Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Universitatea Politehnica București

Arhitecturi de obiecte distribuite
Proiect Inginerie Software

Studenți: Bățroș Elena(443A)
Ciorobea Cristian Alexandru(433A)

Anul Universitar 2014-2015

Cuprins

Bățroș Elena(443A)

1. Introducere

În acest proiect vom trata 2 tehnologii de arhitecturi de obiecte distribuite. În primul capitol se va trata tehnologia CORBA, iar în partea a două va fi prezentată tehnologia OLE. Obiectele distribuite nu au nevoie doar de un limbaj, au nevoie de o întreagă arhitectura. Aceste tehnologii au nevoie de arhitecturi cu o infrastructură bună pentru a administra eficient obiectele. Arhitecturile sunt responsabile să asigure un set bun de servicii și capacități care fac obiectele distribuite cât mai simple.

2. CORBA

Object Management Group (OMG) a fost format în 1989 să se ocupe de complexitatea ridicată a dezvoltării aplicațiilor software. Acest grup creează și publică specificații care să permită interoperabilitatea și optimizează procesul dezvoltării de aplicații software standardizate pentru obiecte distribuite. În prezent, acest grup are mai mult de 700 de membri.

Grupul a creat Common Object Request Broker Architecture (CORBA). CORBA este doar o specificație, care permite vânzătorilor să pună în aplicare arhitectura în orice fel doresc.

- # Prezentare generală

Scopul principal al CORBA este de a permite interoperabilitatea între obiectele de pe sisteme distribuite. Începând cu versiunea 2.0 CORBA, furnizori diferiți pot comunica acum cu fiecare parte, prin crearea "intergalactic object bus". Astfel, CORBA oferă o arhitectura bună pentru a lucra cu obiecte distribuite pe o rețea eterogenă; Acest lucru ajută foarte mult în procesul de integrare cu aplicațiile vechi.

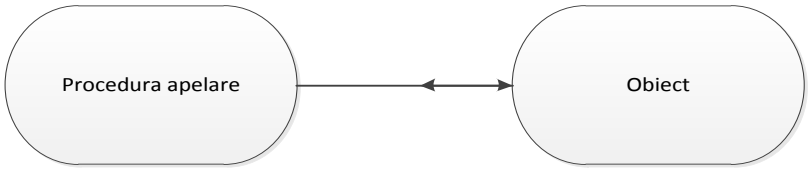


Figura 1-Metoda de invocare tradițională

CORBA permite clienților și serverelor comunicarea optimă prin protocoale de rețea și alte aspecte de comunicare. În această descriere a CORBA, "clientul" este procesul care face uz de un obiect și "serverul" este procesul care conține obiectul. Clientul nu are nevoie să știe unde este serverul; clientul poate lucra la fel ca și în cazul în care obiectul lucrează cu entități ce există în același spațiu de proces. O implementare CORBA realizează acest lucru prin crearea unei legaturi special pentru client și legături multiple către server; care inițiază si deinițiază cereri și răspunsuri. Transparența a fost un obiectiv prioritar de proiectare al CORBA.

Figurile 1 si 2 ilustrează diferența dintre o metodă de invocare tradițională și una care utilizează un ORB. Clientul și serverul nu știu unde se află cealaltă entitate; ORB se ocupă de medierea procesului.

Următoarea secțiune descrie diverse părți ale arhitecturii în detaliu.

