EXTINDEREA MODELULUI ARHITECTURII SISTEMULUI DE GESTIUNE A REȚELELOR DE TELECOMUNICAȚII

ş.l. dr. ing. Liliana DOBRICĂ

Universitatea POLITEHNICA din Bucureşti

Rezumat: Această lucrare este dedicată extinderii modelului arhitecturii sistemului pentru gestiunea rapoartelor problemelor din reţelele de telecomunicaţii. Posibilitatea realizării unui sistem de gestiune a rapoartelor problemelor din reţelele de telecomunicaţii, executabil pe orice platformă (Unix, Solaris, Windows) datorită tehnologiei Java, elimină dependenţa de produsele de comerciale de operare şi reduce costul de dezvoltare. Luând în consideraţie această posibilitate, arhitectura sistemului de gestiune este adaptată tehnologiei Java şi se realizează un sistem distribut, accesibil pe o reţea bazată pe TCP/IP, care îndeplineşte funcţiile de gestiune cerute.

Cuvinte cheie: sisteme distribuite, gestiune reţelele de telecomunicaţii, arhitectura client/server, socket, Java, WWW.

1. Introducere

Procesul de extindere a modelului arhitecturii aplicaţiei de gestiune este unul iterativ. Scopul este obţinerea unui model care să realizeze funcţionalităţile unui sistem pentru gestiunea standardizată a reţelelor, în particular, a reţelelor de telecomunicaţii [3]. Pe lângă aceste cerinţe, se doreşte ca modelul final să fie independent de platformă astfel încât serviciul de gestiune a reţelei de telecomunicaţii pe care-l oferă să fie disponibil în diferite medii de utilizare.

Modelul arhitecturii cu linie de comandă cere utilizatorului uman introducerea manuală a mesajelor de comandă [4]. Dezavantajele acestui model rezultă din complexitatea sintaxei mesajului de comandă şi din imposibilitatea de a-i oferi utilizatorului un ghid pentru realizarea funcţionalitatilor pentru care a fost concepută aplicaţia de gestiune. Din această cauză, în forma cu linie de comandă, operatorul nu poate decât să testeze dacă la introducerea unui mesaj de comandă corect obţine un anumit răspuns, într-o forma destul de criptică. De aceea, se pune problema extinderii acestui model cu o interfaţă grafică pentru utilizator. În [1][9] se face o analiză a caracteristicilor unei interfeţe grafice. Se consideră, aici, că interfaţa grafică este modalitatea de prezentare a obiectelor ce se gestionează şi se menţionează că funcţionalitatea reală a obiectelor asociate se află în spatele acestei forme de prezentare.

Figura 1 prezintă o soluţie la problema extinderii. Cele două entităţi (Manager şi Agent) ce comunică prin protocolul de gestiune comună a informaţiilor, CMIP cu TCP/IP, prin intermediul platformei de gestiune distribuită, HP-OV DM, se extind cu câte un modul de comunicaţii de tip *server*. Serverul are rolul de a permite conectarea prin socket a unor module de interfaţă grafică. Interfaţa grafică are avantajul că oferă utilizatorului un mediu mult mai prietenos pentru introducerea comenzilor şi primirea răspunsurilor. Astfel, interfaţa grafică are rolul de a:

- ascunde sintaxa complexă şi criptică a formulării mesajelor de comandă, necesare emiterii cererilor pentru efectuarea funcţiilor CMIS;
- interpreta mesajele răspunsurilor şi ale notificaţiilor primite după efectuarea unor funcţii CMIS;
- oferi utilizatorului informaţiile necesare operării asupra sistemului de gestiune astfel încât acesta să-şi îndeplinească rolul pentru care a fost proiectat;
- ghida utilizatorul în operarea asupra sistemului.

Cele două interfeţe grafice GUI-M, asociată managerului, şi GUI-A, asociată agentului, sunt extinse cu câte un modul, Client – M, respectiv, Client – A, pentru a putea comunica mesajele între acestea şi aplicaţiile dezvoltate în C++ sub un mediu de dezvoltare particular Managed Object Toolkit (MOT).

