *m1* şi *m2* există o *ordine cauzală* care se notează astfel:

$$\mathrm{m1} \rightarrow \mathrm{m2} \tag{4}$$

Există mai multe situaţii în care, în modelul abstract al sistemului distribuit, se consideră că mesajul *m1* este o *cauză potenţială* pentru mesajul *m2* (Fig. 4.10), situaţiile notate 1, 2 şi respectiv, 3):

1) un proces a difuzat mesajul *m1* şi ulterior a difuzat mesajul *m2*;
2) un proces a recepţionat mesajul *m1* şi apoi a difuzat mesajul *m2*;
3) la nivelul unui proces există mesajul *m'* astfel încât:
   m1 este anterior lui *m'*, iar *m'* este anterior lui *m2*, ceea ce se notează:

$$m1 \rightarrow m'$$
$$m' \rightarrow m2$$

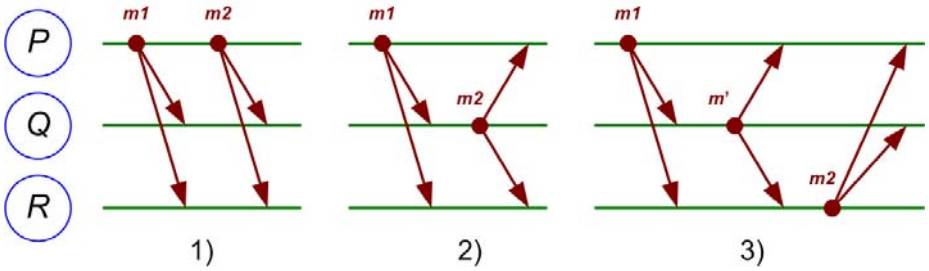

Fig. 4.10. Ordinea cauzală a mesajelor

În faza de livrare a mesajelor către destinatar, ordinea cauzală înseamnă că, dacă un mesaj este livrat, atunci toate mesajele corecte precedente au fost livrate.

În modelul sistemului distribuit se poate abstractiza difuzarea sigură (*reliable broadcast*) a mesajelor în care *livrarea să respecte ordinea cauzală a mesajelor* astfel:

Clasa: *CausalOrderReliableBroadcast*
Denumirea instanţei clasei: *causalOrderReliableBroadcast*
Evenimente:

Difuzare: Difuzează mesajul *m* către toate procesele (la nivelul procesului sursă):

$$\{\,\mathrm{causalOrderReliableBroadcast},\ \mathrm{Broadcast}\ |\ \mathrm{m}\,\} \tag{5}$$

Recepţionare: Recepţionează mesajul *m* difuzat de procesul *P*:

$$\{\,\mathrm{causalOrderReliableBroadcast},\ \mathrm{Receive}\ |\ \mathrm{P},\ \mathrm{m}\,\} \tag{6}$$

Livrare: Livrează mesajul *m* difuzat de procesul *P* (la nivelul procesului destinaţie):

$$\{\,\mathrm{causalOrderReliableBroadcast},\ \mathrm{Deliver}\ |\ \mathrm{P},\ \mathrm{m}\,\} \tag{7}$$

Proprietăţi:

- **P1.** *Validitate*. Dacă procesul corect *S* difuzează un mesaj *m*, atunci mesajul este eventual livrat procesului destinaţie *R*.
- **P2.** *Ne-duplicarea mesajelor*. Un mesaj este livrat o singură dată.
- **P3.** *Ne-crearea mesajelor*. Dacă un mesaj *m* difuzat de procesul sursă *S* a fost recepţionat de procesul destinaţie *R*, atunci mesajul *m* a fost, cu siguranţă, difuzat anterior de către procesul *S*.
- **P4**. *Acord (agreement)*. Dacă un mesaj *m* este livrat de un proces corect, atunci mesajul *m* este eventual livrat în fiecare proces corect din cadrul sistemului.
- **P5**. *Livrare cauzală*. Dacă mesajul *m1* este o cauză potenţială a difuzării mesajului *m2* ($m1 \rightarrow m2$), atunci nici un proces corect nu livrează mesajul *m2* înainte lui *m1*.

4.8. SISTEM DISTRIBUIT CU MANAGEMENT CENTRALIZAT ŞI EXECUŢIE SEPARATĂ

Considerăm un sistem distribuit în care există mai multe procese cu roluri bine definite, situate pe mai multe gazde (Fig.4.11).

Caracteristicile principale ale acestui sistem sunt:

- Procesul C, numit în continuare *client*, are stabilită o conexiune sigură cu procesul M numit, în continuare, *manager*.
- Procesul M este localizat pe o altă gazdă faţă de procesul C.
- Pe conexiunea sigură dintre C şi M circulă mesaje de la C către M, numite *cereri* (*requests*), dar şi mesaje de la M către C numite *răspunsuri* (*responses*).
- Procesul M recepţionează mesajele cerere trimise de clienţi şi le livrează componentelor sale cu *respectarea ordinii în care au fost recepţionate*.
- Procesul manager M este *capabil să stocheze cererile* făcute de procesul client C, într-o formă care permite regăsirea ulterioară a unei cereri.
- În sistem există şi procesele W1, W2, W3, numite în continuare, *procese executant* (*worker*), care au, fiecare dintre ele, stabilită o conexiune sigură cu procesul manager M, dar nu au legături directe între ele.
- Procesele executant W1, W2 şi W3 sunt *localizate pe gazde diferite*, altele decât gazda procesului M.
- Procesele worker W1, W2 şi W3 sunt *deterministe*, în sensul că atunci când primesc mesaje (cereri) cu conţinut identic din partea procesului M, efectuează operaţii identice şi formulează mesaje de răspuns identice, pe care le trimit către procesul manager M. Deci, nu există factori locali care să determine răspunsuri diferite la cereri identice din partea managerului M.
  - Procesul M trimite *mesaje către procesele worker* W1, W2 şi W3 cu *respectarea ordinii cauzale a mesajelor*, ca urmare a recepţionării cererilor primite de la procesele client.
  - Conţinutul mesajelor trimise de M către procesele executant W1, W2, W3 este determinat de doi factori:
    1. conţinutul mesajului cerere trimis de procesul C;
    2. existenţa unui acord (*agreement*) la nivelul sistemului distribuit privind formatul acestor mesaje.

- Managerul M recepţionează mesajele de răspuns trimise de procesele executant W1, W2, W3 şi *procesează aceste răspunsuri într-o fază separată a execuţiei sale* cu scopul de a forma un mesaj de răspuns către client.

Avantajele acestui sistem distribuit sunt:

- folosirea puterii de procesare a mai multor calculatoare fizice datorită localizării separate a proceselor care folosesc intensiv procesorul;
- localizarea datelor stocate mai aproape de alte sisteme care pot solicita accesul la aceste date prin folosirea mai multor procese executant localizate separat;
- timp de răspuns mai mic al procesului de management în relaţia cu procesele client prin separarea procesului de management de procesele de execuţie;
- uşurinţă în depanarea proceselor executant prin localizarea lor separată;
- posibilitatea de adăugare relativ rapidă a altor sisteme executant la sistem;
- posibilitatea de a folosi în cadrul sistemului distribuit procese executant W2, W3 … Wn, cu rol de *replici ale unui proces principal*, W1, având o logică de procesare identică cu cea a procesului principal W1 şi controlate de acelaşi proces manager M, localizat separat.

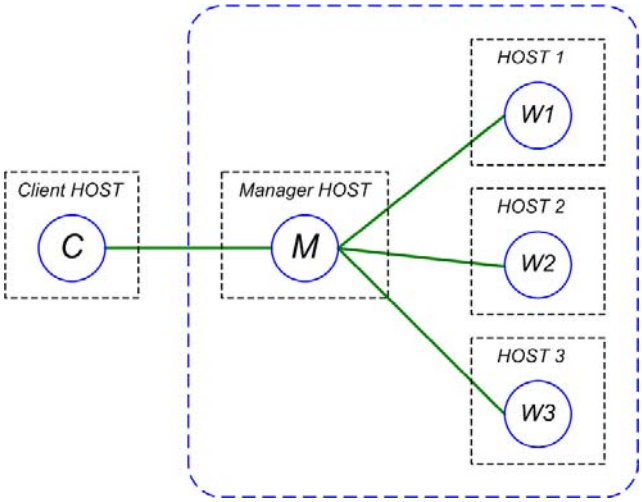

Fig. 4.11. Sistem distribuit cu management centralizat şi execuţie separată

Funcţionarea sistemului distribuit prezentat în figura de mai sus se bazează pe următoarele *ipoteze*:

- Conexiunile dintre procese sunt sigure, astfel încât *difuzarea mesajelor este un proces sigur* (*reliable broadcast*);
- Mesajele trimise de procesele client C sunt aşezate într-o *structură FIFO la nivelul procesului manager M*;

- Mesajele trimise de procesul manager M către procesele executant respectă o *ordine cauzală*;
- Mesajele recepţionate de fiecare proces executant W_i (i = 1, 2 … n) *sunt stocate într-o structură FIFO* şi sunt livrate pe rând componentelor procesului, respectând o *ordine cauzală*.

4.9. SCHIMBUL DE MESAJE ÎNTRE PROCESELE CLIENT ŞI MANAGER

În urma acţiunilor unui utilizator, procesul client C trimite un mesaj către sistemul distribuit, practic către procesul manager, prin care solicită executarea unei acţiuni în sistemul distribuit şi obţinerea unui mesaj de răspuns. Acest mesaj de răspuns conţine date, dar volumul acestora diferă în funcţie de mai mulţi factori, cel mai vizibil fiind natura acţiunii solicitate.

Acest schimb de mesaje între client şi sistem presupune un acord anterior asupra formatului mesajelor, astfel încât procesele aflate în comunicaţie să poată „dialoga”.

Mesajul procesului client conţine numele acţiunii solicitate (sau a serviciului solicitat) şi parametrii necesari execuţiei.

Procedura care trimite un mesaj către procesul manager este:

Execute (action, params, clientID)

Această procedură determină crearea unui eveniment la nivelul procesului client:

{ *client*, Send | [*action, params*], *manager, self* }

Mesajul ajunge la procesul manager M şi, ca urmare, la nivelul acestuia se declanşează, succesiv, procedurile:

Receive (action, params, client)
[operationID, operationName, parameters] := Operation (action, params)
Log (operationID, operationName, parameters)
Execute (operationID, operationName, parameters, *worker*)
Send (result, client)

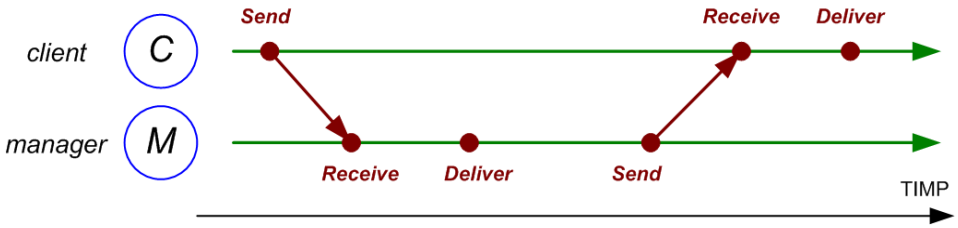

Fig. 4.12. Schimbul de mesaje între procesul client C şi procesul manager M

Abstractizarea schimbului de mesaje între procesul client C şi procesul manager M (prezentate în Fig. 4.12), precum şi procedurile declanşate, sunt prezentate în algoritmul 4.1.

ALGORITMUL 4.1. Mesaje între Client şi Manager şi operaţii efectuate

```
Implements:
        Class Name: Client, instance name: client
        Class Name: Manager, instance name: manager

upon event { manager, Receive | [action, params], client } do
      [operationID, operationName, parameters] := Operation ( action, params )
      successfullyLog := Log ( operationID, operationName, parameters );
      if successfullyLog then
            trigger { self, Deliver | [operationID, operationName, parameters],
                  client };
      else
            result : = "Logging Error. Abort";
            trigger { manager, Send | client, [operationID, result] };

upon event { manager, Deliver | [operationID, operationName, parameters],
client } do
      workers := CheckWorkers();
      for each worker in workers do
            if ( status(worker) == "Ready") then
                  trigger { worker, Send | [operationID, result] , manager , self };
            else
                  trigger { self, Resychronize | worker, operationID }

upon event { client, Receive | result, self, manager } do
      trigger { self, Deliver | result };

upon event { client, Deliver | result } do
      trigger { self, GetResponse | result };
```

4.10. SCHIMBUL DE MESAJE ÎNTRE PROCESELE MANAGER ŞI EXECUTANT

Ca urmare a unei cereri făcute anterior de un proces client, la nivelul procesului manager M (după o procedură de înregistrare a cererii) se declanşează procedura

Execute (operationID, operationName, parameters, worker, self)

Această procedură determină crearea unui eveniment Send la nivelul procesului manager M:

{ *manager*, Send | [*operationID, operationName, parameters*], *worker, self* }

Ca urmare, la nivelul procesului worker W se declanşează, succesiv, procedurile:

Receive (operationID, operationName, parameters, manager)
Log (operationID, operationName, parameters)
Execute (operationName, parameters)
Send (operationID, result, manager, self)

În final, procesul executant W trimite un mesaj către procesul manager M. Ca urmare a acestui mesaj, în cadrul procesului manager M se declanşează succesiv procedurile:

Receive (operationID, result, worker)
PrepareResponseForClient (result)

Procedurile menţionate mai sus folosesc tipul de date numit *operaţie*. Practic, este o clasă publică în cadrul sistemului distribuit. În cadrul sistemului există, deci, un *agreement* pentru folosirea acestei clase pentru transportul datelor între componentele software.

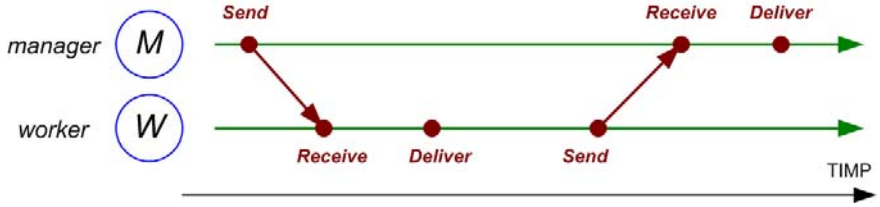

Fig. 4.13. Schimbul de mesaje între procesele manager şi executant

Abstractizarea schimbului de mesaje între procesul manager M şi procesul executant W (prezentate în Fig. 4.13), precum şi procedurile declanşate, sunt prezentate mai jos în algoritmul 4.2.

ALGORITMUL 4.2. Mesaje între Manager şi Worker şi operaţii efectuate

Implements:
Class Name: Manager, instance name: *manager*
Class Name: Worker, instance name: *worker*

upon event { *worker*, Receive | [*operationID, operationName, parameters*], *manager* } do
successfullyLog := Log (*operationID, operationName, parameters*);
if successfullyLog then
trigger { *self*, Deliver | [*operationID, operationName, parameters*], *manager* };
else
result : = ”Logging Error. Abort”;
trigger { *worker*, Send | *manager, [operationID, result]* };

upon event { *self*, Deliver | [*operationID, operationName, parameters*], *manager* } do
result := Execute (operationName, parameters);
trigger { *worker*, Send | *[operationID, result] , manager , self* };

upon event { *manager*, Receive | *[operationID, result], self, worker* } do
trigger { *self*, Deliver | *[operationID, result], worker* };

upon event { *manager*, Deliver | *[operationID, result], worker* } do
trigger { *self*, PrepareResponseForClient | *result* };

4.11. CONCLUZII ŞI CONTRIBUŢII

În acest capitol se propune un model de sistem distribuit care, în capitolul următor, va servi ca bază pentru definirea unui model de replicare pentru asigurarea toleranţei la defecte.

Contribuţiile aduse în acest capitol sunt:

- Sistemul distribuit propus în acest capitol a fost analizat conform metodelor specifice sistemelor distribuite pornind de la elementele primordiale: *procesul*, *conexiunea dintre procese* şi *mesajul*. Au fost prezentate aspecte privind localizarea şi rolul proceselor într-un sistem distribuit.
- În cadrul modelului de proces au fost abstractizate componentele software şi nivelurile procesului. A fost modelată comunicaţia dintre componentele procesului folosind *evenimente*.
- Au fost analizate problemele care influenţează funcţionarea unui sistem distribuit: *problema acordului distribuit* şi *ordinea cauzală a mesajelor*.
- S-a propus şi analizat un anumit model de sistem distribuit în care mai multe procese separate au rol de execuţie a unor operaţii specifice, iar operaţiile de management referitoare la aceste procese de execuţie sunt centralizate într-un proces de management. Astfel s-au separat logic preocupările legate de aceste două *categorii de operaţii: management şi execuţie*.
- În modelul propus, administrarea şi dezvoltarea separată a celor două categorii de procese sunt avantaje aduse de acest sistem distribuit. Ulterior, separarea fizică a procesului de management şi a celor de execuţie aduce alt avantaj, respectiv, posibilitatea de a folosi puterea de procesare a mai multor calculatoare.
- În sistemul distribuit propus s-a prezentat modul cum sunt abstractizate *comunicaţiile dintre procesul client şi procesul manager*, precum şi *comunicaţiile dintre procesul manager şi procesele executant* din cadrul sistemului.

5. MODEL DE REPLICARE CU SEPARAREA MANAGEMENTULUI DE EXECUŢIE

5.1. INTRODUCERE

În cele ce urmează se va propune şi se va analiza un sistem distribuit cu o arhitectură orientată pe servicii şi mesaje, în care unele componente au rol executiv şi sunt orientate pe sarcini, în timp ce altele sunt orientate pe date şi au rol de asigurare a persistenţei datelor.

Sistemul acceptă *cereri* (solicitarea unor operaţii) din partea unor *aplicaţii client*. Aceste cereri (engl. *requests*) sunt recepţionate în punctul de intrare în sistem şi apoi sunt înregistrate, primind un identificator unic. Clienţii şi serviciile oferite de sistem comunică prin *mesaje*. Aplicaţiile-client şi aplicaţiile-server rulează pe calculatoare diferite, dar conectate în reţea. Aceste aplicaţii trebuie să comunice între ele indiferent de sistemele de operare.

În cadrul sistemului, componentele executive comunică între ele prin mesaje, atunci când rulează pe calculatoare diferite, sau prin apeluri de metode, atunci când rulează pe acelaşi sistem de operare.

Toleranţa la erori este asigurată, în esenţă, de replicarea datelor pe mai multe noduri de reţea. Se consideră acceptat modelul de sistem prezentat în [15]. La nivelul sistemului distribuit sunt luate în considerare atât căderile nodurilor reţelei, precum şi întreruperea legăturilor dintre noduri.

O serie de operaţii sunt absolut necesare pentru desfăşurarea în bune condiţii a replicării datelor, dar mai ales pentru resincronizarea replicilor în procesul de recuperare după întreruperea unor servicii datorită unor defecte fie în nodurile reţelei, fie la nivelul legăturilor dintre noduri.

Aceste operaţii considerăm că trebuie să cadă în sarcina unui subsistem de management, separat logic (uneori, şi fizic) de subsistemul de execuţie a operaţiilor normale determinate de interacţiunea sistemului software cu aplicaţiile client, cărora le oferă servicii. Sarcinile specifice acestui subsistem de management vor fi tratate pe larg în cele ce urmează.

Vom considera un sistem ale cărui componente nu trebuie să fie neapărat *sincronizate în timp*. Transmiterea de mesaje, invocarea metodelor, replicare operaţiilor şi resincronizarea replicilor nu depind de o sincronizare în timp. Sistemul păstrează o ordine strictă a cererilor venite din partea aplicaţiilor client. Pe fiecare subsistem-replică, cererile trebuie recepţionate exact în aceeaşi ordine.

Vom lua în considerare cazul sistemelor deterministe, deci, efectuarea aceloraşi operaţii în aceeaşi ordine vor duce la aceleaşi date stocate în sistemele care asigură persistenţa datelor. Recepţionarea cererilor, înregistrarea lor şi aşezarea lor într-o structură de tip FIFO constituie sarcini ale componentei de management.

În cadrul sistemului sunt definite componente cu rol de execuţie. Ele trebuie să primească cererile clienţilor exact în aceeaşi ordine în care au ajuns la intrarea în sistem şi să execute operaţii de citire şi scriere într-o ordine bine stabilită. Difuzarea listei de cereri din partea clienţilor, într-o ordine strictă, către componentele de execuţie este o altă sarcină a subsistemului de management.

Asigurarea persistenţei datelor după operaţii efectuate de serviciile sistemului cade în sarcina componentelor de execuţie. Pentru aceasta, în cadrul sistemului se va folosi o componentă complexă numită *WS-Resource* (resursă serviciu Web) care are, pe de o parte, rolul de procesare a datelor pentru implementarea unei logici de business, iar pe de altă parte, are şi rolul de stocare a datelor persistente. Datorită capabilităţii de stocare, componenta *WS-Resource* poate memora starea (deci este o componentă *stateful*).