Figura 2-Metoda de invocare utilizand ORB

- **Arhitectură**

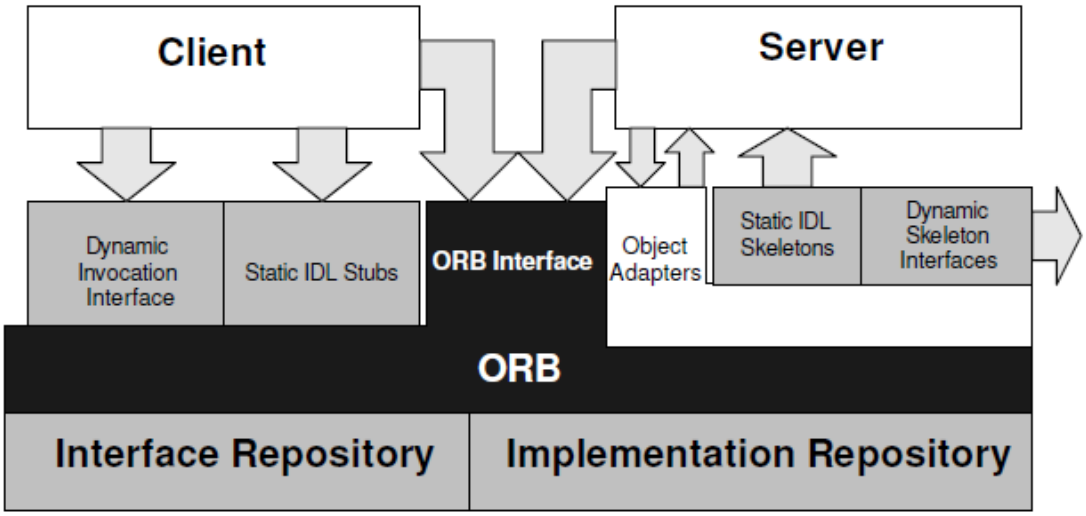


Figura 3- Arhitectura CORBA (Diagrama Mecanisme ORB)

Clientul și serverul sunt conectate prin Object Request Broker (ORB). ORB mediază întreg procesul, ținând activitatea de comunicare de la client și server. Obiectul trebuie să fie descris pentru prima dată nu în limbajul în care este pus în aplicare, ci în Interface Description Language (IDL).IDL specifică contactul între apelant și implementatorul obiectului. În acest fel implementatorul și utilizatorul obiectului nu trebuie să fie aceeași persoană. O interfață în CORBA reprezintă un grup de metode și variabile ale membrilor. Variabilele membrilor pot fi, de asemenea, read-only.

Figura 3 este o diagramă mai detaliată a mecanismelor ORB decât figura 2. Există mai multe componente vizibile la ORB în această diagramă.

- **Object Request Broker (ORB)**

ORB prevede "liantul" între client și server. Acasta conduce cereri și răspunsuri între clienți și servere. Specificația nu dictează cum să-l pună în aplicare, astfel încât nu există loc pentru concurenți; acesta poate fi o parte integrantă a sistemului de operare, o aplicație tip daemon care rulează în fundal, sau doar o pereche de biblioteci. Pentru a ajuta la sarcina sa, ORB are două părți principale - Interface Repository și Implementation Repository.

IDL este un limbaj pur declarativ care specifică interfața pe care un obiect ar trebui să o pună în aplicare și un client trebuie să utilizeze. Acesta este un superset a unui subset de C ++. Dar, din moment ce este declarativ, un utilizator nu poate scrie un program care funcționează în IDL. Declarațiile IDL se integrează într-un limbaj de programare într-un mod standard, astfel încât un client și un server pot fi "portați" pentru alte implementări CORBA, cu puține sau nici o schimbare de cod. Mapări oficiale există în prezent pentru C, C ++, Smalltalk, și Ada95. În curs sunt mapările Java și COBOL. În graba de a sari pe bandwagonul Java, unele companii au implementat deja propriile lor mapări de Java. Fișierul IDL este utilizat pentru a genera fișiere de legătură pentru client și server; inițiază parametrii și îi trimite la ORB. Astfel, clientul și serverul nu trebuie să știe informații despre locație; permițând programatorilor să se concentreze pe punerea în aplicare a obiectelor, nu de rețeaua de manipulare a acestora.

Secventa de cod de mai jos este un exemplu de fișier IDL. „Car” este o interfață. Acesta conține două atribute și o metodă. Atribute, viteza și direcția, harta pentru a obține și hotărî metode. Interfața „Police_Car” moștenește de la „Car”, ceea ce înseamnă că poate folosi aceleași funcții ca și „Car”. De asemenea, rețineți că, deoarece atributele sunt legate de metode, stabilirea „siren_on = TRUE” poate invoca o metodă. Aceasta este o modalitate convenabilă pentru un utilizator pentru a activa sirena, de exemplu. Atunci când un compilator IDL pentru C ++ le prelucrează, se produc fișiere similare cu cele din tabelul 1, de legătură între client și server.

```
interface Car {
attribute float speed;
attribute float direction;
void drive(in float distance);
};
interface Police_Car : Car {
attribute boolean siren_on;
};
```

myprog_c.hh	Fișier antet folosit pentru client
myprog_c.cc	Implementarea codului legăturii clientului
myprog_s.hh	Fișier antet folosit pentru server
myprog_s.cc	Implementarea codului legăturii serverului

Tabel 1

Figura 4 ilustrează rolul compilatorului IDL. Când un client invocă o metodă, o metodă de myprog_c este de fapt invocată. Această metodă inițiază parametrii și spune ORB să transmită către server. Scheletul serverului myprog_s apoi primește apelul, și transmite obiectului real. Răspunsul este tratat într-un mod similar.