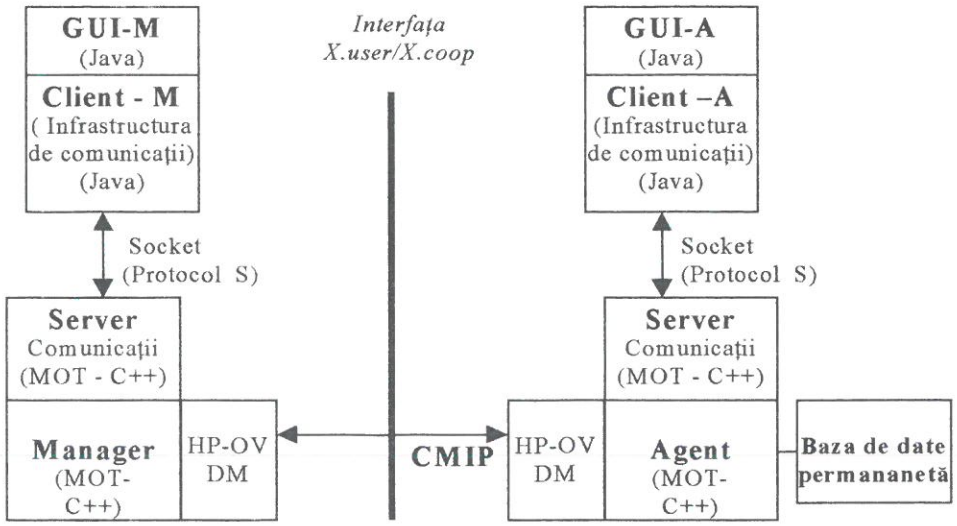

Figura 1. Extinderea modelului arhitecturii aplicaţiei de gestiune cu interfaţă grafică

Din punct de vedere al integrării în tipurile de modele de comunicaţii posibile, existente la ora actuală, prezentate în [2], [11], infrastructura de comunicaţii asociată modulelor de interfaţă grafică utilizator are rol de client, iar modulul de comunicaţii asociat aplicaţiei MOT-C++ are rol de server. Comunicaţia prin socket permite modularizarea aplicaţiei de gestiune şi o independenţă relativă de platforma distribuita de gestiune HP-OV DM.

Apariţia noilor tehnologii de tip Java [8], [13] permite dezvoltarea unor aplicaţii independente de platformă, ele putând fi executate în aceleaşi condiţii şi dacă sunt încărcate printr-un browser WWW. Avantajele multiple aduse de aceste tehnologii precum şi exploatarea Internet-ului [10] şi a serviciilor pe care această reţea le asigură se iau în considerare la stabilirea modelului extins al arhitecturii aplicaţiei de gestiune prezentat în figura 2.

Începutul lucrării reprezintă o analiză a caracteristicilor comunicaţiilor bazate pe TCP/IP şi a mediului de comunicaţie, disponibil în tehnologia Java prin J.D.K.1.1. Urmează descrierea componentelor modelului arhitecturii extinse, bazat pe Java. Partea finală se ocupă cu dezvoltarea infrastructurii de comunicaţie de la agent şi de la manager.

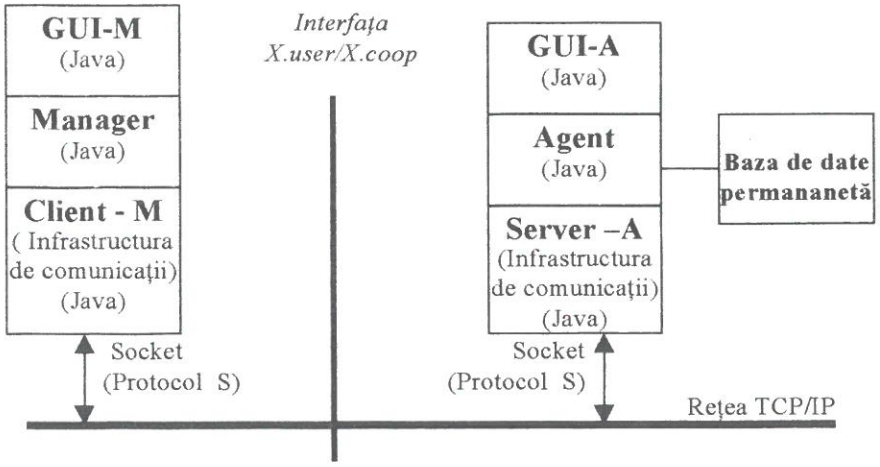

Figura 2. Model extins bazat pe tehnologia Java

2. Reţeua Internet bazată pe TCP/IP

În [10] se defineşte *Internet*-ul ca o colecţie de mai multe reţele cu grad de conectivitate larg, bazată pe protocolul nivelului reţelei. Internet este de fapt o un sistem de interconectare deschis, format din reţele TCP/IP, care folosesc adresele de IP şi protocoalele sub controlul jurisdicţional al Societăţii Internet.

TCP/IP este modelul stivei de interconectare implementată pe internet. Conform acestui model, funcţiile de interconectare sunt împărţite în cinci niveluri conceptuale şi anume, nivelul aplicaţiei, nivelul de transport (TCP), nivelul reţelei (IP), nivelul legăturii de date (de exemplu, frame-uri Ethernet), nivelul fizic (tensiuni electrice). Fiecare nivel comunică numai cu nivelurile superior şi inferior lui din stivă. Această arhitectură de stivă se bazează pe *încapsularea* şi *decapsularea* datelor ce se comunică. Încapsularea datelor este procesul încorporării pachetelor fiecărui nivel în structura pachetelor corespunzătoare nivelului inferior şi are loc pe maşina care transmite datele. Decapsularea este procesul invers încapsulării, în care, pe măsură ce datele recepţionate urcă în stiva de protocoale, sunt eliberate de structura adiţională de informaţii din pachetele corespunzătoare nivelurilor inferioare. O prezentare detaliată a formatelor pachetelor de pe fiecare nivel apare în [14].