5.2. OBIECTIVELE SISTEMULUI CU REPLICARE

Obiectivele majore urmărite în funcţionarea sistemului sunt:

- REPLICAREA OPERAŢIILOR.
  Sistemul trebuie să realizeze, în principal, *replicarea operaţiilor solicitate de clienţi pe mai multe servicii-replică*, păstrând ordinea strictă a solicitărilor.
- REPLICI DETERMINISTE.
  Serviciile replică trebuie să fie *deterministe*. Codul executat şi datele stocate trebuie să depindă numai de conţinutul cererii clientului şi să nu depindă de vreun parametru specific serverului-replică şi nici de timp (serviciile replică trebuie să fie deterministe).
- COMPONENTE EXECUTIVE CARE MEMOREAZĂ STAREA.
  Sistemul trebuie să conţină componente executive de tip *WS-Resource* care trebuie să fie în acelaşi timp atât servicii Web (*stateless*), acceptând cereri şi executând operaţii, dar şi resurse de date, având capacitatea de a stoca şi de a regăsi datele care trebuie să fie persistente (*stateful*).
- ÎNREGISTRAREA CERERILOR CLIENŢILOR.
  Sistemul trebuie să conţină o componentă specială care recepţionează cererile clienţilor şi răspunde de înregistrarea acestora, *Request Logging*. Rolul acesteia este să asigure înregistrarea persistentă a cererilor primite de la clienţi, adică să creeze un identificator unic pentru fiecare cerere venită din partea clienţilor şi să stocheze fiecare cerere împreună cu identificatorul său într-un format de date accesibil pentru operaţii ulterioare desfăşurate de alte componente.
- RELUAREA CERERILOR.
  Sistemul trebuie să fie capabil să regăsească o anumită solicitare (cerere) a unui client (în datele stocate de *Request Logging*) şi să o trimită componentelor executive, pentru execuţie pe anumite servicii replică (*Request Replay*).
- LOCALIZAREA SERVICIILOR.
  Componentele executive, respectiv serviciile responsabile cu replicarea operaţiilor cerute de clienţi, se găsesc, în mod obişnuit, pe alte calculatoare decât cel al componentei care recepţionează cererile clienţilor. Ca un caz particular, o componentă executivă poate fi pe acelaşi calculator.

- VERIFICAREA STĂRII SERVICIILOR.
  Sistemul trebuie să efectueze verificarea periodică a serviciilor replică şi să identifice funcţionarea degradată sau nefuncţionarea unui serviciu.
- RESINCRONIZAREA SERVICIILOR.
  Sistemul trebuie fie capabil să declanşeze procedura de *resincronizare* a unui serviciu replică (sau a mai multor servicii), după o întrerupere din funcţionare. În procesul de pregătire a resincronizării (reactualizării) unui anumit serviciu replică, sistemul trebuie să fie capabil să determine operaţiile „pierdute" de serviciul respectiv şi să pregătească lista de operaţii şi ordinea lor strictă.
- REPLICARE OPTIMISTĂ.
  Sistemul trebuie să poată funcţiona chiar dacă nu toate serviciile replică sunt funcţionale pe baza ipotezei optimiste că operaţiile vor fi propagate ulterior pe toate serviciile replică. Trebuie să existe un mod de operare *degradat* în care se pot executa operaţii de citire şi scriere a datelor, iar pentru serviciile aflate în stare de nefuncţionare se fac verificări periodice, iar la momentul revenirii acestora în starea de funcţionare se va declanşa procedura de resincronizare. În cazul extrem în care doar un serviciu replică este funcţional, atunci se acceptă din partea clienţilor numai cereri de citire a datelor, iar celelalte replici, nefuncţionale în acel moment, se monitorizează individual şi, în momentul revenirii lor în starea de funcţionare, se declanşează *resincronizarea*.
- TRANSPARENŢĂ FAŢĂ DE CLIENT.
  Replicarea operaţiilor pe mai multe servicii trebuie să se efectueze fără participarea aplicaţiei client. Mai mult, sistemul trebuie să rezolve intern situaţiile în care unele replici devin nefuncţionale, precum şi operaţiile de resincronizare a replicilor, fără să fie necesară informarea sau participarea aplicaţia client. În caz contrar, pentru a participa la procesele din sistemul cu replicare, ar fi necesară modificarea codului şi recompilarea aplicaţiei client.

În Fig. 5.1 se prezintă un tablou general cu cerinţele funcţionale ale sistemului analizat, precum şi succesiunea logică a operaţiilor principale.

5.3. COMPONENTELE SISTEMULUI CU REPLICARE

Analizând obiectivele sistemului se constată că sunt două mari *categorii de operaţii*:

- Operaţii executive de replicare propriu-zisă;
- Operaţii pregătitoare pentru replicarea propriu-zisă, pentru determinarea stării serviciilor replică, pentru resincronizare etc.

Ca urmare, în spiritul principiului separării preocupărilor, considerăm necesare în sistemul propus două categorii de componente: *executive* şi *de management*.

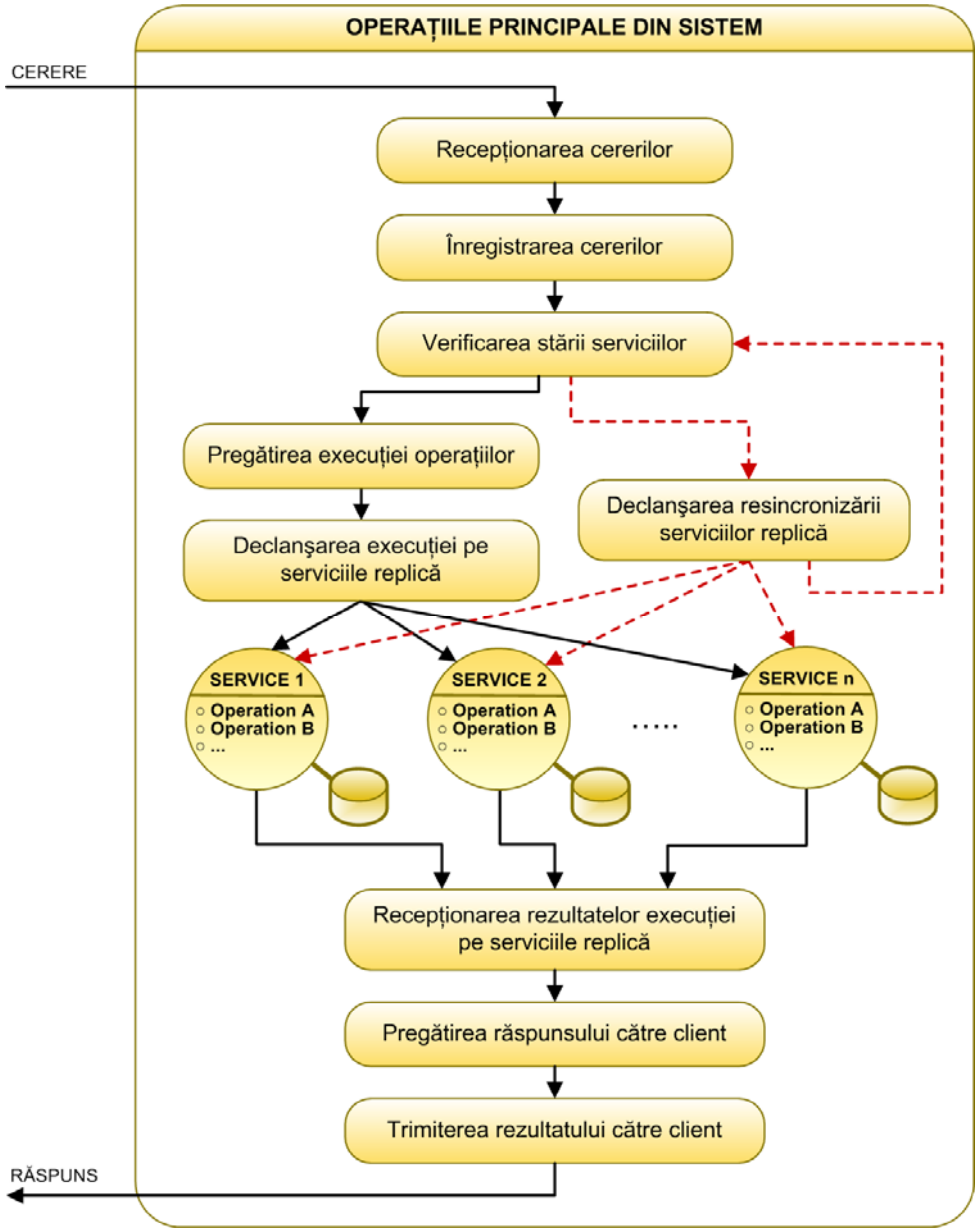

Fig. 5.1. Succesiunea operaţiilor principale din sistem

În primul rând, în sistemul analizat operaţiile executive sunt efectuate de *componentele executive* ale sistemului.

Procesele de replicare propriu-zisă se efectuează pe maşini separate, fără să fie nevoie ca aceste procese să aibă cunoştinţă unele de altele. Sistemele de stocare ataşate serviciilor replică sunt, de asemenea, pe maşini separate.

Avantajele separării serviciilor replică sunt:

- Posibilitatea de a folosi la maxim puterea de procesare a maşinii respective, independent de celelalte procese de replicare.
- Posibilitatea de a avea aceleaşi date stocate în diferite noduri ale reţelei (sau chiar în Internet) ceea ce permite, ulterior, o optimizare a apelurilor de tip READ către replica cea mai apropiată de locaţia clientului.
- Administrarea, salvările de siguranţă şi depanarea sistemelor replică se distribuie între mai mulţi specialişti aflaţi în locaţii diferite (repornirea sistemelor după căderi de tensiune, defecte hardware etc.).

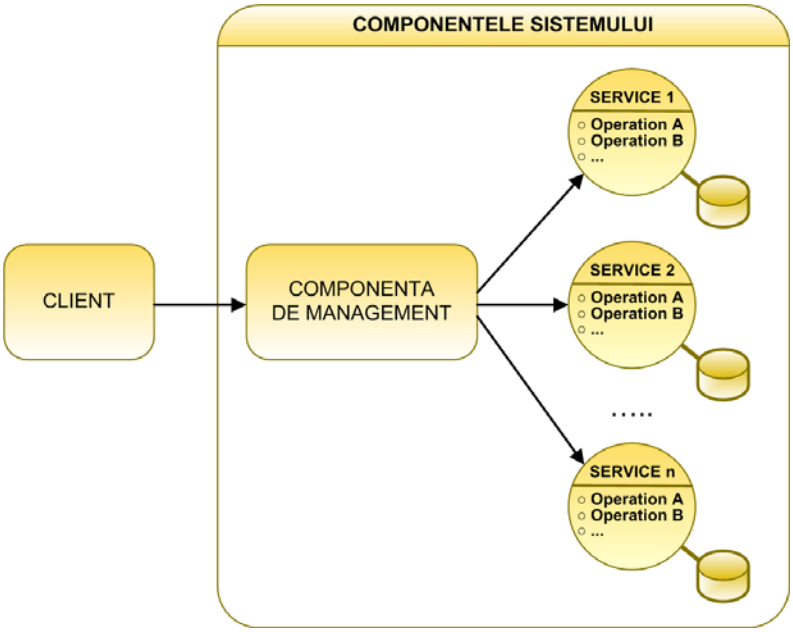

Fig. 5.2. Componentele principale ale sistemului

În al doilea rând, în sistemul analizat trebuie efectuate multe operaţii pregătitoare pe lângă cele de execuţie, iar aceste operaţii pregătitoare conduc la coordonarea şi controlul operaţiilor de execuţie. Ca urmare, alături de componentele de execuţie, pentru o coordonare unitară, sistemul trebuie să conţină o *componentă de management*.

Mai mult, pentru a separa efortul de procesare în aceste multiple operaţii pregătitoare de efortul de procesare necesar componentelor de execuţie, în spiritul principiului separării preocupărilor, considerăm necesară *separarea logică şi fizică a componentei de management de componentele de execuţie*.

5.4. MODURI DE OPERARE

Toate sistemele care implementează scheme de replicare au un aşa numit *mod normal de funcţionare*, în care toate replicile sunt active, sunt gata să răspundă cererilor clienţilor, iar datele persistente sunt aceleaşi pe toate replicile.

Dacă cel puţin un serviciu replică încetează să funcţioneze (ceea ce înseamnă un defect într-un nod al reţelei) sau nu mai este apelabil (ceea ce înseamnă o întrerupere a legăturii către acel serviciu), atunci sistemul trece într-un *mod de funcţionare degradată* sau chiar intră într-un *mod de blocare* a întregii activităţi.

Sistemele care au la bază scheme de replicare de tip *Primary-Backup*, în cazul în care serviciul principal nu este funcţional sau legătura către el nu este funcţională, intră într-o *stare de blocare* [11]. Sistemele care se bazează pe consensul unui cvorum de replici, dacă nu se atinge cvorumul necesar, intră într-un *mod de operare degradat* [43].

În practică, unele sisteme distribuite nu necesită în orice moment o consistenţă a datelor pe toate replicile. De aceea, aceste sisteme pot funcţiona chiar dacă, temporar, au fost relaxate restricţiile privind consistenţa datelor pe toate replicile din sistem. După o asemenea relaxare, există o fază de *resincronizare* (sau *reconciliere*) a tuturor replicilor.

În modelul de sistem propus în această lucrare se disting următoarele moduri specifice de funcţionare:

- *Modul de operare normală*, când replicile sunt sincronizate şi funcţionale
- *Modul de operare degradată, cu replicare parţială*, când replicarea este parţială, deoarece unele replici nu sunt funcţionale. Se consideră necesare minim 2 replici.
- *Modul de operare minimală, fără replicare*, când o singură replică a serviciului respectiv este funcţională şi sunt permise doar operaţii de citire a datelor existente până la repunerea în funcţiune a celorlalte replici (măcar una din ele).
- *Modul de resincronizare*, când cererile de la clienţi sunt doar înregistrate la nivelul componentei de management, dar nu sunt trimise spre execuţie pe replicile sistemului pentru că acestea rulează proceduri de resincronizare la nivelul datelor şi serviciilor. Când sistemul este în acest mod, clienţii sunt în aşteptare până când sunt sincronizate şi funcţionale un număr minim de replici (sincronizare parţială), sau toate (sincronizare totală). Numărul minim de replici necesare reluării operaţiilor în sistem este un parametru al funcţionării sistemului. Ideal este ca, în urma resincronizării, să se ajungă la starea normală de funcţionare, cu toate serviciile replicate sincronizate şi active. Dar, uneori, este posibilă revenirea doar la modul de operare cu replicare parţială.

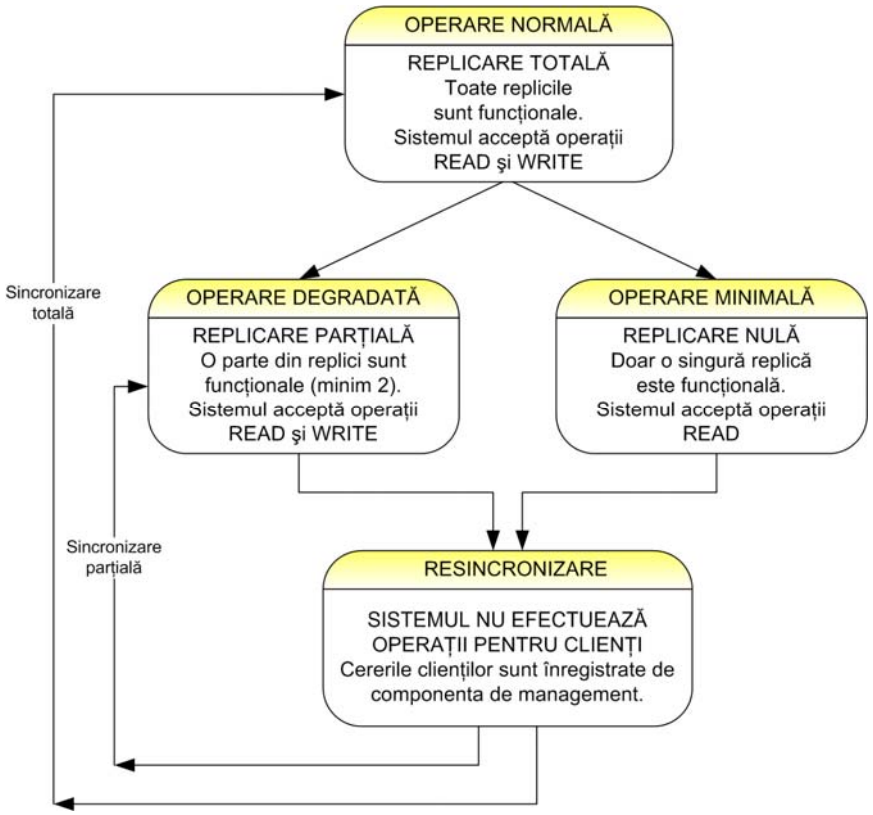

Fig. 5.3. Diagrama stărilor sistemului şi a tranziţiilor dintre stări

5.5. FAZE DE EXECUŢIE ŞI PROTOCOALE

O cerere venită din partea unei aplicaţii client, sub forma unui mesaj, determină o serie de operaţii pe partea serviciilor şi, în final, un mesaj de răspuns. Considerăm, pentru început, *funcţionarea normală a sistemului*, în care toate subsistemele de replicare sunt funcţionale şi sincronizate. Ulterior se va trata cazul *resincronizării*, după o întrerupere din funcţionare.

La nivelul serviciilor atomice, care nu se folosesc de alte servicii, ci doar de sisteme de stocare a datelor, se identifică următoarele *faze de execuţie*:

- REQ – *Recepţionarea cererii din partea aplicaţiei client (REQuest)*.
  Funcţionarea normală a aplicaţiei client şi a reţelei determină recepţionarea de către server a unui mesaj SOAP care conţine o cerere (engl. *request*). În esenţă, este indicată o anumită acţiune/operaţie care trebuie desfăşurată pe partea serverului. După modul în care influenţează datele persistente, aceste operaţii pot fi împărţite în două mari categorii: de citire, numite GET (sau READ), şi, respectiv, de scriere: ADDNEW, MODIFY, REMOVE (sau: INSERT, UPDATE, DELETE).

- LOG – *Înregistrarea cererii clientului (LOGging)*.
  Se creează un identificator unic al cererii (RQID), apoi cererea este serializată, iar tuplul {RQID, REQUEST} se păstrează într-un sistem de stocare, uzual într-o bază de date. Scopul este regăsirea ulterioară a unei anumite cereri, folosind identificatorul unic RQID.
- STA – *Verificarea stării serviciilor replică (STAtus).*
  În această fază, componenta de management interoghează serviciile pentru a verifica dacă sunt gata de a executa o operaţie (READY) şi care este identificatorul ultimei operaţii efectuate cu succes pe serviciul respectiv.
- INI – *Iniţializarea (pregătirea) procesului de replicare (INItialization).*
  Această fază este mai complexă, determină mai multe decizii la nivelul componentei de management şi va fi descrisă pe larg mai jos.
  Pe scurt, componenta de management a sistemului determină starea curentă a fiecărui serviciu replică (ex. READY, BUSY etc.) şi apoi decide dacă se poate continua cu replicarea cererii curente pe toate serviciile replică (*replicare totală*) sau se vor folosi mai puţine servicii-replică (*replicare parţială*), deoarece celelalte sunt fie ocupate (în starea BUSY), fie nefuncţionale. Pentru aceste servicii care „rămân în urmă" se va declanşa ulterior procesul de resincronizare, astfel încât, pe toate serviciile replică, să fie executate aceleaşi operaţii, până la ultima, în aceeaşi ordine, şi datele persistente de pe fiecare replică să fie identice.
- EXE – *Executarea operaţiilor determinate de cererea clientului (EXEcution)*.
  Pe fiecare replică apelată de componenta de management se tratează cererea clientului, identificată printr-un RQID unic. Operaţiile efectuate determină citirea sau modificarea unor date persistente.
- RPR – *Pregătirea mesajului de răspuns (Response PReparation)*.
  Componenta de management preia răspunsurile date de fiecare serviciu replică activ şi, în urma aplicării unei politici de tratare a răspunsurilor replicilor (referitoare la viteză, redundanţă etc.), ajunge la un mesaj de răspuns către client.
- RSP – *Trimiterea mesajului de răspuns către client (ReSPonse)*.

Analizând fazele enunţate mai sus se constată că replicarea efectivă are loc doar în faza notată EXE din funcţionarea sistemului în întregul său. În această fază sunt implicate componentele de execuţie din cadrul sistemului.

Toate celelalte faze, deşi nu se referă la replicarea propriu-zisă, totuşi sunt extrem de importante pentru că pregătesc condiţiile pentru buna desfăşurare a proceselor din sistem şi, efectiv, conduc la îndeplinirea obiectivelor de management definite anterior. Ele se desfăşoară la nivelul componentei de management din cadrul sistemului.

În continuare, componenta de management din cadrul sistemului va fi denumită *managerul serviciilor Web* şi va fi notată *WS Manager*, iar componentele executive vor fi denumite *Worker 1*, *Worker 2* etc.

5.5.1. PROTOCOLUL COMPONENTEI DE EXECUŢIE

În sistemul propus există mai multe componente de execuţie care sunt, practic, replicile serviciilor oferite. Sunt notate *Worker 1*, *Worker 2* … *Worker n*, unde n este numărul total de replici din sistem.