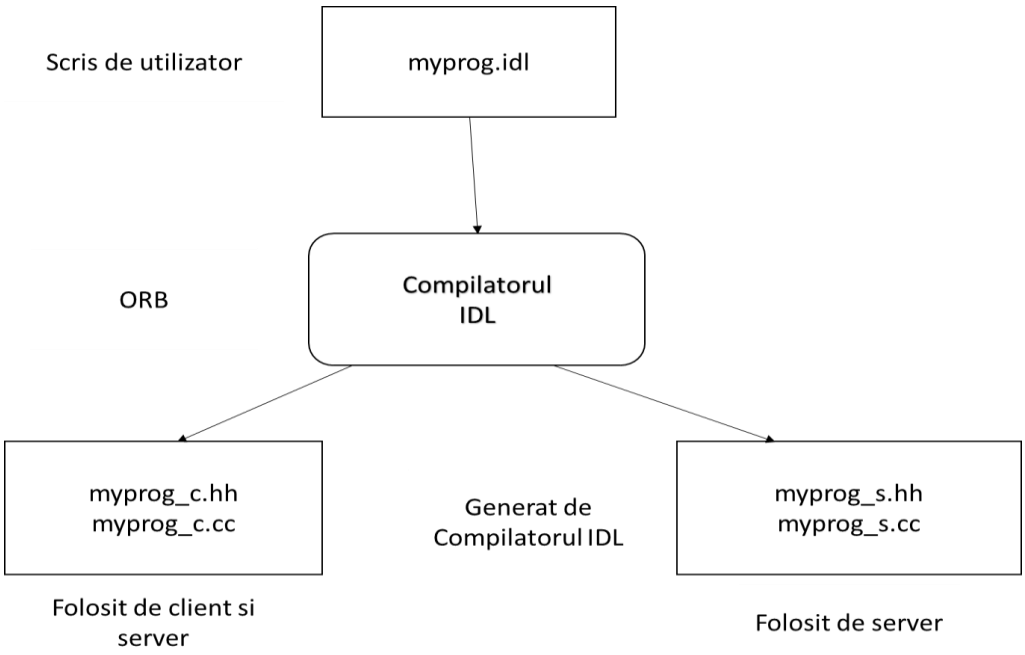


Figura 4-Rolul Compilatorului IDL

- **Object Adapters**

Sunt atât de multe moduri de a pune în aplicare și setare a obiectelor, astfel că trebuie să existe un mod standardizat pentru ORB de a lucra cu obiectele. Acest lucru este asigurat de Object Adapter, responsabil cu:

• Înregistrarea implementărilor
• Generarea și interpretarea referințelor obiectelor
• Maparea referințelor obiectelor cu implementările lor corespunzătoare
• Activarea și dezactivarea implementarilor obiectelor
• Invocarea metodelor
• Coordonarea interacțiunilor de securitate

Object Management Group a standardizat un obiect Adaptor, numit Basic Object Adapter (BOA). Există și alte adaptoare de obiecte din procesul de standardizare momentan. Detaliile Basic Object Adapter sunt dincolo de sfera de aplicare a acestei lucrări.

- **Interface Repository (IR)**

IR este o parte importantă a ORB ce conține informații tip pentru interfețe. IR necesită Dynamic Invocation Interface, ce permite invocărilor să fie construite în timp real. IR necesită o formă de stocare optimă pentru stoca interfețele. Informațiile dintr-un fișier IDL sunt suficient pentru a fi stocate în IR. Cu toate acestea, un obiect poate adăuga, modifica sau șterge o interfață în IR în timpul rulării.

De reținut că, deși Implementation Repository și Interface Repository ambele au aceleași litere inițiale, IR este folosit pentru acronimul interface Repository.

- **Implementation Repository**

Implementation Repository este o parte a ORB care conține informații pentru a localiza obiecte, precum și obiectele care sunt în prezent instanțiate. Un administrator de sistem poate înregistra obiecte manual, sau un server poate înregistra obiecte în timpul rulării. Implementation Repository este utilizat de către ORB atunci când un client dorește să se conecteze la un anumit obiect sau pentru a activa unul nou.

- **Dynamic Invocation Interface (DII)**

După cum s-a menționat mai sus, Dynamic Invocation Interface (DII) este utilizată pentru a crea invocații în timpul rulării, spre deosebire de utilizarea legaturilor client-server (rețineți că utilizarea legăturilor client-server pentru invocarea unei metode folosește Static Invocation Interface, sau SII). Acest lucru permite unui client să utilizeze o interfață pentru care nu a fost compilat. Un utilizator poate naviga astfel prin IR, vedea noi obiecte ar putea dori să le folosească, iar apoi invoca să invoce o metodă.