IP este protocolul nivelului 3 din stivă (al reţelei) cu rol în adresarea maşinilor gazdă şi rout-area pachetelor de date între acestea. Principalele elemente ale protocolului IP sunt:

- adresele IP care au forma *x.x.x.x.*, unde *x* este un octet. Grupul de adrese IP grupate în scopuri administrative formează o reţea;
- subreţele IP care reprezintă diviziuni logice ale unei reţele şi constau din adrese ale reţelei IP. Fiecare subreţea are propria adresă de broadcast;
- ARP (Address Resolution Protocol) folosit de IP să găsească adresele hardware asociate unei adrese IP date;
- datagrama IP ce este formată din header şi unitatea de date a protocolului (PDU- Protocol Data Unit). Header-ul conţine informaţii despre fragmentare, lungime, timp de viaţă (TTL), iar PDU conţine datele ce sunt încapsulate şi pot fi segmente TCP sau pachete UDP;
- rout-area IP: din punct de vedere conceptual, reţelele IP se bazează pe maşini gazdă şi router-e. Maşinile gazdă implementează stivele IP şi se ocupă cu controlul fluxului de transmitere, verificarea erorilor (dacă există) şi alte prelucrări de date. Router-ele au rolul doar de a găsi o cale optimală prin reţea.

2.1. Analiza caracteristicilor TCP – protocolul de control al transmisiei

TCP este protocolul nivelului de transport şi face parte din categoria protocoalelelor de transport, orientate pe conexiune. El se situează deasupra protocolului IP. Caracteristici ale protocolului:

- multiplexarea (porturi) – O maşină gazdă dată poate susţine simultan mai multe conexiuni bazate pe TCP datorită faptului că nivelul TCP realizează multiplexarea şi demultiplexarea canalului de comunicaţie pe baza numerelor porturilor. Figura 3 prezintă mecanismul de multiplexare care permite ca aplicaţiile FTP, Telnet, SMTP şi HTTP, ce sunt servicii ale TCP/IP, să aibă numere de porturi diferite pentru comunicaţie;
- transmiterea este unică şi sigură – fiecărui segment pe care TCP îl transmite i se asignează câte o secvenţă de numere. La capătul celălalt al canalului de comunicaţie, se efectuează suma de control şi, dacă lipsesc date, îl informează pe cel care a transmis să-i retransmită mesajul. Dacă datele nu sunt corupte, nodul receptor trimite mesajul de confirmare, numit ACK şi TCP distruge automat segmentele duplicate din reţea astfel încât nodul receptor să primească un singur mesaj corect;
- pornirea lentă conduce la mecanismul de control al fluxului de date al TCP. Cauza principală a pierderii pachetelor în reţea este congestia. Această problemă apare dacă pachetele sosesc la router prea repede şi acesta, pentru a preveni umplerea bufferului, trebuie să le ignore pe unele dintre ele. TCP foloseşte un algoritm de pornire lentă care, iniţial, limitează lungimea de bandă a noii conexiuni astfel încât să nu se transmită cantităţi excesive de date dacă reţeaua este foarte ocupată;
- controlul fluxului de date permite TCP să ajusteze viteza de transmisie a nodului transmiţător evitând astfel blocarea bufferului la nodul receptor. Aceasta se realizează printr-o fereastră de transmisie de lungime variabilă, care este ajustată continuu, la ambele capete ale canalului de comunicaţie, pe baza mesajelor de confirmare;
- transmisie full-duplex – prin conexiunile TCP se poate realiza simultan transmiterea şi recepţia datelor, fapt ce câştigă timp faţă de o conexiune half-duplex ce foloseşte semnale dus-întors.

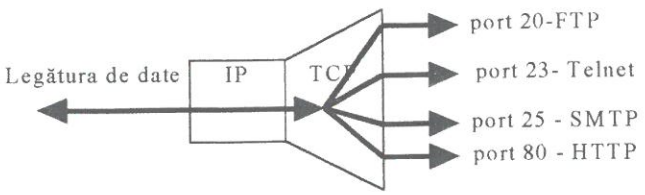

Figura 3 Multiplexarea TCP

2.2. Porturi şi socket-uri în Java

Un socket reprezintă o conexiune TCP pe reţea TCP/IP stabilită între două aplicaţii pentru ca acestea să poată comunica între ele. El este specificat prin adresele de IP şi numerele de porturi de la ambele capete ale conexiunii. După autorul lucrării [12], în modelul de comunicaţie client-server, în modulul server se execută un proces care ascultă pe un anume port. În Java, folosind socket-ul, clientul poate stabili un canal de comunicaţie, bazat pe stream TCP, cu o maşina gazdă remote, numită *host* (figura 4). De regulă, pentru a putea comunica cu o aplicaţie de pe host, clientul trebuie să creeze un socket pe host. De câte ori se stabileşte o conexiune pe portul respectiv se creează un stream TCP între server şi client.