Sistemul considerat nu promovează neapărat o replică principală, aşa cum există în schemele de replicare de tipul Primary-Backup. Replicile sistemului sunt independente între ele şi „nu au cunoştinţă unele de altele". Deoarece unele servicii-replică vor fi, la un moment dat, nefuncţionale şi nu vor efectua ultimele operaţii solicitate, putem considera ca important pentru resincronizarea sistemului unul din serviciile-replică ce a funcţionat normal şi a executat toate operaţiile până la momentul respectiv. Acesta poate fi numit *Primary*. Resincronizarea serviciilor care au revenit în funcţionare după o întrerupere foloseşte lista de operaţii a acestui serviciu *Primary* pentru a determina, pentru fiecare serviciu care trebuie resincronizat, lista de operaţii pierdute ce trebuie efectuate pentru a ajunge la o stare de sincronizare între toate serviciile replică.

Fiecare componentă executivă din sistem este controlată de componenta de management notată *WS Manager*. Aceasta solicită componentelor de execuţie executarea unor operaţii, în urma cărora preia răspunsuri. La nivelul sistemului se poate defini o politică de prelucrare a acestor răspunsuri, care să balanseze sistemul către *redundanţă* sau către *viteză*.

În mod obişnuit, executarea operaţiilor conduce la modificarea datelor persistente păstrate în sisteme de stocare a datelor, uzual, în baze de date.

Aplicaţia client care a trimis o cerere către sistem primeşte de la WS Manager un răspuns pregătit într-o etapă distinctă de pregătire a răspunsului către client, etapă în care se aplică politica locală a sistemului de tratare a răspunsurilor venite din partea serviciilor replică. De exemplu, într-o primă variantă, se poate trimite clientului primul răspuns venit din partea serviciilor replică sau, varianta a doua, se aşteaptă răspuns de la toate serviciile replică, se analizează aceste răspunsuri şi apoi se ia o decizie de formare a unui răspuns către client.

Componenta de execuţie se poată găsi în una din stările următoare: READY, BUSY. Starea fiecărei componente de execuţie (starea fiecărui serviciu replică) este monitorizată de WS Manager şi notată într-o structură de date specială, WS-STATUS, existentă la nivelul componentei de management. În WS-STATUS, starea unui serviciu apare: READY, BUSY sau FAULT, ultimul caz fiind cel în care serviciul nu răspunde la interogările componentei de management, având un anumit defect.

Toate cererile venite din partea componentei WS Manager în faza REQ sunt înregistrate într-un sistem local de înregistrare a cererilor, la nivelul fiecărui serviciu replică. Foarte important este faptul că identificatorul fiecărei cereri este primit de la WS Manager şi este unic în întreg sistemul, pentru cererea respectivă.

Componenta de tip Worker trebuie să poată raporta în orice moment care este identificatorul, notat uzual LAST_RQID, al ultimei operaţii executate cu succes.

Protocolul pentru componenta de tip Worker precizează fazele şi regulile după care se desfăşoară operaţiile în cadrul fiecărui serviciu replică (Fig. 5.4).

Fig. 5.4. Protocolul de operare normală a unui serviciu Worker

Serviciile Worker din sistem funcţionează după următoarele *faze de execuţie*:

- REQ – *Recepţionarea cererii din partea aplicaţiei client (REQuest)*.
  Serviciul Worker recepţionează un mesaj trimis de WS Manager în care se solicită efectuarea unei *operaţii*. După modul în care influenţează datele persistente, aceste operaţii pot fi împărţite în două mari categorii: de citire (numite de exemplu GET sau READ), şi, respectiv, de scriere (numite de exemplu ADD, MODIFY, REMOVE sau INSERT, UPDATE, DELETE).
  Este foarte important formatul acestui mesaj primit pe reţea de la WS Manager, care rulează pe o altă gazdă. Este un mesaj SOAP. În corpul mesajului este indicată o operaţie pe care toate serviciile Worker o au precizată în interfaţă.
  În sistemul propus, considerăm că este necesar ca WS Manager să transmită serviciului Worker şi identificatorul RQID al cererii făcute de aplicaţia client, iar această informaţie trebuie stocată ca dată persistentă la nivelul serviciului Worker, pentru raportări ulterioare şi resincronizări.
- LOG – *Înregistrarea cererii clientului ca informaţie persistentă (LOGging)*.
  La nivelul serviciului Worker, identificatorul unic al cererii făcute de client (RQID) împreună cu cererea propriu-zisă serializată formează un tuplu {RQID, REQUEST} care se păstrează în sistemul local de stocare, uzual într-o bază de date. Astfel este posibilă regăsirea ulterioară a unei anumite cereri, raportarea, la solicitarea componentei WS Manager, a identificatorului RQID al ultimei operaţii efectuate cu succes, toate acestea servind, practic, la menţinerea sincronizării cu lista de cereri existentă la nivelul componentei WS Manager şi, la nevoie, la resincronizarea serviciilor.
- INI – *Iniţializarea (pregătirea) procesului de replicare (INItialization).*
  În această fază, serviciul Worker foloseşte operaţia primită în format
  serializat şi îşi construieşte, la nivel local, comanda potrivită pentru apelarea unei anumite operaţii. Tot în această fază, în cazul comenzilor de citire şi scriere pe baza de date, se deschid conexiunile necesare către bazele de date.
- EXE – *Executarea operaţiilor determinate de cererea clientului (EXEcution)*.
  În această fază se efectuează operaţia solicitată de WS Manager, identificată printr-un RQID unic. Operaţiile efectuate determină citirea sau modificarea unor date persistente. Se lucrează pe bază de tranzacţii.
- RPR – *Pregătirea mesajului de răspuns (Response PReparation)*.
  În această fază serviciul Worker preia datele primite de la bazele de date, notează faptul că operaţia s-a desfăşurat cu succes şi pregăteşte mesajul de răspuns către WS Manager, respectând formatul datelor din *DataContract*.
- RSP – *Trimiterea mesajului de răspuns către WS Manager (ReSPonse)*.

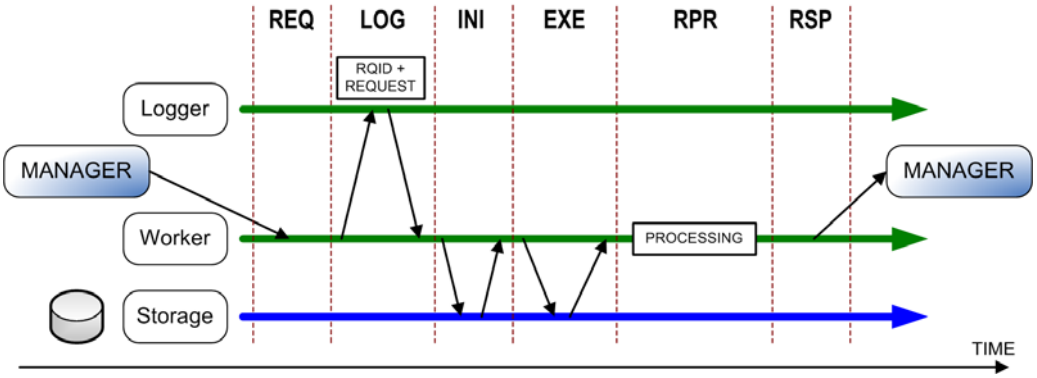

Fig. 5.5. Detalii despre protocolul de operare normală a unui serviciu Worker

5.5.2. PROTOCOLUL COMPONENTEI DE MANAGEMENT PENTRU OPERAREA NORMALĂ A SISTEMULUI

În modul de funcţionare, după cum s-a arătat mai sus, toate serviciile replică sunt funcţionale şi în starea de aşteptare a cererilor (READY). Protocolul precizează fazele şi regulile după care se desfăşoară operaţiile în cadrul sistemului.

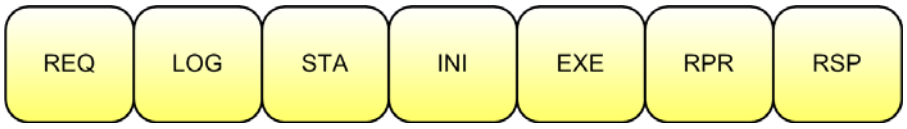

Fig. 5.6. Protocolul pentru modul de operare normală a componentei Manager

Componenta WS Manager funcţionează după următoarele *faze de execuţie*:

- REQ – *Recepţionarea cererii din partea aplicaţiei client (REQuest)*.
  Funcţionarea normală a aplicaţiei client şi a reţelei determină recepţionarea de către server a unui mesaj SOAP care conţine o cerere (engl. *request*). În esenţă, este indicată o anumită acţiune/operaţie care trebuie desfăşurată pe partea serverului. După modul în care influenţează datele persistente, aceste operaţii pot fi împărţite în două mari categorii: *de citire*, numite GET (READ), şi, respectiv, *de scriere*: ADDNEW, MODIFY, REMOVE.
- LOG – *Înregistrarea cererii clientului ca informaţie persistentă (LOGging)*.
  Se creează un identificator unic al cererii (RQID), apoi cererea este serializată, iar tuplul {RQID, REQUEST} se păstrează într-un sistem de stocare, uzual într-o bază de date. Scopul este regăsirea ulterioară a unei cereri, folosind identificatorul RQID.

- STA – *Verificarea stării serviciilor replică (STAtus).*
  În această fază, componenta de management interoghează serviciile pentru a verifica dacă sunt gata de a executa o operaţie (READY) şi care este identificatorul ultimei operaţii efectuate cu succes pe serviciul respectiv.
- INI – *Iniţializarea (pregătirea) procesului de replicare (INItialization).*
  Această fază este mai complexă, determină mai multe decizii la nivelul componentei de management şi va fi descrisă pe larg mai jos. Pe scurt, componenta de management a sistemului determină starea curentă a fiecărui serviciu replică (ex. READY, BUSY etc.) şi apoi decide dacă se poate continua cu replicarea cererii curente pe toate serviciile replică (*replicare totală*) sau se vor folosi mai puţine servicii-replică (*replicare parţială*), deoarece celelalte sunt fie ocupate (în starea BUSY), fie nefuncţionale. Pentru aceste servicii care „rămân în urmă" se va declanşa ulterior procesul de resincronizare, astfel încât, pe toate serviciile replică, să fie executate aceleaşi operaţii, până la ultima, în aceeaşi ordine, şi datele persistente de pe fiecare replică să fie identice.
- EXE – *Executarea operaţiilor cerute de client (EXEcution)*.
  Pe fiecare replică apelată de componenta de management se tratează cererea clientului, identificată printr-un RQID unic. Operaţiile efectuate determină citirea sau modificarea unor date persistente.
- RPR – *Pregătirea mesajului de răspuns (Response PReparation)*.
  Componenta de management preia răspunsurile date de fiecare serviciu replică activ şi pregăteşte un mesaj de răspuns către client într-un format corespunzător, conform contractului de date. Deoarece serviciile replică rulează, în mod normal, pe alte gazde din reţea, mesajele de răspuns sunt mesaje SOAP. Elementele necesare pregătirii unui răspuns către client se găsesc în interiorul acestor mesaje SOAP.
  În această fază este posibilă aplicarea unei politici (referitoare la viteză, redundanţă etc.) de tratare a răspunsurilor primite de la serviciile replică.
- RSP – *Trimiterea mesajului de răspuns către client (ReSPonse)*.
  WS Manager transmite aplicaţiei client răspunsul corespunzător cererii făcute anterior de aceasta. În cazul când sistemul nu este în starea de funcţionare normală sau, măcar, în starea degradată, datorită nefuncţionării unui număr inadmisibil de mare de servicii replică, WS Manager transmite un mesaj *„Operaţia solicitată nu poate fi efectuată în acest moment*".

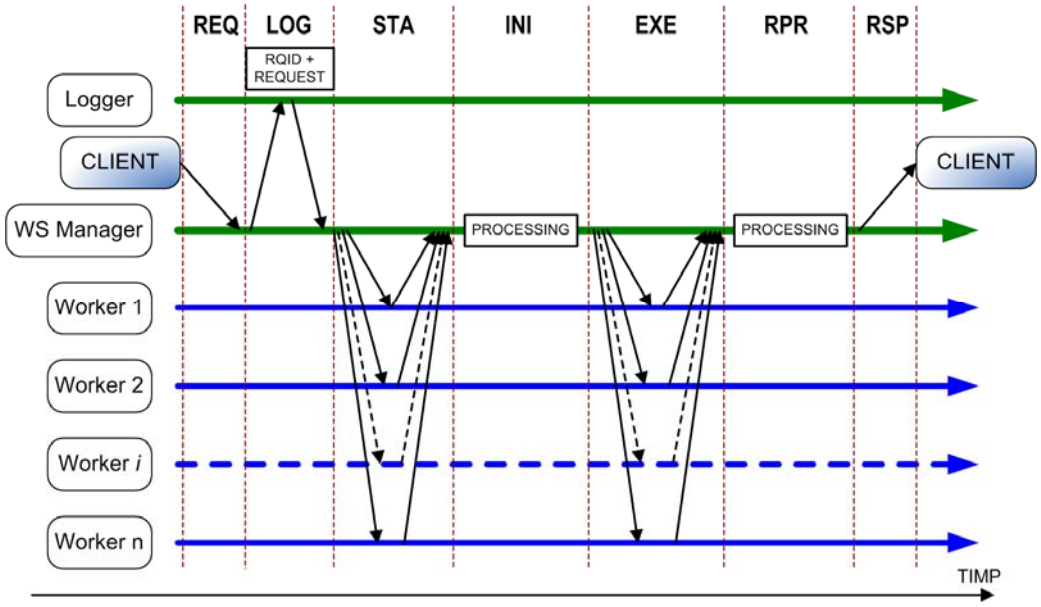

Fig. 5.7. Detalii despre protocolul de operare normală a componentei Manager

5.5.3. PROTOCOLUL COMPONENTEI DE MANAGEMENT PENTRU OPERAREA DEGRADATĂ A SISTEMULUI

În modul de funcţionare, după cum s-a arătat mai sus, unele din serviciile replică nu sunt funcţionale sau se găsesc în starea BUSY, deoarece nu au terminat operaţia anterioară. Dacă WS Manager determină că sunt funcţionale şi gata de lucru minim 2 servicii replică, atunci consideră că poate opera într-un mod de lucru degradat, deservind totuşi clientul. În acelaşi timp, WS Manager notează care sunt serviciile replică cu care nu a putut lucra în momentul respectiv. Aceste servicii vor fi „lăsate în urmă" şi ulterior, printr-un proces separat, independent de clienţi, vor fi sincronizate cu celelalte servicii care nu au fost blocate.

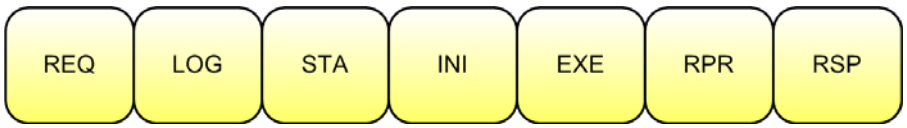

Fig. 5.8. Protocolul de operare degradată a componentei Manager

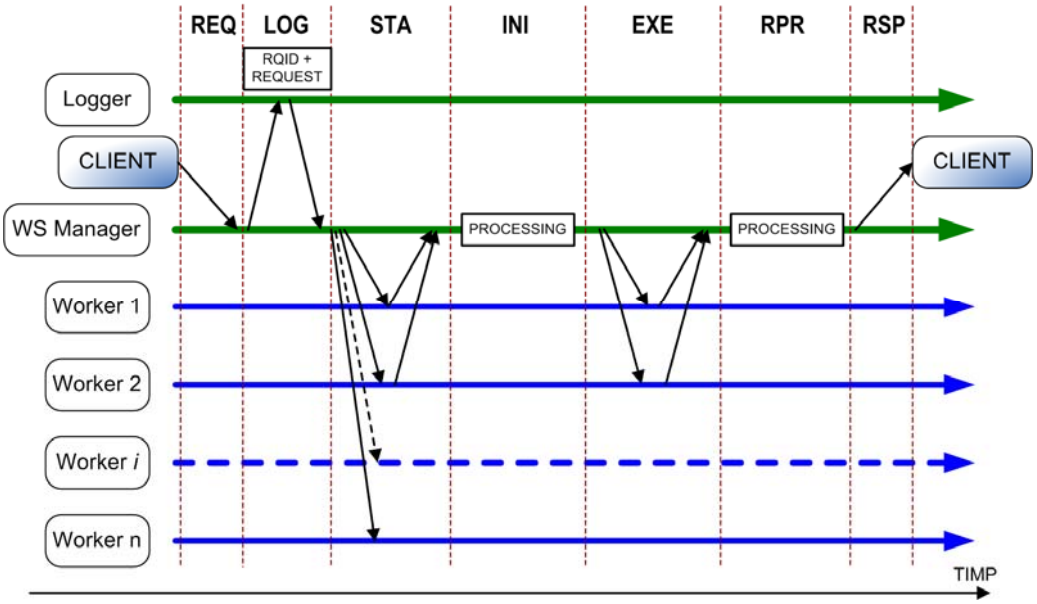

Fig. 5.9. Detalii despre protocolul de operare degradată pentru Manager

5.5.4. PROTOCOLUL COMPONENTEI DE MANAGEMENT PENTRU OPERAREA MINIMALĂ A SISTEMULUI

În modul de funcţionare, după cum s-a arătat mai sus, unele din serviciile replică nu sunt funcţionale sau se găsesc în starea BUSY, deoarece nu au terminat operaţia anterioară. Dacă WS Manager determină că numai o singură replică este funcţională şi gata de lucru, atunci consideră că trebuie să opereze într-un mod de lucru minimal, deservind numai operaţii de tip READ din partea clientului. Această decizie se ia la sfârşitul fazei INI şi influenţează decisiv faza EXE pentru că nu se vor mai executa decât operaţii de citire a datelor existente.

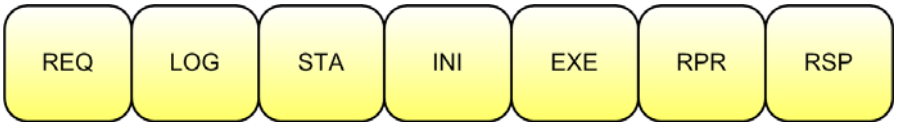

Fig. 5.10. Protocolul de operare minimală a componentei Manager

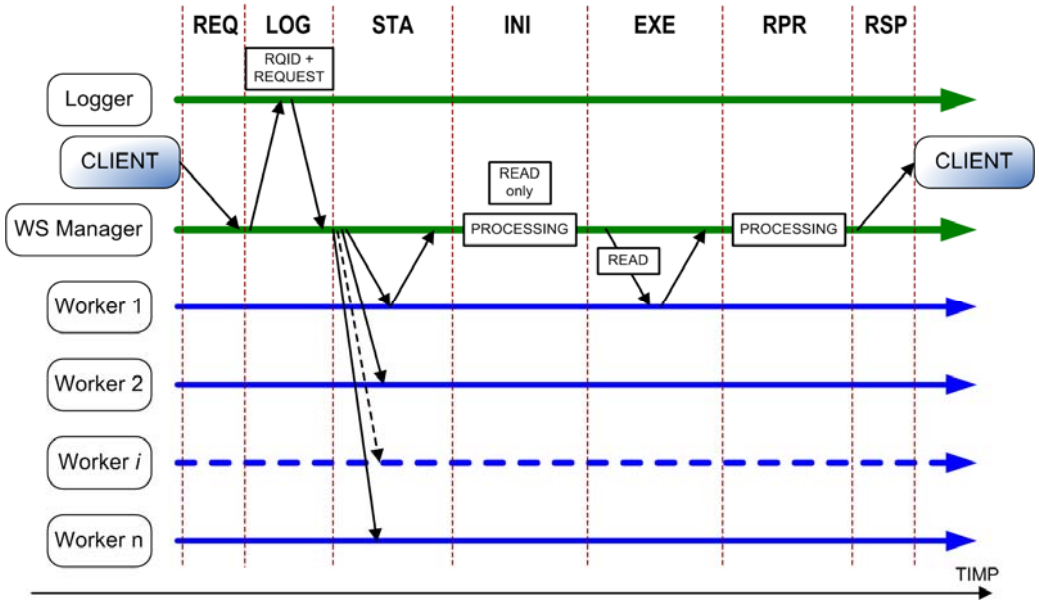

Fig. 5.11. Detalii despre protocolul de operare minimală pentru Manager

WS Manager declanşează procesul de resincronizare a serviciilor care au fost blocate sau au fost lăsate în urmă (pentru că erau în starea BUSY şi s-a continuat într-un mod de operare degradată) cu serviciul care a funcţionat până a ajuns singurul funcţional (în figura Fig. 5.11 este prezentat cazul serviciului Worker 1).

5.5.5. PROTOCOLUL PENTRU RESINCRONIZAREA SERVICIILOR

Resincronizarea se face pentru fiecare serviciu revenit în starea de funcţionare după o întrerupere a legăturii de reţea sau o cădere a nodului de reţea.

Mai întâi se determină, la momentul respectiv, identificatorul LAST_RQID al ultimei cereri pentru o operaţie de scriere a datelor, efectuată cu succes pe minim 2 servicii replică, respectiv cele care au funcţionat când sistemul a fost în modul de operare degradată.

Toate celelalte replici trebuie să primească spre execuţie operaţiile de scriere de date pe care nu le-au efectuat, în ordinea strictă în care au fost înregistrate de componenta de înregistrare a cererilor, până ajung la ultima cerere identificată prin LAST_RQID.