Un socket este creat prin specificarea:

- numelui sau adresei maşinii gazdă cu care se conectează;
- numărul portului: un întreg cu valori între 1-65535.

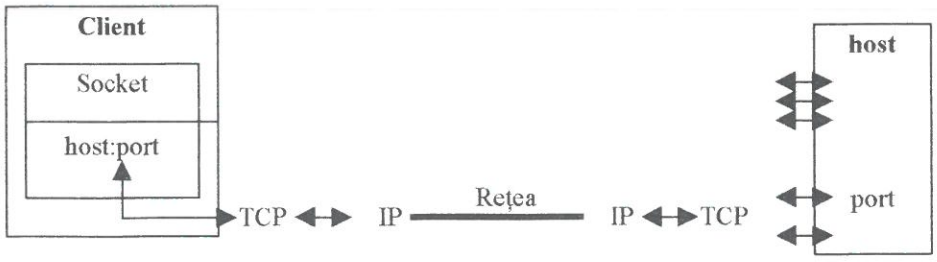

Figura 4. Comunicarea pe socket

Pentru a se stabili conexiunea pe maşina gazdă trebuie să fie activ server-ul care să asculte pe portul specificat în socket, altfel conectarea nu se va realiza. După crearea socketului, pentru a se efectua comunicaţiile prin conexiunea TCP stabilită, trebuie obţinute stream-urile (de intrare şi de ieşire).

2.3. Stream-uri în Java

Stream-urile reprezintă partea cea mai importantă a programării în reţea. Stream-ul este abstractizarea la nivel înalt reprezentând o conexiune la canalul de comunicaţii, de tipul conexiunii la o reţea TCP/IP. Cu ajutorul acestora se pot fie citi date de pe canalul de comunicaţie, fie se pot scrie date în acesta, astfel că sunt stream-uri de intrare şi stream-uri de ieşire ca în figura 5.

Figura 5. Stream-uri de intrare şi de ieşire

Figura 5 reprezintă un stream ca un capăt al unui canal de comunicaţie uni-sens. Canalul de comunicaţie conectează un stream de ieşire la stream-ul de intrare corespunzător. Astfel, orice informaţie ce se scrie în stream-ul de ieşire poate fi citită sub aceeaşi formă, pe stream-ul de intrare. Caracteristici:

- este de tip FIFO, însemnând că prima informaţie ce se scrie pe stream-ul de ieşire va fi şi prima informaţie citită de la stream-ul de intrare corespunzător;
- asigură un acces secvenţial la canalul de comunicaţie, şi nu aleator;
- permite fie numai funcţii de citire sau numai funcţii de scriere; stream-ul de ieşire scrie date în canalul de comunicaţii, stream-ul de intrare citeşte datele de pe canalul de comunicaţie; pentru a citi şi scrie pe un singur canal de comunicaţie sunt necesare două stream-uri, unul de intrare şi unul de ieşire (de exemplu, socketul).

3. Descrierea modulelor componente ale arhitecturii extinse independente de platformă

Modelul arhitecturii extinse independent de platformă conţine două module principale manager şi agent, care realizează funcţiile de gestiune ale sistemului.

Din punct de vedere al programării, ele sunt formate din clase de tip Java [7]. Modelul de programare în Java este orientat obiect. Clasele sunt implementate conform definiţiei GDMO a modelului informatic, iar specificaţia acestora este descrisă în [6]. Maparea modelului specificat GDMO pe clase Java s-a realizat astfel:

- clasele obiectelor gestionate definite au devenit clase Java ce au ca membri:
  - date de tipul atributelor conţinute;
  - constructor cu parametru sau fără şi constructor de copiere;
  - metode asociate operaţiilor pe atributele conţinute, care apelează metodele atributelor pentru transformarea acestora în format extern, corespunzător protocolului S;
  - metode asociate operaţiilor locale pe atribute.
- sintaxele atributelor definite în ASN.1 devin clase Java, care dau tipurile de date membre ale claselor obiectelor gestionate;
- atributele de tipuri simple pot fi uşor mapate pe tipuri de date Java, dar cele compuse trebuie definite conform sintaxei ASN.1;
- o interfaţă specifică trebuie implementată pentru a se putea transmite valorile atributelor prin protocolul S a cărui gramatică a fost definită în capitolul precedent. Aceasta este formată din metodele care:
  - transformă fiecare tip de atribut în format extern după sintaxa descrisă anterior;
  - setează valoarea fiecărui atribut, în funcţie de formatul extern, primit prin protocolul S.

Realizarea agentului necesită crearea şi întreţinerea bazei de date permanente. Soluţia adoptată este asemănatoare celei discutate în [5] şi, în Java, se realizează pe baza capacităţii de serializare a oricărui tip de dată definit. Dezavantajul acestei soluţii este imposibilitatea modificării conţinutului acesteia prin alte mijloace decât tot aplicaţie Java, ce conţine tipul de dată respectiv.

Baza de date permanentă are stabilit în programul principal directorul în care se construieşte. Ea este citită la pornirea aplicaţiei agent şi încărcată în memorie. Pe măsură ce agentul modifică valorile atributelor unui obiect gestionat sau creează alte obiect, gestionate ca urmare a unei cereri locale sau prin socket de la manager are loc salvarea obiectului respectiv, pe suportul permanent. Modelul de organizare a bazei de date propus impune adaptarea claselor obiectelor gestionate cu metode şi date pentru:

- indicarea şi verificarea căii fişierului în care se salvează obiectul gestionat;
- crearea obiectului din fişierul corespunzător;
- salvarea obiectului în fişierul corespunzător.

4. Infrastructura de comunicaţie

În general, aplicaţiile distribuite se pot realiza după modelul client-server. Mecanismul prin care două procese ale acestei aplicaţii comunică, poate fi realizat prin socket cu un protocol definit sau prin RPC (Remote Procedure Call). În mediile ce folosesc tehnici orientate obiect RPC este numit RMI (Remote Method Invocation).

În cazul aplicaţiei de gestiune, se optează pentru tehnica prin socket, cu protocolul S definit. Agentul are rol de server extins, cu mecanism de avansare evenimente, iar managerul are rol de client extins cu mecanism recepţie a rapoartelor de eveniment. Încadrarea celor două entităţi manager şi agent în modelul client – server s-a realizat luând în considerare funcţiile interfeţei X.user/X.coop. Modelul general de încadrare este similar celui utilizat de platforma distribuită HP-OV DM, cu diferenţa că notificaţiile pe care le primeşte managerul nu sunt confirmate.

4.1. Infrastructura de comunicaţie la agent

Agentul are rol de server. Aplicaţia principală creează şi porneşte un obiect de tip server. Caracteristicile obiectului de tip server sunt următoarele:

- este asociat unui thread;
- crearea obiectului server este echivalentă cu crearea unui socket de ascultare pe un port dat şi a listei de subscriere pentru clienţii conectaţi;
- crează un thread de aşteptare;
- în execuţie acceptă cererile de conectare ale clienţilor pe socketul creat, deschizând câte un canal de comunicaţie pentru fiecare client acceptat;
- crează şi adaugă elemente la lista canalelor de comunicaţie.

Conexiunea clientului reprezintă al doilea element principal al modelului de comunicaţie. Un obiect de acest tip este creat la fiecare cerere de conectare pe socket, acceptată de server. Caracteristicile sale sunt:

- este asociat unui thread care este activat la creare;
- primeşte cereri de la manager, cereri ce au formatul definit;
- verifică şi interpretează mesajul cererii primite;
- dacă mesajul este de cerere de conectare a unui nou manager, verifică dacă acesta este unic, prin testarea numelui contului (atributul accountName al obiectului gestionat Account);
- dacă este o cerere de execuţie, efectuează operaţia cerută şi transmite mesajul rezultatului prin aceeaşi sintaxă definit;
- în cazul în care un canal de comunicaţie se închide, ca urmare a cererii unui client sau datorită refuzării stabilirii asociaţiei cu managerul, acest obiect notifică thread-ul de aşteptare de închiderea canalului de comunicaţie şi se autodistruge.

Thread-ul de aşteptare are rolul de scoate elementele din lista canalelor de comunicaţie, atunci când acestea se închid. El este notificat de obiectul conexiunii clientului asociat canalului de comunicaţie, în momentul când acesta se autodistruge.

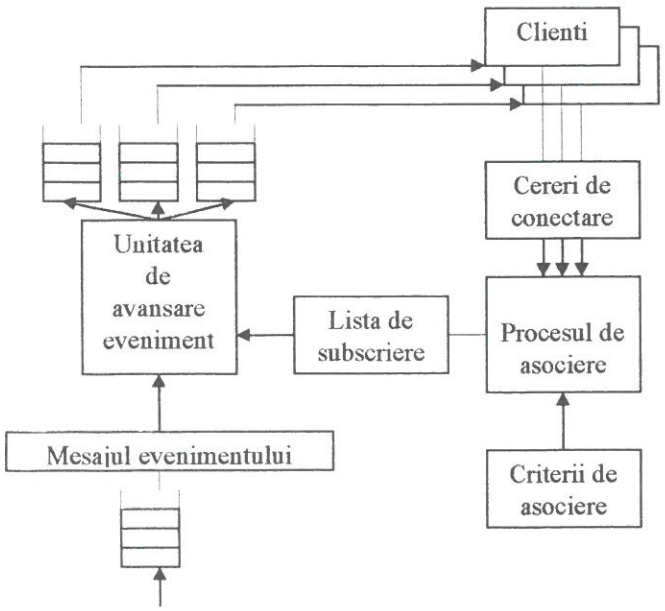

Figura 6. Modelul listei de subscriere

Lista de subscriere a clienţilor conectaţi. Reprezintă obiectul principal, necesar rezolvării funcţionalităţilor de gestiune, când sistemul este format dintr-un agent şi mai mulţi manageri. El este utilizat în *stabilirea asocierii între agent şi un manager* asociat unui cont şi în *avansarea evenimentelor* managerului corespunzător. Lista păstrează numele contului clientului (atributul *accountName* al obiectului gestionat *Account*) asociat canalului de

comunicaţie deschis. Figura 6 descrie modelul conceput pentru rezolvarea celor două procese. Modelul realizat este format din procesul de asociere, care creează lista de subscriere, pe baza cererilor de conectare şi a unor criterii de asociere şi procesul de avansare evenimente ce se realizează în unitatea de avansare a evenimentelor.