Resincronizarea serviciilor replică este un proces cu următoarele *faze*:

- LREQ – *Determinarea ultimei cererii de scriere efectuată de sistem*.
  WS Manager determină, la momentul respectiv, identificatorul LAST_RQID al ultimei cereri pentru o operaţie de scriere a datelor, efectuată cu succes pe toate serviciile replică rămase active (cele care au funcţionat când sistemul a fost în modul de operare normală) sau pe minim 2 servicii replică (cele care au funcţionat când sistemul a fost în modul de operare degradată). Operaţiile de tip citire de date solicitate de clienţi nu au semnificaţie în procesul de resincronizare.

- PREP – WS Manager determină, pentru fiecare serviciu replică cu care a reluat legătura după o stare FAULT, lista de operaţii care trebuie efectuate pentru a ajunge la ultima solicitare (LAST_RQID), respectiv la starea de sincronizare a serviciilor replică. Informaţiile necesare sunt luate din sistemul de înregistrare a cererilor clienţilor folosit de WS Manager.
- SYNC – Fiecare serviciu WORKER primeşte pentru execuţie operaţiile din lista pregătită anterior de WS Manager. Operaţiile se execută exact în ordinea în care au fost înregistrate.
- CHK – La finalul executării listei de operaţiuni, fiecare serviciu Worker trebuie să aibă executate aceleaşi operaţii cerute de clienţi, exact în aceeaşi ordine, pornind de la o stare de sincronizare anterioară.
  Pe baza faptului că serviciile Worker nu execută nici o modificare a datelor persistente fără o comandă din partea componentei WS Manager şi, în plus, pe baza ipotezei iniţiale că serviciile Worker din sistemul propus sunt deterministe, se poate trage concluzia că, la finalul unei resincronizări reuşite (la nivelul serviciilor), datele persistente existente în sistemele de stocare utilizate de fiecare serviciu Worker sunt aceleaşi. În cadrul acestei faze, CHK, dacă se constată că totuşi, pe un serviciu replică, datele persistente diferă de datele de pe un serviciu care nu a fost în starea FAULT, trebuie să se poată declanşa operaţii de sincronizare a datelor la nivelul bazelor de date, nu numai la nivelul serviciilor, cum a fost cazul descris mai sus.

Succesiunea fazelor menţionate mai sus este prezentată în Fig. 5.12.

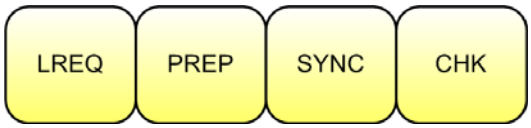

Fig. 5.12. Protocolul pentru resincronizare

5.6. CONCLUZII ŞI CONTRIBUŢII

Contribuţiile aduse în acest capitol sunt:

- S-a propus şi s-a analizat o arhitectură de sistem distribuit, tolerant la defecte. Sistemul are o arhitectură orientată pe servicii şi efectuează operaţii în urma unor *cereri* din partea unor *aplicaţii client*. Aceste cereri sunt înregistrate sub un identificator unic. Clienţii şi serviciile oferite de sistem comunică prin *mesaje*.
- S-au formulat *obiectivele ale sistemului*, care vor influenţa ulterior toată dezvoltarea sa, în ceea ce priveşte fiabilitatea sistemului.

- În sistemul propus toleranţa la defecte este asigurată, în esenţă, de replicarea datelor pe mai multe noduri de reţea. La nivelul sistemului distribuit au fost luate în considerare atât căderile nodurilor reţelei, cât şi întreruperea legăturilor dintre noduri.
- Pentru modelul de sistem propus au fost prezentate modurile de operare, precum şi tranziţiile între acestea.
- Dezvoltarea sistemului a condus la evidenţierea separată a *componentelor de management* (care formează un subsistem de management) şi a *componentelor cu rol de execuţie* (subsistemul de execuţie).
- Analiza modelului a evidenţiat modul în care componentele de execuţie trebuie să primească cererile clienţilor pentru a executa operaţii de citire şi scriere a datelor.
- În modelul de sistem propus asigurarea persistenţei datelor cade în sarcina componentelor de execuţie. Pentru aceasta, în cadrul sistemului s-a folosit o componentă complexă numită *WS-Resource* (resursă serviciu Web) care are, pe de o parte, rolul de procesare a datelor pentru implementarea unei logici de business, iar pe de altă parte, are şi rolul de stocare a datelor persistente. Este important de menţionat că, datorită funcţiei de stocare, componenta *WS-Resource* poate memora starea (este *stateful*).
- În modelul de sistem propus s-au evidenţiat o serie de operaţii necesare pentru desfăşurarea în bune condiţii a replicării datelor, dar mai ales pentru resincronizarea replicilor în procesul de recuperare după întreruperea unor servicii datorită unor defecte fie în nodurile reţelei, fie la nivelul legăturilor dintre noduri. S-a optat pentru varianta în care aceste operaţii cad în sarcina unui subsistem de management, separat de subsistemul format din componentele de execuţie amintite mai sus. În continuare au fost evidenţiate sarcinile specifice componentei de management.
- În modelul de replicare propus în acest capitol se păstrează o ordine strictă a cererilor venite din partea aplicaţiilor client. Pe fiecare subsistem-replică, cererile trebuie recepţionate exact în aceeaşi ordine. A fost considerat cazul sistemelor deterministe, deci, efectuarea aceloraşi operaţii în aceeaşi ordine vor duce la aceleaşi date stocate în sistemele care asigură persistenţa datelor.
- Pentru modelul de replicare propus s-au prezentat în detaliu protocoalele şi fazele de execuţie, respectiv: protocolul componentei de execuţie cu fazele de execuţie corespunzătoare şi protocolul componentei de management cu fazele de execuţie respective, diferenţiat pe modurile de operare ale sistemului.

Modelul de replicare prezentat în acest capitol stă la baza dezvoltării ulterioare în această teză a unui sistem fiabil cu management centralizat şi replicare coordonată şi separată.

6. ARHITECTURA UNUI SISTEM FIABIL CU REPLICARE MANAGERIATĂ

6.1. INTRODUCERE

În cele ce urmează se propune o arhitectură de sistem distribuit ce foloseşte servicii Web pentru deservirea clienţilor şi conţine module specializate pentru creşterea fiabilităţii.

Serviciile Web facilitează interacţiunea aplicaţie-aplicaţie având la baza transportului de date protocoalele Web. În acest context sunt foarte importante problemele de fiabilitate şi securitate. O privire de ansamblu a tehnicilor de realizare a serviciilor Web fiabile şi securizate este realizată de Moser în [41].

Disponibilitatea serviciilor de reţea poate fi crescută folosind diverse scheme care să dirijeze cererile sosite după un eşec al serviciului către un alt server funcţional. Dar asemenea scheme nu suportă recuperarea cererilor pentru care s-a început deja procesarea. Abordările publicate includ folosirea sistemului DNS pentru a şterge adresa serverelor nefuncţionale din cadrul listei de servicii [6], dar şi redirectarea clienţilor către servere alternative care sunt replici ale serverului folosit iniţial [52].

Există diferite scheme pentru servicii de reţea tolerante la erori, dar care nu sunt transparente faţă de aplicaţia-client. Acestea sunt numite generic *soluţii cu clienţi conştienţi* (engl. *client-aware solutions*). Unele din aceste soluţii necesită modificarea aplicaţiilor client, în timp ce altele necesită modificări la nivelul implementării TCP din nucleul sistemului de operare. Există şi o altă categorie de asemenea soluţii „client-aware" care propun implementarea unui sistem pe 3 niveluri (3-tier system) care să folosească un protocol cu 2 faze pentru a asigura realizarea exact o singură dată a tranzacţiilor şi, mai mult, recuperarea cererilor aflate în lucru [27].

Comunitatea programatorilor serviciilor Web a dezvoltat protocoale pentru transmiterea fiabilă a mesajelor la nivel de aplicaţie, protocoale care au la bază SOAP şi HTTP: *WS-Reliable Messaging* 2009 [17]. Dar aceste protocoale nu oferă transparenţă faţă de aplicaţia-client şi nici nu tratează subiecte importante cum ar fi persistenţa mesajelor şi recuperarea după eroare [41].

Un cadru de lucru pentru realizarea unor servicii Web tolerante la erori, bazate pe SOAP, este FT-SOAP. Acesta promovează o configuraţie cu un server principal şi unul de rezervă, iar un manager al replicărilor este capabil să promoveze serverul de rezervă la rolul de server principal. Dezavantajul este că aplicaţiile-client trebuie să fie „conştiente" de existenţa serverului principal şi a celui de rezervă şi, în plus, trebuie modificate într-un anumit mod pentru a fi capabile să redirecteze o cerere către un server de rezervă din sistem [21].

Arhitectura propusă în cele ce urmează este tolerantă la defecte, foloseşte replicarea serviciilor şi a datelor persistente şi este transparentă faţă de clienţi. Rezolvarea problemelor interne din sistem nu este vizibilă de către aplicaţiile client. Ea conţine mai multe servere grupate într-o unitate autonomă bazată pe servere şi servicii Web. Această unitate acceptă cereri din partea aplicaţiilor-client pe o

conexiune de reţea funcţională şi trimite mesaje corespunzătoare de răspuns. Acest grup de servere şi servicii Web rezolvă intern erorile de funcţionare când apar şi menţine o transparenţă faţă de client.

Această unitate este un sistem autonom care depune toate eforturile pentru trimiterea mesajului de răspuns la cererile primite. Autonomia se referă la faptul că procesele blocate pe partea serverelor şi chiar căderile serverelor nu afectează funcţionarea aplicaţiilor-client. De fapt, aplicaţiile client nici nu au cunoştinţă de eforturile depuse pe partea serverelor pentru livrarea unui serviciu corect, fapt exprimat de termenul „transparenţă faţă de client”.

Analiza începe de la arhitectura de sistem prezentată în figura de mai jos.

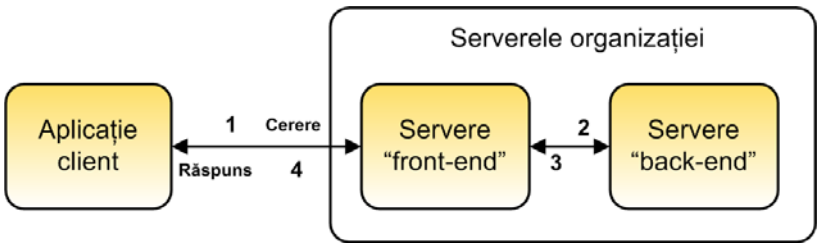

Fig. 6.1. Componentele sistemului distribuit.

Scopul principal urmărit în proiectarea sistemului este următorul: folosind o conexiune funcţională de reţea între aplicaţia client şi serverele companiei, clientul trebuie să primească un răspuns de la aplicaţiile server ale organizaţiei conform cererii făcute şi toate erorile de funcţionare care pot să apară în reţeaua organizaţiei trebuie remediate intern şi nu trebuie „văzute” de aplicaţia-client.

Pentru atingerea acestui scop trebuie implementate pe partea serverelor o serie de funcţionalităţi esenţiale: înregistrarea cererilor, planificarea şi coordonarea replicării, managementul erorilor, sincronizarea serviciilor, transparenţa proceselor interne faţă de client.

Arhitectura propusă în cele ce urmează conţine componente specifice care implementează funcţionalităţile esenţiale menţionate mai sus, precum şi altele necesare.

6.2. ARHITECTURA SISTEMULUI

Se urmăreşte realizarea unei arhitecturi tolerante la erori ce conţine mai multe servere grupate într-o unitate funcţională autonomă bazată pe servicii Web. Acest grupare de servere este văzută din exterior ca o singură entitate software capabilă să accepte conexiuni de la aplicaţii-client, să execute anumite operaţii interne, să trateze în interiorul arhitecturii erorile apărute, fără participare din partea aplicaţiei-client şi, în final, să răspundă în mod corespunzător solicitărilor.

Funcţionalităţile majore ale acestui grup de servere sunt următoarele: managementul replicării, managementul erorilor, înregistrarea cererilor şi sincronizare, transparenţă faţă de aplicaţiile client.

Arhitectura propusă extinde arhitectura serviciilor Web prin adăugarea unor noi componente în scopul de a forma o nouă unitate software care să fie, în primul rând, fiabilă şi, în al doilea rând, să menţină transparenţa faţă de client.

Aceste componente sunt: Managerul serviciilor Web (WS-Manager), înregistratorul stărilor serviciilor Web (WS-Status Logger), înregistratorul cererilor (Request Logger) şi routerul de mesaje (Message Router). Cererile clienţilor sunt recepţionate de managerul serviciilor Web, WS Manager în punctul de intrare în sistem, aşa cum se observă în Fig. 6.2.

WSM răspunde de detectarea defectelor de funcţionare şi recuperare după căderea serviciilor. Pentru a realiza această recuperare, managerul WSM are nevoie, în primul rând, de înregistratorul cererilor (Request Logger), care salvează local toate cererile primite de la clienţi, păstrând ordinea apariţiei acestora.

Fiecare cerere primeşte un identificator pentru a putea fi ulterior identificată fără confuzii. Cererile stocate vor fi folosite atunci când un subsistem trebuie readus în stare de funcţionare şi actualizat după o cădere.

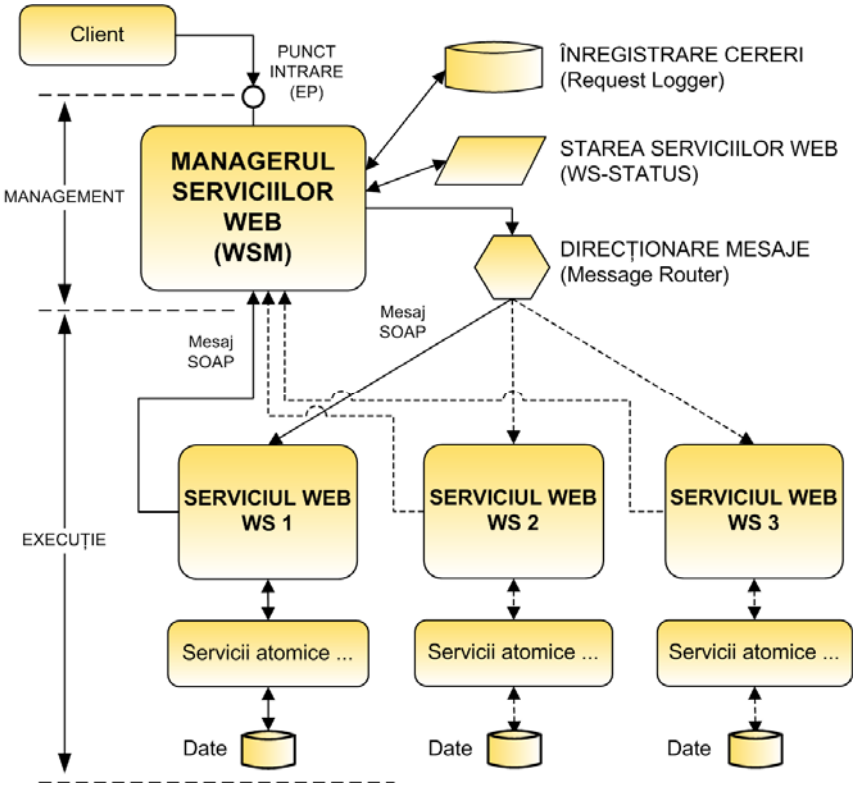

Fig. 6.2. Arhitectura propusă (cazul cu 3 servicii *worker*)

WSM acţionează ca un monitor al serviciilor Web. Pentru aceasta el menţine o structură de date numită WS-Status care conţine informaţii esenţiale despre starea serviciilor şi care se este mereu actualizată. In mod obişnuit, această actualizare se petrece atunci când un serviciu din lista de servicii WS1, WS2, etc. primeşte o sarcină de la WSM sau raportează că a terminat o sarcină .

După verificarea stării serviciilor cu rol de worker, Managerul Serviciilor Web, WSM, acţionează ca un dispecer. El foloseşte o componentă specializată numită Message Router pentru a redirecta cererea curentă către toate serviciile

Web, WS1… WSn. Dacă un serviciu nu este funcţional, WSM notează starea FAULT a serviciului respectiv în WS-Status. În modul normal de operare, fiecare cerere este trimisă tuturor serviciilor Web ale sistemului.

Din punct de vedere al operaţiilor de replicare, arhitectura propusă are 2 niveluri:

- Nivelul *Management*, care conţine Managerul de Servicii Web, WSM, înregistratorul cererilor (Request Logger), înregistratorul modificărilor de stare ale serviciilor (WS-STATUS Logger), precum şi componenta pentru redirectarea cererilor de la clienţi către serviciile-replică (Message Router/Dispatcher);
- Nivelul *Execuţie* (sau nivelul serviciilor), care conţine un număr de servicii Web compozite notate (WS 1, WS 2 ... WS n), şi un număr de servicii atomice care deservesc serviciile compozite şi operează cu baze de date (figura de mai sus).

6.3. IPOTEZE INIŢIALE

Se porneşte de la următoarele ipoteze:

- o conexiune de reţea funcţională este stabilită între aplicaţia client şi WSM şi cererile clienţilor ajung la punctul de intrare (EP – Entry Point);
- în interiorul sistemului, la nivelul serviciilor de pe nivelurile inferioare pot să apară defecte de funcţionare;
- fiecare cerere primeşte un identificator unic de la managerul WSM;
- WSM stochează local toate cererile şi identificatorii corespunzători;
- WSM este capabil să detecteze funcţionarea defectuoasă a serviciului principal (primary worker) şi, în această situaţie, să redirecteze cererea clientului către un serviciu de rezervă (backup worker);
- După o cădere a unui serviciu Web WSM foloseşte lista de cereri (fiecare cu un identificator unic) pentru a porni un proces de readucere a serviciului respectiv într-o stare corespunzătoare cu starea curentă a sistemului (în care sunt soluţionate toate cererile în ordinea strictă a apariţiei lor);
- Aplicaţia client a descoperit serviciul folosind protocolul UDDI şi registrul de servicii (Service Registry), a adresat cererea sa unui anumit serviciu şi primeşte un răspuns fără să aibă cunoştinţă de eventualele defecte de funcţionare, remedierea lor şi resincronizarea componentelor din interiorul serviciului respectiv privit ca un sistem tolerant la erori.

6.4. MODUL NORMAL DE OPERARE

Când o cerere ajunge la punctul de intrare (EP – Entry Point), Managerul Serviciilor Web, WSM, creează, mai întâi, un identificator de cererii (Request ID) pentru a identifica ulterior cu uşurinţă respectivul mesaj.

Mesajul cererii şi identificatorul ei sunt înregistrate de către WSM în structura de date WS-Status. La nevoie, această informaţie trebuie să poată fi regăsită în mod unic. Prin aceste operaţii, WSM implementează una din funcţionalităţile sistemului: jurnalizarea cererilor.

Structura de date WS-Status este menţinută de către Managerul Serviciilor Web, WSM. În modul normal de operare, la un moment dat, un serviciu worker poate fi în una din stările normale de funcţionare: READY sau BUSY.

Tabelul 6.1. Înregistrarea cererilor

Identificatorul cererii (Request ID)	Conţinutul mesajului
RQ20110315-0000001	... (XML code ...)
RQ20110315-0000002	... (XML code ...)
...	...

După faza de iniţializare a sistemului, fiecare serviciu worker este în starea READY. Această situaţie este reflectată de informaţiile din WS-Status. Când un serviciu worker primeşte o sarcină şi începe să lucreze, WSM notează starea BUSY pentru serviciul respectiv în WS-Status. Mai mult, WSM notează în WS-Status şi timpul de început al lucrului pentru rezolvarea sarcinii primite (coloana *Start Time*). Când serviciul worker a rezolvat sarcina primită, identificată în mod unic de un ID, managerul WSM notează în WS-Status timpul de execuţie pentru sarcina respectivă (coloana *Execution Time*). Apoi WSM notează starea READY pentru serviciul worker respectiv.

Funcţionarea sistemului se face sub controlul Managerului de Servicii Web, WSM, care citeşte şi scrie informaţii în WS-Status. Starea unui serviciu Web poate fi, la un moment dat: READY, BUSY (stări normale de funcţionare) sau FAULT (în caz de nefuncţionare).

În Tabelul 6.2 se prezintă o anumită situaţie. WSM trebuie să trimită o nouă sarcină, dar starea serviciilor worker este diferită. Situaţia din Tabelul 6.2 trebuie interpretată astfel:

- Sarcina anterioară (identificată prin valoarea lui RQID) a fost executată de workerul WS1, care a început lucrul la un anumit moment specificat în coloana *Start Time* şi l-a dus la bun sfârşit într-un timp de execuţie specificat în coloana *Execution Time*. Starea actuală a workerului WS1 este READY.
- Aceeaşi sarcină a fost începută de workerul WS2 la momentul specificat în coloana *Start Time*, dar WS2 în că nu a terminat lucrul. Astfel, *Execution Time* nu este disponibil (N/A – Non-Available), iar starea workerului WS2 este BUSY. La finalizarea lucrului, starea lui WS2 va deveni READY, iar durata de execuţie va fi notată în coloana *Execution Time*.
- Starea serviciului WS3 este FAULT, deci acesta nu poate executa noua sarcină.

Tabelul 6.2. Structura de date WS-Status

WS	RQ ID	STATUS	Start time	Execution time (ms)
WS1	RQ...010	READY	18:21:15,652	58
WS2	RQ...010	BUSY	18:22:35,083	N/A
WS3	RQ...010	FAULT		N/A

În orice moment al funcţionării normale, unul din serviciile worker, WS1, WS2 etc. este ales drept *serviciu principal* (*primary worker*). La iniţializarea sistemului, acest rol de serviciu principal este acordat serviciului WS1.

În cazul în care WSM transmite o cerere a unui client către WS1 sunt posibile două variante: fie starea serviciului WS1 este READY, fie starea sa este BUSY sau FAULT.

Dacă starea serviciului WS1 cu rol de Primary Worker este READY, atunci următoarele operaţii sunt executate la apariţia unei cereri din partea unei aplicaţii client:

1) WSM foloseşte subsistemul Message Router pentru a trimite cererea către serviciul principal WS1.
2) Serviciul principal WS1 trimite către WSM o confirmare de primire a cererii şi de începere a lucrului.
3) WSM înscrie în WS-Status noua stare a serviciului principal WS1, respectiv BUSY, precum şi timpul de început al lucrului (start time).
4) Serviciul WS1 rezolvă cererea şi trimite un răspuns către WSM.
5) WSM recepţionează datele de răspuns, notează faptul că serviciul principal a devenit disponibil prin scrierea stării READY pentru WS1 în tabelul WS-Status, precum şi timpul de execuţie.
6) WSM trimite mesajul de răspuns către aplicaţia-client.

Dacă starea serviciului principal WS1 este BUSY, atunci WSM aşteaptă un anumit timp şi reciteşte starea lui WS1 în tabelul WS-Status. WSM încearcă până când starea serviciului WS1 devine READY, sau FAULT (serviciul are o eroare majoră de funcţionare) sau până s-a atins o limită a acestor încercări. În configuraţia sistemului trebuie prevăzut un număr maxim de reîncercări, sau un timp maxim pentru încercări repetate.

Dacă starea serviciului principal WS1 este FAULT, atunci în sistem se petrec o serie de operaţii de resincronizare descrise în cele ce urmează.

6.5. RESINCRONIZAREA SERVICIILOR ÎN SISTEM

Din datele stocate intern în tabelul WS-Status, Managerul de servicii WSM este capabil să determine identificatorul ultimei cereri rezolvate cu succes pe fiecare serviciu worker WS_1, WS_2 … WS_n (n>2).

Există o diversitate de situaţii care pot să apară în timpul funcţionării normale a serviciului principal (Primary Worker) şi să determine eşecul serviciului respectiv. O taxonomie a defectelor de funcţionare este prezentată în [3].

Un eşec (o cădere) apare când este solicitat un serviciu worker pentru a trata o anumită cerere, dar acesta nu returnează rezultate. Datorită naturii distribuite a sistemului, defectele se pot situa în diferite locuri:

- la nivelul sistemului de operare care găzduieşte serviciul principal;
- în cadrul procesului iniţiat de serviciul principal;
- în firul de execuţie existent în cadrul procesului serviciului principal;
- în cadrul unuia din serviciile atomice (situate pe nivelul Service Layer);
- pe conexiunea de reţea dintre sistemul pe care rulează WSM şi cel pe care rulează serviciul worker apelat;
- pe conexiunea de reţea dintre gazda serviciului worker şi gazda unui serviciu atomic;
- la nivelul gazdei sistemului de gestiune a bazelor de date.

În timp, defectele pot să apară:

- în timpul transmiterii cererii către serviciul worker principal;
- în timpul procesării cererii de către serviciul worker principal;
- în timpul procesării la nivelul serviciilor atomice;
- în timpul procesului de transmitere a datelor de la bazele de date către serviciile atomice care au solicitat datele respective;
- în timpul procesului de transmitere a datelor de răspuns de la serviciile atomice către serviciul worker principal;
- în timpul procesului de transmitere a răspunsurilor de la worker la WSM.

Toate situaţiile menţionate mai sus pot determina menţinerea serviciului principal în starea BUSY un timp nedefinit sau chiar blocarea procesului serviciului principal în cadrul sistemului de operare.

Dacă serviciul principal, de exemplu WS1, este blocat în starea BUSY, dar serviciile de sprijin au terminat deja sarcinile primite şi sunt în starea READY, atunci, după un timp de aşteptare, WSM consideră că serviciul principal WS1 este blocat şi notează starea FAULT a acestui serviciu în tabelul WS-Status.

În aceste condiţii, dacă apare o nouă cerere a unei aplicaţii client, operaţiile de lucru, notate mai sus cu 1, 2 … 6, trebuie realizate de către un alt serviciu worker, de sprijin.

În acest timp, la nivelul serviciului principal WS1 trebuie desfăşurat un proces de recuperare după blocare. Managerul de servicii WSM este responsabil de trimiterea tuturor cererilor pierdute către serviciul principal. Toate solicitările clienţilor neefectuate la nivelul serviciului WS1 trebuie procesate în ordinea cronologică a apariţiei lor, astfel încât, la final, toate serviciile din cadrul sistemului să fie în aceeaşi stare. Astfel se pune în practică o funcţionalitate a sistemului: *sincronizarea serviciilor*.

Toate cererile clienţilor care presupun scrierea unor informaţii în bazele de date trebuie procesate tranzacţional. După ce o operaţie s-a finalizat cu succes, managerul de servicii WSM notează acest lucru în baza de date locală cu toate cererile primite şi starea lor actuală.

Înregistrarea tuturor cererilor, folosirea tranzacţiilor în lucrul cu sistemul de baze de date şi notarea finalizării cu succes a cererilor sunt elemente foarte importante necesare în faza de recuperare după o cădere a managerului de servicii WSM. După reiniţializarea managerului de servicii WSM, în procesul nou creat se vor retransmite serviciilor worker toate cererile înregistrate care nu au fost finalizate cu succes. Acest mecanism permite recuperarea WSM după o cădere de sistem şi readucerea sistemului într-o stare consistentă în care toate serviciile au executat cu succes toate cererile primite.

Toate aceste eşecuri de sistem, operaţiile de recuperare şi resincronizare a serviciilor sunt transparente pentru aplicaţia client, în sensul că aceasta nu are cunoştinţă de aceste operaţii desfăşurate pe partea serverelor. Căderea unui serviciu este mascată prin folosirea unui grup de servicii worker şi a unui manager de servicii capabil de următoarele operaţii: înregistrarea tuturor solicitărilor clienţilor şi a finalizării lor, managementul replicării execuţiei pe mai multe servicii, detectarea eşecului unui serviciu şi recuperarea sa ulterioară.

Managerul de servicii WSM menţine date statistice despre activitatea serviciilor worker în propria sa bază de date: timpul total de lucru efectiv în ultimele 5 minute, în ultimele 30 minute şi, foarte important, identificatorul ultimei cereri efectuată cu succes. Astfel, WSM este capabil să detecteze cel mai rapid serviciu worker în ultimele 5 minute sau în ultima jumătate de oră. Periodic, WSM va alege cel mai rapid serviciu drept serviciu principal, sau îl va menţine pe cel actual.

6.6. SECURIZAREA COMUNICAŢIILOR ÎNTRE SERVICII

Analizând arhitectura sistemului propus în Fig. 6.2 şi modul său de operare se constată că serviciile worker pot fi considerate *clienţi* pentru managerul de servicii, iar problema comunicaţiilor dintre manager şi serviciile worker se referă, practic, la *difuzarea securizată a unor date către un grup de clienţi legitimi*.

Trebuie remarcat faptul că în timpul procesului de resincronizare a serviciilor worker, în lipsa unor mecanisme de asigurare a securităţii comunicaţiilor, există posibilitatea ca un atacator să încerce să se prezinte în faţa managerului de servicii drept un serviciu legitim. Înainte de această încercare, prin interceptarea mesajelor pe reţea, el poate aduna date specifice despre operaţiile efectuate de un worker la solicitarea managerului de servicii.

De exemplu, dacă atacatorul ar putea afla identificatorul unei operaţii anterioare efectuate cu succes pe serviciul worker legitim supravegheat, atunci el ar putea încerca să se substituie serviciului legitim încercând să înşele managerul de servicii. Acesta, considerându-l serviciu legitim şi aflând ID-ul real al unei operaţii anterioare, va detecta că respectivul worker este „rămas în urmă" şi va declanşa procesul de resincronizare. Aceasta presupune că va transmite către serviciul impostor o serie de operaţii de scriere în baza de date, ceea ce înseamnă că vor fi divulgate multe date către atacator.

Pe lângă aflarea unor date secrete şi blocarea unui serviciu legitim, această situaţie ar putea fi folosită de atacator pentru construirea unor atacuri ulterioare mai complexe. Deci, datorită faptului că managerul transmite către un serviciu worker un set întreg de operaţii pentru a-l aduce în starea de sincronizare cu celelalte servicii worker, procesul de resincronizare prezintă un risc crescut în faţa unor atacuri potenţiale. De aceea, trebuie asigurată *securitatea comunicaţiilor dintre managerul de servicii şi serviciile worker*.

Pentru a păstra un grad de generalitate, se consideră că în sistemul analizat există un număr maxim de clienţi, notat cu n, şi în scopul securizării comunicaţiilor dintre server şi clienţi, trebuie să existe la nivelul sistemului un *secret S* care se va împărţi într-un anumit mod în n părţi ($n \geq 2$), notate D_i unde $i = 1, 2, \ldots n$, astfel încât să fie îndeplinite condiţiile:

- cunoaşterea oricăror k $(2 \leq k \leq n)$ părţi din totalul de n permite reconstituirea secretului S (valoarea k are rolul unui prag);
- cunoaşterea oricăror $k-1$ (sau mai puţine) părţi lasă secretul S complet nedeterminat (numeroasele variante existente sunt la fel de posibile).

O asemenea schemă de partajare a unui secret S între n componente ale unui sistem, cu un prag de determinare a secretului notat k ($2 \leq k \leq n$), este numită *threshold scheme (k, n)* , [48].

Folosirea unei asemenea scheme presupune rezolvarea a două probleme:

- stabilirea părţilor secretului S, notate D_i unde $i = 1, 2, \ldots n$ ($n \geq 2$);
- reconstituirea secretului S folosind oricare k părţi din n ($2 \leq k \leq n$).

Sunt cunoscute scheme de distribuire a unui secret bazate pe *Teorema Chineză a Restului*. Aceste metode folosesc funcţia *modulo* şi o serie de numere prime între ele, iar una din primele operaţii este alegerea corectă a acestora [14].

În această lucrare se va folosi o altă schemă bazată pe *polinoamele de interpolare Lagrange*.

Considerăm arhitectura de sistem prezentată anterior în Fig. 6.2. Componentele de sistem care intervin direct în această schemă de partajare a unui secret sunt: managerul de servicii şi serviciile worker.

6.6.1. STABILIREA PĂRŢILOR SECRETULUI

Pentru stabilirea părţilor secretului, modul de lucru este următorul:

- Secretul este un număr întreg: S.
- Numărul de părţi în care va fi împărţit secretul S este: n ($n \geq 2$)
- Pragul (numărul de părţi suficiente pentru a reconstitui secretul S) este k ($2 \leq k \leq n$).
- Se construieşte un polinom de gradul k-1 de forma:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \quad (1)$$

unde termenul liber a_0 este egal cu secretul S, iar coeficienţii a_1 şi a_2 sunt aleşi aleator din mulţimea numerelor întregi nenule Z^*:

$$a_0 = S \quad (2)$$

$$a_1 \in Z^*, \; a_2 \in Z^* \quad (3)$$

- Se determină cele n părţi în care urmează să fie împărţit secretul S. Se aleg n valori *nenule* pentru x şi se calculează corespunzător n valori y astfel:

$$y_i = f(x_i), \text{ unde i = 1, 2, … } n \quad (4)$$

Valorile x pot avea semnificaţia de identificatori pentru serviciile worker: 1, 2, …n.

În cazul de faţă se formează n perechi de valori $\{x_i ; y_i\}$, unde i = 1, 2, … n, astfel:

$$D_1 = \{x_1 ; y_1\} ; \ldots D_i = \{x_i ; y_i\} \ldots D_n = \{x_n ; y_n\} \quad (5)$$

- Cele n părţi ale secretului S se distribuie pe cele n servicii worker, câte o parte pentru fiecare. Dacă valorile x de mai sus au sensul de identificator pentru serviciul respectiv, atunci distribuirea părţilor D_i va respecta acest identificator.

6.6.2. RECONSTITUIREA SECRETULUI

În cazul arhitecturii de sistem analizate aici, reconstituirea secretului se face la nivelul managerului de servicii, iar părţile D_i ale secretului S necesare reconstituirii sunt aflate de către managerul de servicii prin interogarea serviciilor worker.

Scopul reconstituirii secretului S este *verificarea legitimităţii serviciilor* interogate.

Dacă valoarea calculată a secretului S coincide cu valoarea stabilită iniţial, din care provin părţile D_i stocate la nivelul serviciilor worker, atunci cu siguranţă serviciile de la care s-au obţinut părţile D_i sunt *servicii legitime*. În caz contrar, cel puţin unul din servicii este impostor.

Reconstituirea secretului S se poate face dacă sunt cunoscute oricare k părţi D_i din totalul de n. Valoarea lui k respectă relaţia ($2 \leq k \leq n$). Secretul nu poate fi reconstituit dacă numărul de părţi cunoscute este mai mic decât k.

Pentru reconstituirea secretului *S* modul de lucru este următorul:

- Numărul de părţi cunoscute: $k = 3$
- Se precizează cele *k* părţi cunoscute $\{x_i ; y_i\}$, ($i = 0, 1, \ldots k\text{-}1$).

$$D_0 = \{x_0 ; y_0\} ; D_1 = \{x_1 ; y_1\} ; \ldots D_i = \{x_i ; y_i\} \ldots D_{k-1} = \{x_{k-1} ; y_{k-1}\} \quad (6)$$

Observaţie. Părţile cunoscute primesc un număr de ordine *i* ($i = 0, 1, \ldots k\text{-}1$).

- Polinomul de interpolare Lagrange este dat de relaţia:

$$L(x) = \sum_{i=0}^{k-1} y_i \, l_i(x) \quad (7)$$

unde valorile y_i ($i = 0, 1, \ldots k\text{-}1$) sunt cele din părţile cunoscute D_i ($i = 0, 1, \ldots k\text{-}1$), iar expresiile $l_i(x)$ ($i = 0, 1, \ldots k\text{-}1$) sunt *polinoamele de bază Lagrange*, calculate mai jos. Polinomul de interpolare Lagrange este o combinaţie liniară de polinoame de bază.

- Se calculează polinoamele de bază Lagrange $l_i(x)$ pentru fiecare *i* ($i = 0, 1, \ldots k\text{-}1$):

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{x - x_j}{x_i - x_j} \quad (8)$$

sau
$$l_i(x) = \frac{x - x_0}{x_i - x_0} \; \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \; \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_{k-1}}{x_i - x_{k-1}} \quad (9)$$

Având determinate polinoamele de bază $l_i(x)$ ($i = 0, 1, \ldots k\text{-}1$) se calculează expresia polinomului de interpolare $L(x)$, de gradul k-1, cu relaţia (7).

Termenul liber al polinomului de interpolare Lagrange, $L(x)$, este chiar secretul *S*.

Deoarece, în final, contează numai valoarea termenului liber din expresia polinomului de interpolare $L(x)$ este util să se analizeze modul cum se formează termenul liber din expresia de calcul a polinomului de interpolare Lagrange $L(x)$.

Fiecare polinom de bază Lagrange este de gradul k-1 având forma generală:

$$l_i(x) = s_i + b_{1i} x + b_{2i} x^2 + \ldots + b_{k-1i} x^{k-1} \quad (10)$$

Analizând relaţia (9) se observă că termenul liber este dat de expresia:

$$s_i = \frac{-x_0}{x_i - x_0} \; \frac{-x_1}{x_i - x_1} \cdots \frac{-x_{i-1}}{x_i - x_{i-1}} \; \frac{-x_{i+1}}{x_i - x_{i+1}} \cdots \frac{-x_{k-1}}{x_i - x_{k-1}} \quad (11)$$

Pentru concizie, produsul de la numitorul expresiei (11) se notează P_i ($i = 0, 1, \ldots k\text{-}1$):

$$P_i = (x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_{k-1}) \quad (12)$$

Relaţia este echivalentă cu

$$P_i = \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \left(x_i - x_j \right) \tag{13}$$

Astfel, relaţia (11) devine:

$$s_i = \frac{(-x_0)(-x_1)\cdots(-x_{i-1})(-x_{i+1})\cdots(-x_{k-1})}{P_i} \tag{14}$$

Relaţia de mai sus este echivalentă cu

$$s_i = \frac{(-1)^{k-1} x_0 x_1 \cdots x_{i-1} x_{i+1} \cdots x_{k-1}}{P_i} \tag{15}$$

Relaţia de mai sus se scrie într-o formă concentrată astfel:

$$s_i = \frac{(-1)^{k-1}}{P_i} \prod_{\substack{j=0 \\ j \neq i}}^{k-1} x_j \tag{16}$$

Se introduce notaţia:

$$X_i = \prod_{\substack{j=0 \\ j \neq i}}^{k-1} x_j \tag{17}$$

Relaţia (16) devine:

$$s_i = (-1)^{k-1} \frac{X_i}{P_i} \tag{18}$$

Folosind relaţiile (10) ÷ (18), termenul liber din expresia (7) a polinomului de interpolare $L(x)$ (având semnificaţia de valoare reconstituită a secretului S) este dat de relaţia următoare:

$$S = y_0 s_0 + y_1 s_1 + \cdots + y_{k-1} s_{k-1} \tag{19}$$

O asemenea schemă de partajare a unui secret S între n componente ale unui sistem, cu un prag de determinare a secretului notat k ($2 \leq k \leq n$), este numită *threshold scheme (k, n)* şi are o serie de *proprietăţi specifice* [48]:

1. Dacă se păstrează nemodificat pragul k, atunci se poate modifica numărul total n de părţi ale secretului S fără a afecta valoarea sa.
2. Dacă se păstrează nemodificat pragul k, atunci părţile D_i ale secretului se pot adăuga sau şterge fără a afecta celelalte părţi existente.
3. Dacă se păstrează nemodificat pragul k, atunci modificarea coeficienţilor notaţi a_1, a_2 … a_{k-1} , care intervin în expresia polinomului $f(x)$ din relaţia (1), nu afectează reconstituirea secretului S.

6.6.3. SECURIZAREA DIFUZĂRII MESAJELOR

Considerăm un sistem distribuit având arhitectura prezentată anterior în acest capitol. Între managerul de servicii şi serviciile worker au loc schimburi de mesaje. Trebuie remarcat că, în modul de operare normală a sistemului când toate serviciile worker sunt funcţionale şi sincronizate, managerul solicită tuturor serviciilor worker aceeaşi operaţie, în urma solicitării unui client.

În cazul în care mesajele nu sunt securizate, managerul de servicii transmite acelaşi mesaj necriptat *M* către serviciile worker (Fig. 6.3).

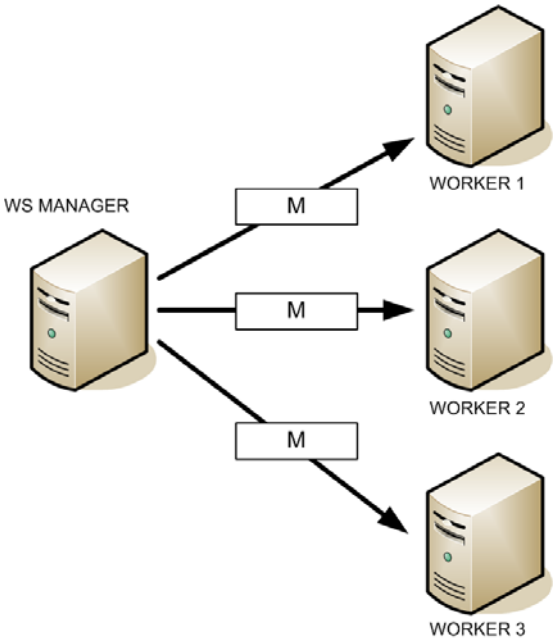

Fig. 6.3. Difuzare de mesaje necriptate către serviciile worker

În abordarea tradiţională, comunicaţiile dintre manager şi serviciile worker sunt securizate folosind cripto-sisteme cu cheie publică sau cu cheie privată.

În cele ce urmează se vor folosi notaţiile:

- *M* – mesajul necriptat
- *C* – mesaj criptat
- *e* – cheia pentru criptarea mesajului *M*
- *d* – cheia pentru decriptarea mesajului *M*.

Dacă se foloseşte o abordare *point-to-point* pentru securizarea difuzării de mesaje, atunci managerul de servicii, în calitate de emiţător de mesaje, trebuie să realizeze criptarea şi transmiterea mesajelor separat pentru fiecare serviciu worker în parte. Mai mult, el trebuie să gestioneze cheile de criptare-decriptare pentru fiecare serviciu worker cu care comunică în cadrul sistemului. Această abordare este total ineficientă deoarece un acelaşi mesaj care trebuie difuzat către mai mulţi destinatari este criptat de mai multe ori şi transmis fiecărui destinatar în parte.

O altă abordare este cea în care se consideră un grup de componente ale sistemului distribuit, pentru care este configurată *o cheie de grup*, dacă se foloseşte un cripto-sistem cu cheie privată, sau *o pereche de chei de grup*, una pentru criptare şi una pentru decriptare, dacă se foloseşte un cripto-sistem cu cheie publică. În cazul sistemului considerat aici, format dintr-un manager de servicii şi mai multe servicii worker, cheia privată de grup este stocată de manager, iar cheia publică de grup este cunoscută de fiecare serviciu worker. Dezavantajul acestei abordări este aceea că, dacă un atacator reuşeşte să obţină cheia publică de grup (sau cheia secretă a grupului) de la un serviciu worker, atunci el va putea descifra mesajele difuzate de manager către oricare din membrii grupului de servicii worker şi va putea trimite mesaje false către WS Manager cu scopul de a compromite întreaga funcţionare a sistemului.