Procesul de stabilire a asocierii între manager şi agent se bazează pe lista de subscriere a clienţilor conectaţi. Se presupune că există o relaţie 1:1 între un manager (în comunicaţie cu rol de client) şi numele contului, conform diagramei de relaţii între entităţi, descrisă în [5].

La stabilirea asocierii se:

Caută în lista de subscriere a clienţilor conectaţi numele contului.
Dacă acesta există
    Se testează starea canalului de comunicaţie asociat
    Dacă este deschisă comunicaţia
        Cererea de conectare este respinsă
        Trimite mesajul de rejectare a conexiunii cu clientul
        Închide canalul de comunicaţie stabilit
    Altfel
        break
Altfel // înseamnă că nu s-a conectat nici un client pe contul respectiv
    Se introduce în lista de subscriere noul canal de comunicaţie stabilit

Procesul de avansarea a evenimentelor are loc în unitatea de avansare a evenimentelor, care are rolul de a trimite mesajele evenimentelor pe canalele de comunicaţie, corespunzătoare pe baza listei de subscripţie. El se execută dupa următorul pseudo-cod:

Caută în lista de subscriere a clienţilor conectaţi numele contului determinat
Stabileşte canalul de comunicaţie asociat
Dacă este activ
    Trimite pe canalul de comunicaţie determinat mesajul evenimentului
Altfel
    Mesajul nu este trimis

Emiterea rapoartelor de eveniment. Conform modelului informatic şi al funcţiilor interfeţei dintre manager şi agent, operaţiile care produc notificaţii sunt cele de creare a unui obiect sau de modificare a valorilor unor atribute. Crearea se realizează prin instanţierea unei clase din modelul de programare, ce reprezintă un obiect gestionat. Modificarea valorilor unor atribute are loc datorită unei comenzi locale a utilizatorului sau datorită efectuării unor operaţii interne de către agent. Mesajele rapoartelor de eveniment au aceeaşi sintaxă cu cea prezentată la protocolul S [4].

În cazul efectuării unei comenzi locale, modelul infrastructurii pentru emiterea mesajelor de eveniment se extinde cu clase ce moştenesc clasele interfeţei grafice a utilizatorului şi au caracteristica de a:

- forma mesajul rapoartelor de eveniment conform sintaxei protocolului S;
- transmite mesajul format pe canalul de comunicaţie corespunzător.

Modificarea atributelor de stare ale unui obiect gestionat de către aplicaţia agent are loc ca urmare a efectuării unei operaţii cerute de manager şi care duce la tranziţia de stare a obiectului gestionat. În acest caz şi emiterea mesajelor de eveniment se bazează pe metode specifice din clasă, ce modelează acest obiect gestionat. La apelarea acestor metode, se formează mesajul raportului de eveniment, conform sintaxei protocolului S şi se transmite pe canalul de comunicaţie corespunzător.

4.2. Infrastructura de comunicaţie la manager

Managerul este un client extins cu mecanism de recepţie a notificaţiilor. Acest mecanism se bazează pe ideea că toate răspunsurile sunt sincrone, iar notificaţiile sunt asincrone. Răspunsuri sincrone înseamnă că se aşteaptă primirea unui răspuns la emiterea unei cereri de efectuare a unei operaţii. Notificaţiile sunt asincrone, putând fi oricând primite.

O altă caracteristică a managerului, dată de funcţionalitatea aplicaţiei de gestiune, este că un manager este asociat unui obiect *Account* cu atributul *accountName* specific. Astfel, domeniul său de vizibilitate este încadrat

de subarborele din CT –ul agentului ce are ca rădăcină o instanţă a unui obiect Account cu acelaşi *accountName* ca managerul.

Aplicaţia principală creează un obiect de tip client. Caracteristicile clientului sunt:

- creează un socket *host : port* prin care se conectează la server-ul ce se execută pe maşina *host* şi care ascultă pe portul cu numărul *port*;
- trimite mesajul de cerere de asociere; dacă asocierea este acceptată de către agent, canalul de comunicaţie stabilit nu se închide şi se creează şi se porneşte un obiect de citire a informaţiilor primite pe socket;
- realizează scrierea cererilor pe socket.

Mesajul de cerere de asociere are sintaxa

{<funcţie>} (<nume cont>}

<funcţie> ::= bind

<nume cont> ::= valoare atribut accountName

Citirea informaţiilor primite prin socket se face de un obiect cu caracteristicile următoare:

- este un thread ce este trecut în execuţie de client;
- în execuţie verifică şi interpretează mesajele primite;
- se opreşte la închiderea canalului de comunicaţie.