O a treia abordare se referă la difuzarea de mesaje către mai mulţi destinatari, *message broadcasting*. Pentru securizarea difuzării mesajelor se foloseşte o *cheie de sesiune*, dacă se foloseşte un cripto-sistem cu cheie privată, sau *o pereche de chei de sesiune*, una pentru criptare şi alta pentru decriptare, dacă se foloseşte un cripto-sistem cu cheie publică.

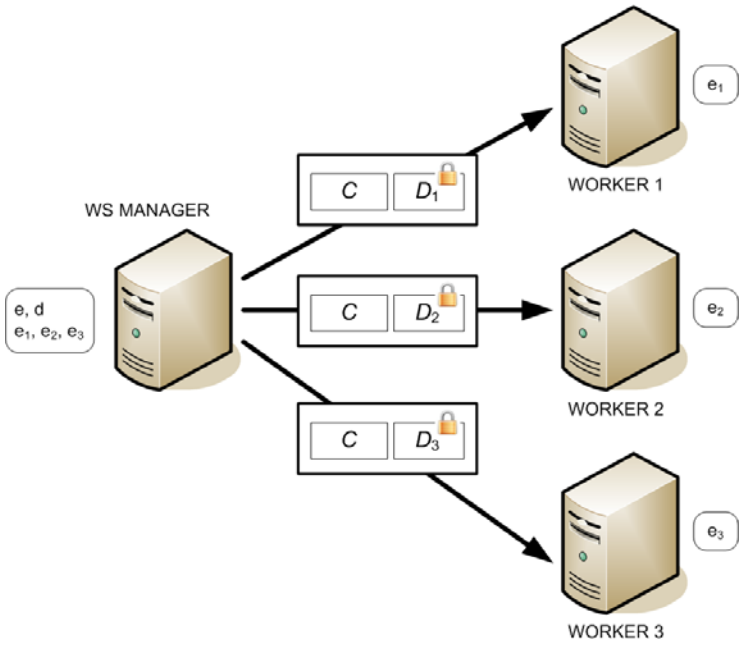

Fig. 6.4. Difuzare de mesaje securizate către serviciile worker

În această abordare (*secure message broadcasting*) trebuie stabilită o metodă sigură de transmitere a cheii de sesiune pentru decriptare către destinatar, simultan cu mesajul criptat. Evident, această cheie de sesiune pentru decriptare nu poate fi transmisă ca text clar.

În această lucrare se propune transmiterea cheii de decriptare de sesiune într-o formă *criptată* şi *protejată cu un lacăt*.

Emiţătorul, WS Manager, difuzează către mai mulţi destinatari, servicii worker, un acelaşi mesaj criptat împreună cu cheia de sesiune pentru decriptare protejată cu un lacăt. La nivelul fiecărui destinatar există toate datele necesare pentru reconstituirea acestei chei de sesiune, după care devine posibilă decriptarea mesajului propriu-zis. Astfel este asigurată o *difuzare securizată a aceluiaşi mesaj către un grup de destinatari legitimi*.

În cele ce urmează se propune o tehnică de protejare a cheii de sesiune cu un lacăt care poate fi înlăturat („descuiat") de oricare din destinatarii legitimi ai mesajului. Mai exact, operaţia de *înlăturare a lacătului* este chiar operaţia de reconstituire a secretului descrisă mai sus, iar *secretul* propriu-zis este valoarea criptată a cheii de decriptare de sesiune *d*. Cheia de decriptare *d* este cea care permite destinatarului mesajului să determine mesajul M original.

La nivelul componentei de management, WS Manager, se cunosc următoarele date:

- *e* – cheia de sesiune pentru criptare
- *d* – cheia de sesiune pentru decriptare
- e_1, e_2, e_3 – cheile de criptare pentru Worker 1, Worker 2 şi Worker 3.

La nivelul unui serviciu worker, secretul trebuie determinat pe baza unor informaţii conţinute în mesajul difuzat de manager şi a altor informaţii existente în cadrul parametrilor de configurare ai unui serviciu worker. Acestea din urmă nu vor fi comunicate pe reţea niciodată, împiedicând atacatorii să determine secretul *S*.

Conţinutul mesajului securizat difuzat de manager cuprinde 2 părţi:

- *C* - încărcătura utilă, respectiv textul criptat
- *Y*(*D*) – valoarea criptată *D* a cheii pentru decriptare, *d*, protejată cu lacăt.

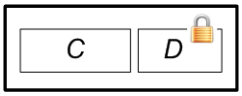

Fig. 6.5. Mesaj securizat difuzat de manager către un serviciu worker

Destinatarul unui asemenea mesaj trebuie să folosească tehnica de reconstituire a secretului, prezentată anterior, pentru a înlătura lacătul şi a obţine valoarea criptată *D* a cheii de sesiune pentru decriptare *d*. Ulterior, el decriptează valoarea *D* folosind cheia de decriptare proprie d_i (i = 1, 2 sau 3) şi, în final, obţine cheia de decriptare de sesiune, *d*. Având cheia de sesiune *d* destinatarul mesajului decriptează mesajul criptat *C*, obţinând, în final, mesajul *M*.

În cazul sistemului cu replicare considerat aici, mesajul *M* reprezintă solicitări pentru executarea unor operaţii la nivelul serviciului worker, în timpul modului normal de operare a sistemului cu replicare. Uzual mesajul M este un fişier XML având o schemă specifică sistemului cu replicare.

6.6.4. PROTEJAREA CHEII DE DECRIPTARE

Considerăm sistemul cu replicare propus în Fig. 6.2. Difuzarea mesajelor securizate de către componenta de management este ilustrată în Fig. 6.4.

Structura mesajelor este cea prezentată în Fig. 6.5. Cheia de decriptare de sesiune, *d*, este, mai întâi, ea însăşi criptată folosind cheia de criptare a destinatarului mesajului, notată e_i (i = 1, 2, 3) obţinându-se valoarea D_i. Această

valoare D_i nu va fi transmisă în această formă pe reţea. Ea este folosită în cadrul unui algoritm bazat pe polinoamele de interpolare Lagrange ce produce un set de valori y_i , i = 1, 2 …*k-1*, care formează ceea ce se numeşte *lacăt*.

La nivelul emiţătorului mesajului se efectuează procedura de creare a lacătului. Pe baza informaţiilor din mesajul securizat, destinatarul mesajului efectuează procedura de înlăturare a lacătului şi reconstituire a valorii criptate D_i.

La nivelul serviciului worker i, reconstituirea valorii D_i este chiar reconstituirea secretului S în cadrul schemei de *partajare a unui secret cu prag (k, n)*, conform relaţiei (20).

Pentru aceasta, serviciul worker trebuie să cunoască mai mulţi parametri. Cu cât valoarea pragului k este mai mare, cu atât creşte numărul de parametri necesari reconstituirii secretului. Modul de înlăturare a lacătului de către destinatar este tratat ulterior.

În această teză se propune folosirea unei *scheme de partajare a unui secret, cu prag (3, n)*, în care, pentru a menţine un *nivel ridicat de protecţie*, anumiţi parametri nu vor fi transmişi pe reţea niciodată, ci vor fi cunoscuţi numai la nivelul componentelor sistemului.

În cazul în care pragul k = 3, relaţia (17) poate fi adusă la o formă mai simplă:

$$S = y_0 \frac{x_1 x_2}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{x_0 x_2}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{x_0 x_1}{(x_2 - x_0)(x_2 - x_1)} \qquad (21)$$

Practic, pentru determinarea secretului S trebuie să fie cunoscute 3 perechi de valori:

$$\{x_0 ; y_0\} ; \{x_1 ; y_1\} ; \{x_2 ; y_2\} \qquad (22)$$

Transmiterea tuturor acestor valori în interiorul mesajului cu structura prezentată în Fig.6.5 ar conduce la expunerea prea multor informaţii pe care un atacator care analizează traficul de reţea le-ar putea folosi pentru determinarea secretului S.

De aceea, este necesară o metodă prin care să nu fie transmise pe reţea toate datele. În această lucrare se propune următoarea variantă de lucru cu parametrii din relaţia (22):

1. Valorile x_0, x_1 şi x_2 nu trebuie să fie transmise pe reţea niciodată. Aceste valori sunt, de fapt, identificatori din cadrul sistemului, de exemplu:
   - *GID* (Group ID) – identificatorul grupului de servicii worker;
   - *SID* (Sender ID) – identificatorul emiţătorului mesajului (WS Manager);
   - *RID* (Receiver ID) – identificatorul destinatarului (serviciul worker).
   Valorile GID, SID, RID sunt parametri cunoscuţi de serviciului worker.

2. Valorile y_0, y_1 şi y_2 din relaţia (19) depind de funcţia *f(x)* din relaţia (1), folosită la stabilirea părţilor secretului. Ele depind de secretul S şi de alţi 2 coeficienţi aleşi aleator, a_1 şi a_2. Valorile S, a_1 şi a_2 se schimbă la fiecare sesiune de difuzare a unui mesaj pentru ca un atacator care interceptează mesaje să nu le poată determina. Valorile y_0, y_1 şi y_2 se transmit pe reţea în mesaje cu structura din Fig. 6.5.

6.6.5. PROTOCOLUL PENTRU CREAREA MESAJELOR SECURIZATE

Mesajele securizate, cu structura dată în Fig. 6.5, se formează la nivelul componentei de management, WS Manager, şi sunt difuzate către serviciile worker. În urma unei cereri făcută de o aplicaţie client şi adresată componentei de management, aceasta pregăteşte un mesaj *M* către serviciile worker, uzual sub forma unui document XML.

Date de intrare:

- *Mesajul original M* (text clar, necriptat) având un conţinut determinat de funcţionarea sistemului cu replicare sub controlul unei componente manager.
- *Cheia de criptare de sesiune*, *e*, care este folosită pentru obţinerea mesajului criptat *C* în urma unui proces de criptare a mesajului *M*.
- *Cheia de decriptare de sesiune*, *d*, necesară procesului destinatar pentru obţinerea mesajului original *M* în urma unui proces de decriptare.
- Valoarea pragului *k* ($k \geq 2$).
- Valorile coeficienţilor a_0, a_1, … a_{k-1} necesari pentru a forma expresia funcţiei *f(x)* din relaţia (26).
- *Cheia de criptare e_i a destinatarului i (i = 1, 2 … n)*, necesară pentru criptarea cheii de sesiune *d* şi obţinerea de valori particularizate D_i pentru fiecare serviciu worker destinatar.

Informaţii de ieşire:

- *Mesajul criptat C*, pentru care s-a folosit cheia de criptare *e*, rel. (24);
- *Valoarea Y_i care conţine cheia de decriptare d protejată de un lacăt*, (care este determinată pe baza valorii criptate a cheii *d*, notată D_i):

$$Y_i\left(D_i, x_0, x_1 \ldots x_{k-1}\right) = \left\{y_{0,i}, y_{1,i}, \ldots y_{k-1,i}\right\} \tag{23}$$

Protocolul de creare a mesajelor securizate este prezentat în detaliu în cele ce urmează, iar schema sa este în Fig. 6.6.

PASUL 1

Se generează o cheie secretă de sesiune *e*, dacă se foloseşte un cripto-sistem cu cheie secretă, sau se generează o pereche de chei de sesiune, notate *e* şi *d*, dacă se foloseşte un cripto-sistem cu cheie publică. În primul caz, cheia de decriptare *d* = *e*, iar în al doilea caz cheile sunt diferite *d* != *e*.

PASUL 2

Se criptează mesajul *M* folosind cheia de criptare de sesiune, *e*, obţinându-se forma criptată *C*:

$$C = Enc\left(M, e\right) \tag{24}$$

PASUL 3

Ştiind cheia de criptare e_i a destinatarului i, i = 1, 2 … n, care este unul din membrii grupului format din n servicii worker, se criptează cheia de sesiune de decriptare d obţinându-se valori diferite pentru fiecare serviciu worker (Fig. 6.4):

$$D_i = Enc(d,\ e_i) \quad i = 1, 2, \ldots n \tag{25}$$

PASUL 4

Valoarea criptată D_i trebuie să fie *protejată cu un lacăt*.
Ea trebuie adusă la o altă formă astfel încât numai destinatarul legitim al acestei informaţii să o poată recalcula pe baza a două categorii de date.
Prima categorie se referă la date existente pe sistemul local şi care nu sunt transmise niciodată pe reţea.
A doua categorie se referă la date care sunt transmise pe reţea de către emiţătorul mesajului, în interiorul mesajului securizat alături de datele criptate conţinând mesajul original adresat serviciului worker.

Emiţătorul mesajului, ştiind valoarea pragului k, generează aleator valorile coeficienţilor întregi a_1, a_2, … a_{k-1} şi alege un secret S Folosind secretul S şi coeficienţii a_1, a_2, … a_{k-1}, se formează funcţia polinomială $f(x)$, astfel:

$$f(x) = S + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \tag{26}$$

În această teză se analizează mai întâi cazul k=3, deci cazul când polinomul $f(x)$ este de gradul 2.
Coeficienţii polinomului $f(x)$ sunt: S, a_1 şi a_2. Aceste 3 valori *nu sunt transmise pe reţea niciodată*.

$$f(x) = S + a_1 x + a_2 x^2 \tag{27}$$

După cum s-a propus mai sus, se consideră următoarele k=3 valori ale lui x:
x_0 = *GID* (Group ID) – identificatorul grupului de servicii worker;
x_1 = *SID* (Sender ID) – identificatorul emiţătorului mesajului (WS Manager);
x_2 = *RID* (Receiver ID) – identificatorul destinatarului (serviciul worker).

Aceste 3 valori, x_0, x_1, şi x_2, sunt cunoscute de către fiecare serviciu worker şi nu sunt transmise pe reţea niciodată.

- *Valoarea* x_0 este aceeaşi pentru toţi membrii unui grup şi este scrisă în fişierul de configurare al fiecărui serviciu worker din grup.
- *Valoarea* x_1 este identificatorul managerului de servicii şi este scrisă, de asemenea, în fişierul de configurare al fiecărui serviciu worker din grup.
- *Valoarea* x_2 este diferită pentru fiecare serviciu worker, deoarece fiecare din ele are un identificator al său, scris în fişierul său de configurare.

Managerul de servicii cunoaşte toţi identificatorii serviciilor pe care le gestionează şi foloseşte aceste valori x_2 în calculul lui $y_2 = f(x_2)$.

Folosind relaţia (24) pentru valorile x_0, x_1, şi x_2 se obţin valorile y_0, y_1 şi, respectiv, y_2.

$$y_0 = f(x_0), \quad y_1 = f(x_1), \quad y_2 = f(x_2) \tag{28}$$

Cele 3 valori $\{y_0, y_1, y_2\}$ sunt specifice pentru fiecare destinatar i.

Acest set de valori reprezintă valoarea cheii *d* criptate şi protejate cu lacăt.

$$\{y_{0,i}, y_{1,i}, y_{2,i}\} = Lock(D_i, x_0, x_1, x_2) \quad (29)$$

Informaţiile de ieşire sunt grupate într-un mesaj securizat cu structura din Fig. 6.5.

Paşii prezentaţi în cele de mai sus sunt prezentaţi în schema din Fig. 6.6.

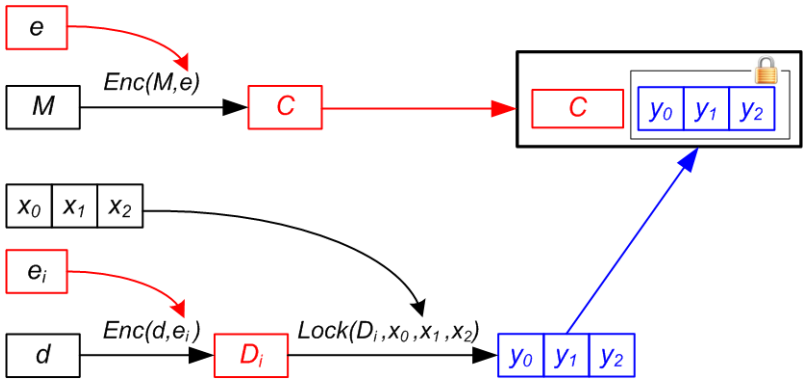

Fig. 6.6. Schema protocolului de creare a mesajelor securizate

Notaţiile folosite în figura de mai sus sunt următoarele:

- *e* – cheia de criptare de sesiune
- *M* – mesajul iniţial necriptat
- *C* – mesajul criptat cu cheia de sesiune *e*
- e_i (i = 1, 2, 3) cheia de criptare a destinatarului *i* (serviciul worker *i*)
- *d* – cheia de decriptare de sesiune (pereche cu cheia *e*)
- D_i – valoarea criptată a cheii *d* cu cheia destinatarului *i*, respectiv e_i
- x_0, x_1, x_2 – valori necesare ca argument în relaţiile (28), având sensurile precizate mai sus.
- y_0, y_1, y_2 – setul de valori care constituie forma protejată cu lacăt a valorii *D*, obţinute conform relaţiei (28) prin folosirea funcţiei *f(x)* din relaţia (27) şi a valorilor x_0, x_1, x_2 precizate mai sus.

Observaţii:

- Valoarea x_2 este specifică destinatarului (serviciului worker).
- Valorile y_0, y_1, y_2 sunt specifice destinatarului *i* (i = 1, 2, 3) şi au fost calculate folosind valori D_i specifice fiecărui destinatar.

Ambele observaţii de mai sus evidenţiază elemente care particularizează mesajul securizat adresat unui anumit serviciu worker. Astfel, fiecare serviciu worker primeşte un mesaj care nu poate fi descifrat de un alt serviciu din grupul de servicii worker, şi nu poate fi descifrat nici de un atacator care monitorizează reţeaua şi capturează mesajele difuzate.

6.6.6. PROTOCOLUL PENTRU DECRIPTAREA MESAJELOR SECURIZATE

Pentru extragerea informaţiei necriptate, destinatarul unui mesaj securizat trebuie să înlăture, mai întâi, lacătul care protejează valoarea criptată *D* a cheii de decriptare de sesiune, iar apoi să obţină cheia de decriptare de sesiune *d* pentru a putea decripta mesajul original *M*.

Date de intrare:
- *Mesajul criptat C.*
- *Valorile y_0, y_1, y_2 care constituie informaţia protejată de lacăt* (specifice destinatarului *i*, unde $i = 1, 2, \ldots n$, iar n = numărul de destinatari din grup):
- *Cheia de decriptare d_i a destinatarului i*, necesară pentru decriptarea valorilor D_i şi obţinerea cheii de decriptare de sesiune *d*.
- *Valorile x_0, x_1, x_2* (respectiv, *GID*, *SID*, *RID*, valori precizate mai sus).

Informaţii de ieşire:
- *Cheia de decriptare de sesiune, d*.
- *Mesajul original, necriptat M*.

Protocolul de decriptare a mesajelor este prezentat în cele ce urmează, iar schema sa este prezentată în Fig. 6.7.

PASUL 1

Înlăturarea lacătului de către destinatarul legitim al mesajului este o operaţie bazată pe *polinoamele de interpolare Lagrange* şi constă, practic, în reconstituirea unui *secret S* conform tehnicii prezentate anterior în acest capitol.

În general, secretul se determină cu relaţia (20). În cazul studiat aici pragul este $k = 3$, iar secretul se determină folosind relaţia (21).

Secretul S din tehnica de reconstituire a secretului este chiar valoarea D_i care reprezintă aici forma criptată cu cheia e_i a cheii de decriptare de sesiune, *d*.

$$D_i = Unlock\left(x_0, x_1, x_2, y_{0,i}, y_{1,i}, y_{2,i}\right) \tag{30}$$

PASUL 2

Folosind cheia de decriptare proprie d_i (perechea cheii de criptare e_i folosită de emiţătorul mesajului), destinatarul *i* decriptează valoarea D_i determinată la pasul anterior şi obţine cheia de decriptare de sesiune, *d*.

$$d = Dec\left(D_i, d_i\right) \quad i = 1, 2, \ldots n \tag{31}$$

PASUL 3

Mesajul iniţial, necriptat, *M* se obţine în urma unei operaţii de decriptare a mesajului criptat *C* folosind cheia de decriptare de sesiune, *d*:

$$M = Dec\left(C, d\right) \tag{32}$$

Paşii prezentaţi în cele de mai sus sunt prezentaţi în schema din Fig. 6.7.

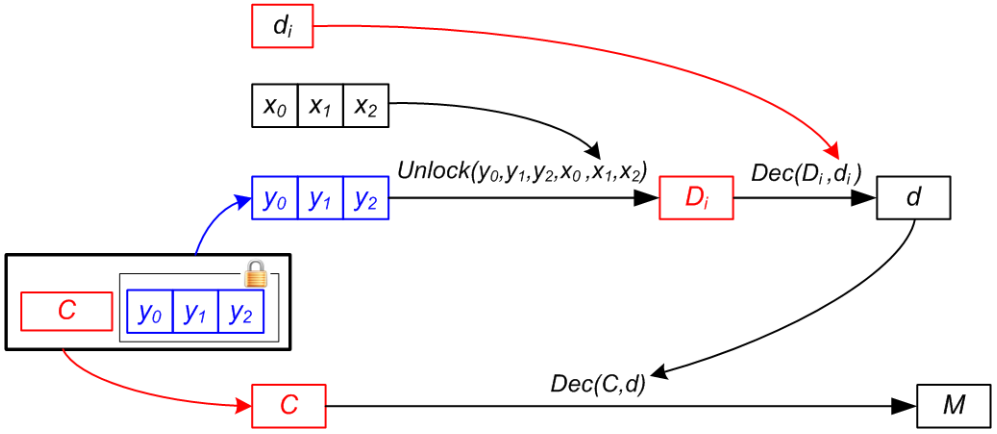

Fig. 6.7. Schema protocolului de decriptare a mesajelor securizate

6.6.7. MODUL DE PROTECŢIE ÎMPOTRIVA ATACURILOR

În sistemul cu replicare cu arhitectura din Fig. 6.2 în care mesajele sunt securizate şi difuzate conform Fig. 6.4 şi protocoalelor prezentate mai sus, sunt posibile următoarele *atacuri care vizează securitatea sistemului*:

- Atacul pentru descifrarea cheii de decriptare de sesiune, *d*.
- Atacul pentru obţinerea cheii de decriptare d_i a unui destinatar legitim.

A1. *Atacul pentru descifrarea cheii de decriptare de sesiune*, *d*, este efectuat de un intrus din afara grupului de destinatari legitimi ai mesajelor. Putem presupune că acest atacator cunoaşte că sistemul foloseşte pentru securizarea mesajelor o schemă bazată pe polinoame de interpolare Lagrange, cu pragul k=3 dedus din numărul de valori *y* existente în conţinutul mesajului, respectiv 3. Deci, el va încerca să determine datele necesare folosite în tehnica de reconstituire a secretului *S*, prezentată mai sus. Din conţinutul unui mesaj adresat unui destinatar legitim, atacatorul determină valorile y_0, y_1, y_2.

Ştiind valorile y_0, y_1, y_2, atacatorul va încerca să rezolve sistemul de ecuaţii de mai jos pentru a determina valoarea secretului *S*:

$$\begin{cases} S + a_1 x_0 + a_2 x_0^{\ 2} = y_0 \\ S + a_1 x_1 + a_2 x_1^{\ 2} = y_1 \\ S + a_1 x_2 + a_2 x_2^{\ 2} = y_2 \end{cases} \quad (33)$$

Sistemul are 6 necunoscute: *S*, a_1, a_2, x_0, x_1, x_2. Având doar valorile y_0, y_1, y_2, atacatorul nu poate rezolva sistemul de ecuaţii (33) şi, deci, nu poate afla secretul *S*.

Şansele atacatorului ar putea creşte dacă la următoarele mesaje adresate aceluiaşi destinatar legitim s-ar folosi aceleaşi valori pentru *S*, a_1, şi a_2. Dar acest

lucru nu se întâmplă pentru că, la fiecare mesaj difuzat către acelaşi destinatar, valorile S, a_1, şi a_2 se schimbă de către emiţător, chiar dacă valorile x_0, x_1, x_2 rămân aceleaşi.

Atacatorul ar putea încerca să rezolve sistemul de ecuaţii (33) folosind alt set de valori y_0, y_1, y_2 extrase dintr-un alt mesaj interceptat pe reţea, dar adresat altui destinatar.

Deoarece în protocolul propus mai sus, fiecare destinatar are o altă valoare x_2 şi, în plus, emiţătorul mesajului foloseşte un alt secret S pentru fiecare destinatar şi pentru fiecare sesiune de difuzare, atacatorul nu poate să rezolve sistemul de ecuaţii (33).

Secretul S, după cum s-a arătat mai sus, este valoarea D_i , respectiv valoarea criptată şi protejată cu un lacăt a cheii de decriptare de sesiune, d. Deoarece atacatorul nu poate determina secretul S, atunci el nu are cum să determine cheia de decriptare de sesiune, d.

Într-un scenariu pesimist, chiar dacă atacatorul determină secretul S, adică valoarea criptată D_i, el nu reuşeşte decât să îndepărteze lacătul, dar tot nu poate ajunge la mesajul M pentru că nu cunoaşte cheia de decriptare d_i a utilizatorului legitim i.

A2. *Atacul pentru obţinerea cheii de decriptare d_i a unui destinatar legitim* poate fi efectuat numai de un alt destinatar legitim din grupul de destinatari.

Un program maliţios care rulează pe maşina unde se află unul din serviciile worker legitime ar putea încerca să afle cheia de decriptare d_i a unui alt serviciu worker pentru a putea interveni (în sens negativ) în comunicaţiile dintre acesta şi componenta de management.

Dacă în sistemul distribuit s-a configurat un *cripto-sistem cu cheie secretă*, atunci fiecare serviciu worker are o cheie secretă unică, iar componenta de management foloseşte în difuzarea de mesaje către serviciile worker chei de criptare diferite. Astfel că valorile D_i care rezultă în urma criptării cheii de decriptare de sesiune d sunt diferite pentru destinatari diferiţi.

Dacă în sistemul distribuit s-a configurat un *cripto-sistem cu cheie publică*, atunci în relaţia dintre fiecare serviciu worker şi componenta de management se folosesc perechi diferite de chei pentru criptare-decriptare.

Un atacator (un program maliţios) existent pe maşina i din grupul de destinatari legitimi nu poate afla cheia de decriptare d_j a unui alt destinatar legitim j ($j \neq i$), chiar dacă a aflat cheia de decriptare d_i a sistemului pe care rulează şi de pe care îşi desfăşoară atacul.

Prin urmare, un atacator (fie că este membru al grupului de destinatari, fie un intrus din afara grupului) nu are cum să decripteze mesajele criptate C şi să afle conţinutul mesajului original, necriptat, M. În concluzie, protocolul propus mai sus, bazat pe polinoamele de interpolare Lagrange, menţine un nivel ridicat de securitate a mesajelor.

6.7. ABORDĂRI EXPERIMENTALE

Sistemul cu replicare manageriată a fost implementat pe sisteme de operare Windows. Pentru a găzdui procesele server s-au folosit sisteme identice, respectiv sisteme de operare Windows 2008 Server cu .NET Framework 4.0, iar serverul Web folosit a fost IIS 7. Ca server de baze de date s-a folosit MS SQL Server 2008.

Pe platforma .NET Framework au fost create, mai întâi, serviciile WCF (Windows Communication Foundation). Acestea pot fi găzduite de aplicaţii executabile, servicii de sistem Windows sau de aplicaţii ASP.NET. Evitarea conexiuni inutile de reţea s-a făcut prin izolarea segmentului de reţea cu gazdele aplicaţiei-client şi serverelor faţă de restul reţelei.

În scopul evaluării degradării performanţei când are loc procesul de replicare s-a prevăzut în codul aplicaţiei timpul de început al transmiterii cererii către servere şi timpul când ajunge răspunsul. Pentru a verifica dacă grupul de servicii este capabil de recuperare după eşecuri, au fost simulate erori hardware, în principal, întreruperea conexiunilor de reţea. S-a urmărit durata operaţiilor în diferite situaţii de eşec.

În timpul replicării apare o întârziere a răspunsului, ceea ce este normal. O întârziere mai mare a răspunsului este sesizată de aplicaţia-client atunci când sistemul execută sincronizarea serviciilor. Valoarea absolută, în secunde, a acestor întârzieri este mai puţin relevantă pentru că cererile sunt, în general, diferite, iar timpul de procesare a cererilor poate să varieze pe o plajă largă, iar mesajele de răspuns pot să aibă dimensiuni foarte diferite.

Considerăm că un indicator mult mai relevant pentru gradul de influenţă al fazei de replicare este *întârzierea relativă a sistemului, RD – Relative Delay*, definită astfel:

$$RD = \frac{t_r}{t_0} \tag{34}$$

Aici, t_0 este durata normală de procesare a unei anumite cereri, iar t_r este durata de procesare a aceleaşi cereri, dar în condiţiile în care a apărut şi o operaţie de replicare la nivelul serviciilor.

Sistemul poate fi configurat diferit în ceea ce priveşte modul de formare a răspunsului:

- *Răspuns rapid* – este trimis către client un mesaj de răspuns format în cel mai scurt timp pe baza răspunsului dat de serviciul worker cel mai rapid (Fig. 6.8).
- *Răspuns sigur* – este trimis către client un mesaj de răspuns după ce managerul de servicii a recepţionat răspunsuri identice de la toate serviciile worker (Fig. 6.9).

Dacă sistemul a fost setat pe modul de răspuns sigur, atunci timpul de răspuns al sistemului către aplicaţia client este puţin mai mare datorită faptului că managerul de servicii aşteaptă un anumit timp toate răspunsurile serviciilor worker şi apoi le verifică. Diferenţa se observă comparând Fig. 6.8 şi Fig. 6.9.

Timpul total de aşteptare la nivelul aplicaţiei client depinde de timpii necesari desfăşurării următoarelor etape:

- Cererea făcută de client traversează WAN către manager (REQ) – t_{REQ}
- Componenta manager desfăşoară operaţii pregătitoare – t_{PREP}
  Folosind notaţiile fazelor protocolului pentru operare normală, componenta manager înregistrează cererea clientului (LOG), verifică starea serviciilor worker (STA), iniţializează procesul de replicare (INI).
- Operaţia cerută de client este replicată pe serviciile worker (EXE) – t_{EXE}
- Răspunsurile serviciilor worker sunt preluate de manager care pregăteşte un răspuns către client (RPR) – t_{RPR}
- Răspunsul managerului către client traversează WAN (RSP) – t_{RSP}

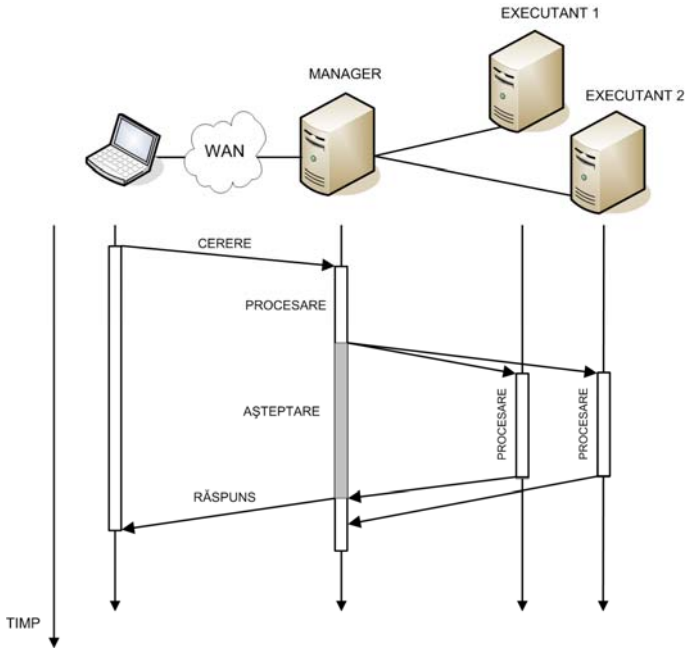

Fig. 6.8. Procesare rapidă pentru formarea răspunsului sistemului

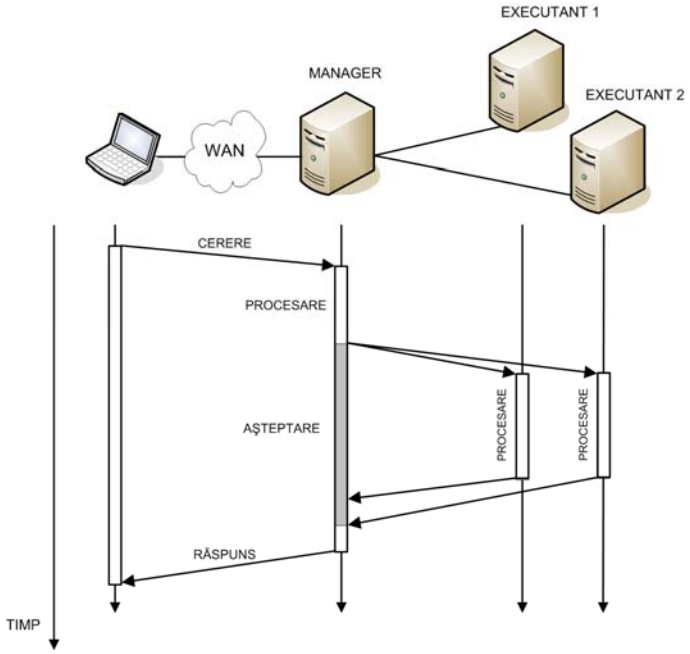

Fig. 6.9. Procesare sigură pentru formarea răspunsului sistemului

Deci, timpul total de aşteptare sesizat de client este:

$$t = t_{REQ} + t_{PREP} + t_{EXE} + t_{RPR} + t_{RSP} \quad (35)$$

Valorile t_{REQ} şi t_{RSP} depind de viteza reţelei şi de încărcarea ei în momentul măsurătorilor. Deci, viteza de răspuns a sistemului cu replicare este dată, practic, de suma valorilor timpilor $t_{PREP} + t_{EXE} + t_{RPR}$, ultimul termen al sumei fiind influenţat de modul de formare a răspunsului, după cum s-a precizat mai sus.

În timpul experimentelor efectuate s-a constatat că valorile întârzierii relative *RD* au fost influenţate semnificativ de complexitatea cererii făcute de client şi de dimensiunea mesajelor de răspuns [62].

Pentru sistemul folosit, valorile determinate s-au încadrat în intervalul:

$$RD = 1{,}12 \div 1{,}43 \quad (36)$$

Valorile întârzierii relative *RD* sunt specifice fiecărui sistem distribuit. Dezvoltatorii unui asemenea sistem pot determina valorile *RD* la punerea în funcţiune a sistemului şi să folosească aceste valori drept valori de referinţă (*baseline*) pentru ca, ulterior, noile valori *RD*, determinate periodic, să poată fi comparate cu cele de referinţă şi, ca urmare, să se poată cuantifica eventualele scăderi de performanţă ale sistemului respectiv.

6.8. CONCLUZII ŞI CONTRIBUŢII

În acest capitol a fost prezentat un sistem fiabil care implementează conceptele esenţiale ale unei arhitecturi tolerante la erori. Arhitectura propusă este structurată pe două niveluri şi este capabilă să rezolve intern defectele de funcţionare apărute fără să modifice interoperabilitatea serviciilor existente. Algoritmul intern al acestei arhitecturi permite dezvoltatorilor software să utilizeze diverse modele de programare pentru serviciile Web.

În cele de mai sus au fost prezentate importante funcţionalităţi ale acestei arhitecturi în ceea ce priveşte fiabilitatea: managementul erorilor, înregistrarea cererilor, recuperarea, replicarea şi transparenţa faţă de client.

Aplicaţiile client, ca de exemplu browserele Web, precum şi sistemele de operare client nu sunt sub controlul furnizorilor de servicii. De aceea, cazul în care aplicaţiile pe partea serverelor se bazează pe o participare din partea clientului în problema toleranţei la erori nu este nici practic, nici realist, şi nu a fost discutat aici.

Arhitectura propusă este bazată, în principal, pe Managerul serviciilor Web, Înregistratorul cererilor clienţilor şi Înregistratorul stării serviciilor Web, care lucrează împreună. Pe baza informaţiilor primite de la subsistemele de înregistrare, Managerul Serviciilor Web este capabil să decidă care serviciu Web este cel principal şi când trebuie făcută sincronizarea sistemului. Aplicaţia client tratează acest sistem ca un tot unitar, ca o unitate capabilă să rezolve la nivel intern defectele apărute şi să dea un răspuns corespunzător.

În cele de mai sus au fost prezentate aspecte ale implementării şi detalii despre rezultatele experimentale. De asemenea, a fost propusă o mărime relativă care să exprime influenţa fazei de replicare a serviciilor Web. O evaluare a influenţei fazei de replicare asupra performanţei globale a sistemului a arătat întârzieri previzibile, dar de mărime acceptabilă, în cazul sincronizării sistemului.

Sistemele în care se folosesc mai mult de două servicii replică permit configurări suplimentare pentru modul de formare a răspunsului către client la nivelul componentei de management în funcţie de răspunsurile date de serviciile worker. Se poate avea în vedere criteriul *timp de răspuns cât mai scurt* sau criteriul *răspuns sigur, format numai dacă un număr minim de replici au dat acelaşi răspuns componentei manager*. În această zonă sunt posibile diferite variante de optimizare a funcţionării sistemului cu replicare.

S-a acordat o importanţă specială securizării comunicaţiilor în cadrul sistemului distribuit. Managerul serviciilor Web difuzează către serviciile coordonate mesaje pentru efectuarea unor operaţii specifice sistemului cu replicare.

Pentru securizarea acestor mesaje şi protejarea împotriva unui atacator din afara sau din interiorul grupului de servicii s-a propus în acest capitol o tehnică de protejare a mesajelor împotriva utilizatorilor nelegitimi.

Varianta propusă foloseşte o protecţie bazată pe polinoame de interpolare Lagrange, dar şi cripto-sisteme cu cheie secretă sau cu cheie publică. S-a prezentat în acest capitol protocolul de creare a mesajelor securizate cât şi protocolul de decriptare şi s-a arătat că se menţine un nivel ridicat de securitate în difuzarea mesajelor în faţa atacurilor.

Contribuţiile aduse în acest capitol sunt:

- S-a propus *o arhitectură de sistem distribuit, tolerant la defecte*. Au fost definite componentele sistemului şi s-a precizat rolul fiecărei componente şi interacţiunile dintre ele. Sistemul transpune în practică un model abstract de sistem definit anterior, cu replicare coordonată de o componentă de management separată.
- S-a abordat problema securităţii sistemului distribuit, ca o componentă importantă a conceptului de *fiabilitate*. Pentru arhitectura de sistem cu replicare manageriată s-a studiat problema difuzării mesajelor de către componenta de management către serviciile pe care le coordonează. În scopul protejării sistemului de atacuri informatice a fost analizată *problema securizării mesajelor difuzate* în cadrul unui sistem tolerant la defecte, care foloseşte replicarea operaţiilor.
- S-a propus *un protocol pentru securizarea mesajelor* folosind o tehnică de protejare a informaţiei sensibile cu un *lacăt informatic*, iar pentru implementarea algoritmului de protejare s-au folosit polinoamele de interpolare Lagrange. S-a arătat că protocolul propus menţine un nivel ridicat de securitate în difuzarea mesajelor în faţa atacurilor.
- S-a propus *un nou parametru al sistemelor distribuite tolerante la defecte* care să exprime influenţa fazei de replicare a serviciilor Web asupra timpului de răspuns, numit *Relative Delay*. Această mărime relativă permite evaluarea modificărilor performanţei sistemului faţă de valori de referinţă.
- Abordările experimentale au arătat o influenţă a fazei de replicare asupra performanţei globale a sistemului, dar valorile întârzierilor sistemului pot fi menţinute la valori acceptabile. Monitorizarea evoluţiei în timp a valorilor acestui parametru permite aprecierea scăderilor de performanţă a sistemului faţă de o stare iniţială şi deschide noi posibilităţi de control şi optimizare.

Contribuţiile aduse în acest capitol, sistemul propus şi concluziile proprii au fost publicate în [62] şi [63].

7. CONCLUZII ŞI CONTRIBUŢII PERSONALE

Această teză abordează, mai întâi, problematica organizării resurselor unei aplicaţii Web conform principiilor REST şi continuă cu problema integrării în sistemele noi, moderne, a resurselor existente anterior. Se propune şi se analizează o variantă de implementare a unei arhitecturi orientate pe resurse, denumită aici *Arhitectură RESTful Multilayer*. Aceasta respectă principiile esenţiale: principiul adresabilităţii resurselor şi principiul interfeţei uniforme pentru accesul la resurse.

Organizarea aplicaţiilor distribuite a fost analizată în strânsă legătură cu conceptul de resurse algoritmice. Aici au fost incluse şi serviciile Web. Astfel s-a făcut trecerea spre conceptul de sistem distribuit orientat pe servicii Web.

S-au analizat, mai întâi, probleme de fiabilitate ale sistemelor distribuite. S-a efectuat o analiză a conceptelor actuale legate de disponibilitate, siguranţă, toleranţă la defecte şi fiabilitate, ca un concept integrator. Studiul problemelor de fiabilitate a sistemelor distribuite, fiind orientat către arhitecturi cu servicii Web, a continuat cu analiza replicării optimiste. S-au prezentat caracteristicile şi elementele replicării optimiste, comparativ cu replicarea pesimistă.

S-a propus şi s-a analizat un model de sistem distribuit în care un proces situat pe o anumită gazdă are rolul de a coordona procese situate pe alte gazde. Astfel s-a conturat modelul de sistem distribuit cu procese de management şi de execuţie separate.

Acest model a fost dezvoltat, în continuare, prin specificarea rolurilor proceselor implicate şi anume: rolul de executant al unor procese de replicare şi, respectiv, rolul de manager al acestor procese de replicare. Astfel s-a modelat un sistem tolerant la defecte ce foloseşte o tehnică de replicare în care intervine un proces de management al replicării şi mai multe procese executant.

Într-un capitol al tezei se definesc modurile de funcţionare ale acestui sistem de replicare cu separarea managementului de execuţie, precum şi protocoalele de funcţionare ale componentelor sistemului.