Mesajele ce pot fi primite sunt notificaţii sau răspunsuri la cereri şi au o sintaxă definită. Pentru interpretarea acestora se aplică mecanismul descris prin următorul pseudocod:

```
se verifică şi se obţine informaţia utilă conţinută în mesaj;
dacă mesajul primit este notificare
    se prelucrează notificarea
altfel
        dacă mesajul primit este răspuns la cerere
                se pune răspunsul în buffer
        altfel
                mesaj necorespunzător
```

Prelucrarea notificărilor depinde de tipul acestora, astfel dacă este:

- notificare de creare, se creează local o instanţă a obiectului specificat în mesaj şi se afişează utilizatorului în forma grafică informaţia primită;
- notificare de modificarea valorilor unor atribute, se caută obiectul local specificat şi se înlocuiesc valorile vechi cu noile valori.

Datorită faptului că răspunsurile sunt sincrone şi pot fi multiple, se aplică *mecanismul producător – consumator*. Producătorul este thread-ul care citeşte datele primite pe socket. Consumatorul este thread-ul care, având disponibil răspunsul în buffer, îl preia din buffer şi-l prelucrează. Prelucrarea răspunsului depinde de natura acestuia.

Dacă răspunsul este multiplu (ex. cazul unei cereri de resincronizare) se aşteaptă primirea tuturor răspunsurilor, după care se transmit interfeţei grafice. În acest caz, se modifică protocolul S astfel:

- pentru cerere se foloseşte ca token de funcţie:
  <funcţie> ::= resync | tHR
- pentru răspunsuri, indicatorul de efectuare a operaţiei devine
  <flag de succes/eşec> ::= NEXT | LAST | NOT_OK

Trebuie notat că, în cazul în care arhitectura modelului este cea din figura 1, protocolul S nu se modifică pentru că se primeşte un singur răspuns, dar care conţine mai multe sintaxe concatenate. Problema tratării acestui caz se rezolvă de modulul care verifică şi identifică informaţia utilă din răspuns.

Obiectul asociat răspunsurilor este vizibil şi producătorului şi consumatorului şi este definit printr-o variabilă booleană şi un obiect ce conţine metodele pentru adăugarea şi scoaterea din buffer a răspunsurilor primite. Se prezintă, mai jos, în limbaj de programare Java, rezolvarea problemei producător-consumator:

```
public class RespArrive {
```

```
    public   boolean RespArrive=false;
    public   SocketBuffer respBuffer=new SocketBuffer();
}
```

Metoda pentru adăugarea unui răspuns în buffer:

```
public synchronized void add(Response r) {
        while (Opening.ra.RespArrive==true) {
                try {wait();}
                catch(InterruptedException e) {}
        }
        buffer=new Response(r);
        Opening.ra.RespArrive=true;
        notify();
}
```

Metoda pentru scoaterea din buffer a unui răspuns:

```
public synchronized Response  take() {
        while (Opening.ra.RespArrive==false) {
                try {wait();}
                catch(InterruptedException e) {}
        }
        Opening.ra.RespArrive=false;
        notify();
        return buffer;
}
```

Thread-ul producător:

```
{
... ...
// răspuns disponibil
Opening.ra.respBuffer.add(resp);
}
```

Thread-ul consumator:

```
{
r=new Response(Opening.ra.respBuffer.take());
... ...
// prelucrează răspunsul
}
```

4.3. Extindere cu posibilitatea de încărcare prin browser WWW

Mediul Java este proiectat să lucreze în sistem distribuit. El s-a lansat ca limbajul de programare pe Web prin posibilitatea de a construi applet-uri Java. Totuşi, el conţine toate facilităţile de programare ca aplicaţii propriu-zise.

Diferenţa principală dintre un applet Java şi o aplicaţie Java este lipsa restricţiilor de securitate în cazul aplicaţiilor. Ca aplicaţie, nu sunt restricţii de acces la resursele sistemului, astfel se poate deschide un socket de tip server, se pot deschide orice fişiere pentru citire sau scriere sau se pot crea propriile *class loaders*. Avantajul utilizării apletului este că acesta poate fi executat într-un browser; browser-ul facilitează distribuţia automată a programului. Astfel că, pentru a putea fi folosit, un program applet nu mai trebuie instalat pe platforma pe care se doreşte a fi folosit, ci se aduce prin intermediul browser-ului. Procesul întreţinerii unui program pe un număr mare de maşini este evitat în acest fel. Cele mai cunoscute browsere de Web sunt Netscape Navigator (accesibil pe diferite platforme de operare ca Windows NT sau Unix) sau Microsoft Internet Explorer (accesibil numai pe platforme Windows).