După tratarea problemelor specifice modelării sistemului de replicare cu separarea managementului de execuţie, într-un alt capitol al tezei, s-a arătat un mod de planificare şi realizare a unui sistem informatic orientat pe servicii Web, care, pentru a fi tolerant la defecte, foloseşte tehnica de replicare manageriată, descrisă anterior.

Aici se prezintă şi abordările experimentale, sistemele hardware şi software folosite şi, mai mult, se evidenţiază corelaţia dintre întârzierea sistemului în formularea unui răspuns la cererile clienţilor şi procesele de replicare. În această teză se defineşte un indicator de performanţă a funcţionării sistemelor tolerante la defecte, bazate pe tehnica de replicare cu procese de execuţie la distanţă. Acest parametru a fost denumit *întârziere relativă*.

Lucrarea prezintă valorile experimentale specifice sistemelor folosite şi, pentru generalizare, propune folosirea valorilor iniţiale ale acestui parametru ca valori de referinţă pentru a monitoriza, în timp, eventuala degradare a performanţelor sistemelor care folosesc replicarea proceselor pe gazde aflate la distanţă.

În cadrul conceptului de *fiabilitate* a sistemelor, care este un concept mai larg, apare inclus şi conceptul de *securitate*. În această teză s-a abordat problema securizării comunicaţiilor între procesul manager al replicării şi procesele de execuţie. Comunicaţiile dintre aceste procese legitime în cadrul sistemului cu replicare trebuie securizate pentru a împiedica eventualele atacuri din partea unor procese externe care vizează accesul la date sensibile (date de autentificare) sau date specifice fazelor de lucru ale sistemului cu replicare manageriată.

În această teză s-a prezentat o tehnică de securizare a mesajelor transmise între procesele manager şi executant folosind un *lacăt informatic*. S-au formulat protocoalele pentru crearea mesajelor securizate, respectiv, pentru decriptarea mesajelor securizate. Paşii necesari sunt descrişi pe larg, cu detalii, apoi sunt prezentaţi pe scurt în schemele protocoalelor menţionate.

În final, s-a efectuat o analiză care arată că atacurile venite din partea unor procese nelegitime nu vor avea succes şi nu vor conduce la dezvăluirea datelor sensibile din mesajele transmise pe reţea între procesele legitime.

În această teză s-au realizat următoarele:

- S-a efectuat o analiză a evoluţiei modului de adresare a resurselor într-o aplicaţie Web şi o clasificare a arhitecturilor pe baza stilului de adresare a resurselor cu prezentarea elementelor specifice şi a avantajelor acestora [56], [57].
- S-a propus un mod de proiectare a ierarhiei resurselor algoritmice ale unei aplicaţii Web pentru a implementa principiile arhitecturale REST, ceea ce conduce, mai departe, la o structurare mai bună a resurselor, la creşterea gradului de uniformitate a modului de adresare a resurselor şi la utilizarea cu avantaje maxime a capabilităţilor protocolului HTTP [57].
- S-a propus o modalitate de adaptare a ierarhiei virtuale a resurselor, utilizată la nivelul superior, cel al aplicaţiei complexe bazată pe o multitudine de resurse, cu ierarhia reală a resurselor de pe nivelurile inferioare, fie resurse nou create, fie cele moştenite din sistemele existente anterior, pe principiul dezvoltării sistemelor complexe cu păstrarea compatibilităţii cu sistemele existente [60].
- S-a propus şi s-a definit *Arhitectura RESTful Multilayer*, orientată pe resurse, în care fiecare strat are rolul său specific în organizarea ierarhică şi adresabilitatea resurselor utilizate într-o aplicaţie Web, toate acestea având la bază principiul interfeţelor uniforme pentru accesul la resurse [58].
- S-a prezentat un mod de implementare a clienţilor RESTful care să utilizeze capabilităţile protocolului HTTP şi să beneficieze de organizarea resurselor unei aplicaţii complexe cu arhitectură *RESTful Multilayer* [59].
- S-a abordat problema fiabilităţii serviciilor Web în contextul utilizării lor pentru dezvoltarea unor aplicaţii bazate pe principiile arhitecturilor orientate pe servicii (*SOA – Service Oriented Architectures*).
- S-au analizat şi evidenţiat conceptele de *disponibilitate*, *siguranţă* şi *fiabilitate* şi s-au prezentat detalii despre taxonomia termenilor folosiţi în domeniul fiabilităţii sistemelor: *serviciu corect*, *eroare*, *eşec al sistemului*, *defecte*, *toleranţă la defecte*.
- S-a prezentat tehnica de *replicare* folosită cu scopul de a realiza sisteme tolerante la defecte şi s-au analizat tehnicile tradiţionale de replicare cu caracteristicile lor. S-a analizat conceptul de *replicare optimistă* şi s-au prezentat caracteristicile sale.

- S-au definit elementele esenţiale ale unui model de sistem distribuit care conţine, în esenţă sa, procese şi conexiuni între procese şi, în care s-au formalizat comunicaţiile între procese. S-a formulat un model de sistem distribuit care va constitui baza unui sistem cu replicare.
- S-au formulat o serie de *obiective ale sistemului cu replicare*, care au influenţat ulterior toată dezvoltarea acestuia, în ceea ce priveşte asigurarea fiabilităţii. În sistemul propus *toleranţa la defecte* este asigurată, în esenţă, de replicarea datelor pe mai multe noduri de reţea. La nivelul sistemului distribuit au fost luate în considerare atât căderile nodurilor reţelei, cât şi întreruperea legăturilor dintre noduri. S-au prezentat modurile de operare ale sistemului propus, precum şi tranziţiile între acestea.
- În dezvoltarea sistemului tolerant la defecte, s-a propus separarea *componentelor de management* (care formează un subsistem de management) şi a *componentelor cu rol de execuţie* (care formează un subsistem de execuţie). Avantajele sunt semnificative în cazul localizării acestor componente pe sisteme hardware separate.
- S-au analizat în detaliu rolurile componentelor de management şi a celor de execuţie şi s-a arătat necesitatea folosirii în cadrul sistemului distribuit a unor componente complexe formate din *servicii Web stateless* şi *resurse stateful*.
- S-au evidenţiat *operaţiile necesare în cadrul unui sistem cu replicare manageriată* pentru desfăşurarea în bune condiţii a replicării datelor, dar mai ales pentru resincronizarea replicilor în procesul de recuperare după întreruperea unor servicii datorită unor defecte fie în nodurile reţelei, fie la nivelul legăturilor dintre noduri.
- Pentru sistemul cu replicare propus în această teză s-au prezentat în detaliu protocoalele şi fazele de execuţie, respectiv: *protocolul componentei de execuţie* cu *fazele de execuţie* corespunzătoare şi *protocolul componentei de management* cu *fazele de execuţie ale componentei de management*, diferenţiat pe *moduri de operare ale sistemului*.
- S-a propus *o arhitectură de sistem distribuit, tolerant la defecte*. Au fost definite componentele sistemului şi s-a precizat rolul fiecărei componente şi interacţiunile dintre ele. Sistemul transpune în practică un model abstract de sistem definit anterior, cu procese de replicare coordonate de o componentă de management separată.
- S-a abordat problema securităţii sistemului distribuit, ca o componentă importantă a conceptului de *fiabilitate*. Pentru arhitectura de sistem cu replicare manageriată s-a studiat problema difuzării mesajelor de către componenta de management către serviciile pe care le coordonează. În scopul protejării sistemului de atacuri informatice a fost analizată *problema securizării mesajelor difuzate* în cadrul unui sistem tolerant la defecte, care foloseşte replicarea operaţiilor.
- S-a propus *un protocol pentru securizarea mesajelor* folosind o tehnică de protejare a informaţiei sensibile cu un *lacăt informatic*, iar pentru implementarea algoritmului de protejare s-au folosit polinoamele de interpolare Lagrange.

- S-a propus *un nou parametru al sistemelor distribuite tolerante la defecte* care să exprime influenţa fazei de replicare a serviciilor Web asupra timpului de răspuns, numit *Relative Delay*. Această mărime relativă permite evaluarea modificărilor performanţei sistemului faţă de valori de referinţă.
- Abordările experimentale au arătat o influenţă a fazei de replicare asupra performanţei globale a sistemului, dar valorile întârzierilor sistemului pot fi menţinute la valori acceptabile. Monitorizarea evoluţiei în timp a valorilor acestui parametru permite aprecierea scăderilor de performanţă a sistemului faţă de o stare iniţială şi deschide noi posibilităţi de control şi optimizare [62], [63].

BIBLIOGRAFIE

[1] IETF, Network Working Group, RFC 5023, *The Atom Publishing Protocol*, 2007, http://www.ietf.org/rfc/rfc5023.txt

[2] Avizienis, A., Laprie, J.C., Randell, B., Fundamental Concepts of Dependability, In: *Research Report no. 1145*, LAAS-CNRS, 2001.

[3] Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C., Basic concepts and taxonomy of dependable and secure computing, In: IEEE Transactions on Dependable and Secure Computing, vol.1, no.1, pp.11-33, 2004.

[4] Berners-Lee, T., W3C, Universal Resource Identifiers - Axioms of Web Architecture, 1996, http://www.w3.org/DesignIssues/Axioms

[5] Booth, D., Hass, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., Web Services architecture, http://www.w3.org/TR/ws-arch

[6] Brisco, T. (1995). DNS support for load balancing, RFC 1794, Available at http://www.faqs.org/rfcs/rfc1794.html

[7] Brusilovsky, P., Nijhawan, H., A Framework for Adaptive E-Learning Based on Distributed Re-usable Learning Activities, In: Proceedings of World Conference on E-Learning, E-Learn, Montreal, Canada, 2002.

[8] Brusilovsky, P., Adaptive and Intelligent Technologies for Web-based Education, In: Artificial Intelligence in Education, AIED'2003, Sydney, Australia, July 2003.

[9] Brusilovsky, P., A Component-based Distributed Architecture for Adaptive Web-based Education, In: Proceedings of International Conference on Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies (AIED'2004), Sydney, Australia, 2004.

[10] Brusilovsky, P., KnowledgeTree: A Distributed Architecture for Adaptive E-Learning, In: Proceedings of the Thirteenth International World Wide Web Conference, New York, 2004.

[11] Budhiraja, N., Marzullo, K., Schneider, F.B., Toueg, S., The Primary-Backup Approach, In: Distributed Systems, editor S.J. Mullender, chapter 8, pp.199-216, ACM Press, Addison-Wesley, 2nd Edition, 1993.

[12] Chandra, T. D., Toueg, S., Unreliable failure detectors for reliable distributed systems. In: Journal of the ACM (JACM), Vol.43, Issue 2, Mar.1996, NY, USA, pp. 225–267.

[13] Chang, W.J., Chen, H.H., Su, H.K., Chu, Y.S., Chen, K.J., Design of Knowledge Management Learning System based on Service Oriented Architecture, In: Proceedings of the Sixth International Conference on Advanced Learning Technologies, ICALT'06, IEEE, 2006.

[14] Chiou, G.H., Chen, W.T., Secure Broadcasting Using the Secure Lock, In: IEEE Transactions on Software Engineering, Vol.15, No.8, pp. 929-934,

Aug.1989.

[15] Cristian, F. (1991). Understanding fault-tolerant distributed systems, In: Communications of the ACM, Vol.34, Issue 2, pp. 56-78, 1991.

[16] Dahlin, M., Chandra, B.B.V., Gao, L., Nayate, A., End-to-end WAN service availability. In: IEEE/ACM Transactions on Networking , Issue 2, Vol.11, NJ, USA, April 2003.

[17] Davis, D., Karmarkar, A., Pilz, G., Winkler, S., Yalcinalp, U., Web services reliable messaging (WS-Reliable Messaging) version 1.2, 2009, Available at http://docs.oasis-open.org/ws-rx/wsrm/200702

[18] Erl, T., SOA. Concepts, Technology and Design, Prentice Hall, 2005.

[19] Erl, T., SOA. Principles of Service Design, Prentice Hall, 2007.

[20] Evjen, B., Hanselman, S., Muhammad, F., Sivakumar, S., Rader, D., Professional ASP.NET 2.0, Wiley Publishing, 2006.

[21] Fang, C.L., Liang, D., Lin, F., and Lin, C.C., Fault tolerant Web services, In: Journal of Systems Architecture no.53, Issue 1, pp. 21–38, January 2007.

[22] Farmer, R., Hughes, B., Pattern-Based End-UserDevelopment with Learning Objects, In: Proceedings of the Sixth International Conference on Advanced Learning Technologies, ICALT'06, IEEE Proceedings, 2006.

[23] Fielding, R.T., Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation, University of California, Irvine, 2000.

[24] Fielding, R.T., Taylor, R.N., Principled Design of the Modern Web Architecture, Information and Computer Science, University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf

[25] Fischer, M. J., Lynch, N. A., Paterson, M. S., Impossibility of distributed consensus with one faulty process, In: Journal of the ACM (JACM), Volume 32, Issue 2, pp.374–382, NY, USA, 1985.

[26] Foster, I., Frey, J., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W., Weerawarana, S., (Computer Associates, Fujitsu Laboratories, Globus, Hewlett-Packard, IBM) Modeling Stateful Resources with Web Services, vers.1.1, Univ. Chicago, USA, 2004.

[27] Frolund, S., and Guerraoui, R., CORBA fault-tolerance: why it does not add up, In: The Seventh IEEE Workshop on Future Trends of Distributed Systems, pp. 229, Tunisia, South Africa, 1999.

[28] Garrett, J.J., AJAX: A New Approach to Web Applications, Adaptive Path, 2005, http://www.adaptivepath.com/publications/essays/archives/000385.php

[29] Gibbs, M., Wahlin, D., Professional ASP.NET 2.0 AJAX, Wiley, 2007

[30] Gregorio, J., XML.com, 2004, How to Create a REST Protocol, http://www.xml.com/pub/a/2004/12/01/restful-web.html

[31] Guthrie, S., Url Rewriting with ASP.NET, 2007, http://weblogs.asp.net/scottgu/archive/2007/02/26/tip-trick-url-rewriting-with-asp-net.aspx

[32] W3 Consortium, RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[33] IonicShade, ISAPI Rewrite Filter, 2007, http://cheeso.members.winisp.net/IIRF.aspx

[34] Qwerksoft.com, IIS Rewrite, http://www.qwerksoft.com/products/iisrewrite/

[35] HeliconTech, ISAPI Rewrite, http://www.isapirewrite.com/

[36] Karampiperis, P., Sampson, D., Designing Learning Services: From Content-based to Activity-based Learning Systems, ACM International World Wide Web Conference, Japonia, 2005.

[37] Lamport, L., Time, Clocks, and the Ordering of Events in a Distributed System. In: Communications of the ACM, Vol.21, Issue 7, pp.558-565, July 1978.

[38] Lamport, L., Using Time Instead of Timeout for Fault-Tolerant Distributed Systems, In: ACM Transactions on Programming Languages and Systems, Vol.6 Issue 2, pp.254-280, USA, April 1984.

[39] Marguerie, F., ASP.NET HTTP module for URL redirections, 2006, http://weblogs.asp.net/fmarguerie/

[40] Mitchell, S., A Look at ASP.NET 2.0's URL Mapping, 2007, http://aspnet.4guysfromrolla.com/articles/011007-1.aspx

[41] Moser, L.E., Melliar-Smith, P.M., and Zhao, W., Building dependable and Secure Web Services, In: Journal of Software, vol.2, no.1, pp.14-26, Febr. 2007.

[42] Onion, F., RedirectModule, 2004, http://www.pluralsight.com

[43] Osrael, J., Froihofer, L., Gladt, M., Goeschka, K.M., Adaptive voting for balancing data integrity with availability. In: On The Move To Meaningful Internet Systems 2006: Confederated International Workshops Proceedings, vol. 4278 of LNCS, pp 1510-1519, Springer, 2006.

[44] Pedone, F., Boosting system performance with optimistic distributed protocols, In: IEEE Computer, Vol. 34, Issue 7, pp.80–86, 2001.

[45] Richardson, L., Ruby, S., RESTful Web Services, O'Reilly Media, 2007.

[46] IETF, Network Working Group, RFC 1831, RPC: Remote Procedure Call Protocol Specification Version 2, 1995, http://www.ietf.org/rfc/rfc1831.txt

[47] Saito, Y., Shapiro, M., Optimistic replication, In: ACM Computing Surveys, Volume 37 Issue 1, pp. 42-81, March 2005.

[48] Shamir, A., How to share a secret, In: *Communications of ACM*, Vol. 22, Issue 11, pp. 612-613, Nov. 1979.

[49] Snell, J., IBM, Resource-oriented vs. activity-oriented Web services, http://www-128.ibm.com/developerworks/xml/library/ws-restvsoap/ , 2004.

[50] W3C, Simple Object Access Protocol (SOAP), 2007, http://www.w3.org/TR/soap/

[51] W3C, Simple Object Access Protocol version 1.2, Part 1: Messaging Framework (Second Edition), aprilie 2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[52] Suryanarayanan, K., Christensen, K., Performance evaluation of new methods of automatic redirection for load balancing of Apache Web servers distributed in the Internet, In: IEEE 25th Conference on Local Computer Networks, pp. 644–651, 2000.

[53] Toader, C., Studiu comparativ privind stadiul actual al serviciilor de reţea în sistemele de operare. Securizarea autentificării şi traficului în reţea, Raport de cercetare pentru doctorat, Universitatea Politehnica din Timişoara, 2005.

[54] Toader, C., Management of Web applications performance, In: International Conference on Sustainable Development Strategies, Baia Mare, Romania, 2006.

[55] Toader, C., Petrovan, A., Costea, C., Bussiness Models and Secure Web Learning on Public Key Infrastructures, In: Proceedings of International Multidisciplinary Conference, North University of Baia Mare, Romania, 2007.

[56] Toader, C., Impact of modern WEB technologies on e-learning platforms, In: Creative Mathematics and Informatics, nr.17, ISSN 1584-286X; 1843-441X, 2008.

[57] Toader, C., Actual aspects concerning the architecture of Web-based Learning Systems, In: Proceedings of International Conference on Sustainable Development Strategies, North University of Baia Mare, Romania, 2008.

[58] Toader, C., Multilayer Resource-Oriented Architecture Supporting Restful And Non-Restful Resources, In: Proceedings of the 20th International DAAAM Symposium, Vol.20, pp.467-468, ISBN: 978-3-901509-70-4, Vienna University of Technology, 2009.

[59] Toader, C., Studiu privind tehnologiile actuale de generare a paginilor Web. Creşterea gradului de interactivitate al aplicaţiilor de tip Web-based Learning, Raport de cercetare pentru doctorat, Universitatea "Politehnica" din Timişoara, 2009.

[60] Toader, C., Studiu privind posibilităţi de realizare a aplicaţiilor distribuite pentru instruire asistată de calculator. Definirea şi implementarea arhitecturii RESTful Multilayer pentru aplicaţii Web-based Learning, Raport de cercetare pentru doctorat, Universitatea "Politehnica" din Timişoara, 2009.

[61] Toader, C., RESTful Multilayer Architectures for Web-based Learning Systems, In: Creative Mathematics and Informatics, nr.18, ISSN 1584-286X; 1843-441X, 2009.

[62] Toader, C., Increasing Reliability of Web Services, In: Journal of Control Engineering and Applied Informatics, Vol.12, No.4, pp.30-35, ISSN 1454-8658, Dec.2010.

[63] Toader, C., Reliability and Fault-Tolerance in Distributed Systems, In: Proceedings of International Multidisciplinary Conference, IMC 2011, 9th Edition, University of Baia Mare, Romania, and University College of Nyiregyhaza, Hungary, pp.257-262, ISBN 978-615-5097-18-8, May 2011.

[64] OASIS (Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, UDDI Version 3.0.2, 2004.

[65] Vesperman, J., Essential CVS, 2nd Edition, O'Reilly Media Inc., CA, USA, 2006.

[66] Weinert, A., Bandt, T., UrlRewritingNet, 2006, http://urlrewriting.net/

[67] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G., Understanding replication in databases and distributed systems, In: Proc. of 20th International Conference on Distributed Computing Systems (ICDCS'2000), Taipei, Taiwan, ROC, pp.464-474, 2000.

[68] W3C, Web Services Description Language (WSDL) Version 2.0, http://www.w3.org/TR/2007/REC-wsdl20-20070626/, Part 1, 2007.

[69] W3 Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008, http://www.w3.org/TR/2008/REC-xml-20081126/

[70] Charles Cook, XML-RPC.NET Specification, http://www.xml-rpc.net/

[71] Dave Winer, XML-RPC Specification, 1999, http://www.xmlrpc.com/spec

[72] Yu, H., Vahdat, A., Design and evaluation of a continuous consistency model for replicated services. In: 4th Symposium on Operating System Design and Impl. (OSDI'00) , pp.305–318, San Diego, CA, USA, 2000.

[73] Yu, H., Vahdat, A., The costs and limits of availability for replicated services. In: 18th Symposium on Operating Sys. Principles (SOSP) , pp.29–42, Lake Louise, AB, Canada, 2001.

[74] Yu, H., Vahdat, A., Minimal replication cost for availability. In: Proceedings of 21st Symp. on Principles of Distributed Computing (PODC'02) , pp.98–107, NY, USA, 2002.

[75] Yu, H., Vahdat, A., The costs and limits of availability for replicated services. In: Journal of ACM Transactions on Computer Systems, Vol. 24, Issue 1, pp.70-113, NY, USA, 2006.