HTTP (Hypertext Transport Protocol) este protocolul WWW care foloseşte portul TCP 80. Prin HTTP se pot transfera datele specificate în fişierele cu format *html*. Pentru a indica resursele de pe WWW, inclusiv documente HTTP, se utilizează sintaxa URL (Uniform Resource Locator). Această sintaxă specifică protocolul, numele maşinii pe care se află serverul de WWW, portul de comunicaţie (80) şi numele fişierului *html*. Fişierul *html* este cel ce specifică fişierul *.class* ce reprezintă formatul *bytecode* pentru execuţia appletului pe maşina virtuală Java activată de browser-ul de WWW.

Un applet este instanţiat şi executat de către browser-ul de WWW. De aceea, accesul applet-ului la resurse de intrare/ieşire este restricţionat de caracteristicile de securitate ale acestuia. Din punct de vedere al socketului, applet-urile nu pot deschide socket decât pe maşina pe care se află codul appletului. Această restricţie de securitate impune ca aplicaţia cu rol de agent să se găsească pe aceeaşi maşină gazdă cu codul applet-ului.

5. Concluzii

În lucrare se acordă o atenţie specială procesului de mapare a modelului informatic, definit în GDMO, pe tipuri de date definite sau construite în Java. Valorile atributelor specificate în sintaxa abstractă ASN.1 sunt translatate în sintaxa protocolului S, definit ca format extern de tip *string*. Funcţiile de gestiune ale sistemului au fost mapate pe obiecte grafice, care permit introducerea şi afişarea valorilor atributelor obiectelor gestionate. Din punct de vedere al comunicaţiei, maparea acestor funcţii pe operaţii CMIS s-a realizat prin sintaxa protocolului S, iar la nivel de implementare prin extinderea claselor ce formează interfaţa grafică pentru utilizator, cu subclase ce permit construirea cererilor de tip CMIS.

Modelul extins al arhitecturii nu modifică funcţiile de gestiune specificate în documentaţie, ci introduce noi elemente dependente de tehnologia de implementare, care să permită păstrarea informaţiilor pe suport permanent astfel încât, în perioada de execuţie a aplicaţiei cu rol de agent, BDP să fie copia obiectelor din memorie. De asemenea, pentru modelul extins, bazat pe tehnologia Java, se descrie comunicaţia între agent şi manager, ce respectă arhitectura client server, cu transmiterea informaţiilor prin socket, cu sintaxa definită prin protocolul S. La agent se rezolvă problema sistemului de gestiune format dintr-un agent şi mai mulţi manageri. Lista de subscriere a clienţilor conectaţi şi emiterea rapoartelor de eveniment reprezintă elementele de bază ale soluţionării acestei probleme. Infrastructura de comunicaţie la manager are particularitatea vizibilităţii domeniului de obiecte gestionate.

Bibliografie

1. **BARTZ, TH.**: Graphical Management Interfaces. În: Network and Distributed Systems Management, Addison-Wesley Publishing Company, 1994, pp. 517-537.
2. **CROWCROFT, J.**: Open Distributed Systems, University College of London Press, 1996.
3. **DOBRICĂ, L.**: Sistem pentru gestiunea standardizată a rapoartelor problemelor din reţele de telecomunicatii, Teza de doctorat, 1998.
4. **DOBRICĂ, L., T. IONESCU**: An adaptable User Interface for a Trouble Ticketing Service. În: Proc. of the 18th IASTED International Conference in Applied Informatics (IASTED AI'2000), Innsbruck, Austria, 14-17 Feb. 2000, pp. 597-601. ISBN 0-88986-280-X, ISSN 1027-2666.
5. **DOBRICĂ, L., T. IONESCU**: Design Procedure of a Distributed System for Telecommunications Management Networks. În: Control Engineering and Applied Informatics Journal, accepted 2000.
6. **DOBRICĂ, A. D. IONIŢĂ, L., T. IONESCU**: GUI Java Integration Specification for X Interface for Trouble Ticketing and Freephone and International Leased Line Service Management, Raport de cercetare, Martie, 1998.
7. **FLANAGAN, D.**: Java in a Nutshell - A Desktop Quick Reference, Ed. O'Reilly & Associates, Inc. 1997, 2nd Edition.
8. **FLANAGAN, D.**: Exploring Java, Ed. O'Reilly & Associates, Inc. 1997, Deluxe Edition.
9. **IONESCU, T., L. DOBRICĂ**: Sistem de gestiune a resurselor Internet la distanţă - Realizarea unei interfeţe la nivel furnizor de servicii în Java, pentru servere WWW, Contract de cercetare, UPB- MCT, Nov.1997.
10. **HUGHES, M&C, M. SHOFFNER, M. WINSLOW**: Java Network Programming, Ed. Manning, 1997
11. **KRAMMER, J.**: Distributed Systems. În: Network and Distributed Systems Management, Addison-Wesley Publishing Company, 1994, pp. 47-66.
12. **TANENBAUM, A.**: Distributed Operating Systems, Prentice Hall, 1995.
13. **WUTKA, M.**: Hacking Java. The Java Professional's Resource Kit , Ed. QUE, 1997.
14. **COMER, D., D. STEVENS**: Internetworking with TCP/IP, Prentice Hall International Editions, New Jersey, 1994.