Utilizarea DII este mult mai complexă decât utilizarea SII. Deci, în general, DII trebuie utilizată numai atunci când este necesar; atunci când informațiile despre interfațe nu sunt disponibile. Un loc comun pentru a utiliza DII este în limbajele de scripting. Toate acestea se intampla pe partea de client; un server nu pot spune dacă o invocare a fost făcută cu ajutorul SII sau DII.

Pentru a ajuta la scripting, metodele invocate dinamic pot fi numite perechi de valori. Acest lucru se integrează bine cu mai multe limbaje macro populare,

inclusiv Microsoft Visual Basic. Astfel, este posibil pentru a apela o funcție cu ceva de genul:
Address = Lookup(ID = 12345, Name = “Jay Ongg”)

Similare cu OLE Automation, complexitatea în a face o cerere dinamică face limbajele macro cele mai bune medii pentru a utiliza DII.

- **Dynamic Skeleton Interface (DSI)**

DSI este un nou plus fata de CORBA 2.0. Acesata permite celor care implementează ORB trecerea la alte sfere sau sisteme non-CORBA. Acest lucru face DSI dă fie importantă intr-o rețea eterogenă, mai ales cu sistemele existente. Se poate vedea DSI ca un "filtru" pentru invocările de metode. Similar cu DII, un client nu poate spune dacă obiectul pe care îl folosește este accesat cu ajutorul DSI sau nu.

- **Internet Interoperability Protocol (IIOP) și Intergalactic Object Bus**

CORBA 2.0 are cerința pentru cei ce implementează ORB să sprijine interoperabilitatea cu Internet Protocol. Acest protocol funcționează peste TCP/IP și permite ORB-urlor cu standarde CORBA 2.0 să comunice între ei. Astfel, Internetul poate fi folosit ca rețeaua în care obiectul este transmis. Aceasta este ceea ce dezvoltatorii CORBA numesc "Intergalactic Object Bus". Nu mai există limite cu privire la maparea obiectelor; un client poate invoca o metodă pe un obiect la jumătatea globului la fel de ușor ca pe un obiect de pe același computer. Administratorii de sistem trebuie doar să se asigure că globurile sunt configurate corect, iar măsurile de securitate necesare sunt decise.
Când un client doreste sa comunice cu un obiect pe un alt ORB, pur și simplu comunică cu ORB local și ORB local va transmite cererea în mod corespunzător. Interfața utilizată în afara unui domeniu local va avea nevoie de un identificator care este unic în spațiu și timp. Prin urmare, o interfață tip "MyInterface" nu este suficient de bună.

CORBA permite două moduri de a identifica interfețe: printr-un nume generat uman, sau prin intermediul Universally Unique Identifier (UUID). UUID este un număr de 16 octeți generat de un algoritm pe care Open Software Foundation l-a descoperit pentru a ajuta la dezvoltarea Distributed Computing Environment.

Algoritmul se uită la ora actuală, precum și la MAC-ul cardului Ethernet al computerului pentru a garanta unicitatea. În cazul în care computerul nu are un card de Ethernet, atunci algoritmul nu va garanta unicitate, dar șansele de coliziune devin extrem de improbabile.

- **Serviciile CORBA și Object Management Architecture (OMA)**

Infrastructura definită mai sus este suficientă pentru obiecte pentru a comunica între ele. Cu toate acestea, OMG a vrut să creeze un set fundamental de interfețe care ar putea pune în aplicare furnizori pentru utilizatori. Ele sunt definite Object Management Arhitecture (OMA), și împărțit în două părți: CORBAservices și CORBAfacilities.

CORBAservices sunt încă în curs de specificare, astfel încât nu mulți producători le-au pus în aplicare încă. Acestea includ interfețe de nivel scăzut, care ar trebui să fie puse în aplicare, iar CORBAfacilities va construi pe partea de sus a acestora. Lista parțială a CORBAservices este:

• Lifecycle Service: Ofera servicii și convenții pentru crearea, ștergerea, copierea, mutarea obiectelor
•Persistent Object Service: Oferă servicii care simplifică crearea de obiecte persistente (adică obiecte care există atunci când ORB este închis și repornit)
• Externalization Serice: Oferă servicii, interfețe și protocoale, care permit ca un obiect să fie scris ca un flux de octeți.
• Naming Service: Oferă o interfață pentru un obiect să fie legat la un nume.
• Trader Service: Oferă o interfață pentru un obiect să fie legat la o anumită categorie. Trader Service poate fi privit ca "Pagini albe" și Trader Service ca "Pagini Aurii".
• Event Service: Oferă o interfață pentru un obiect de a trimite evenimente asincrone clientului său.
• Transaction and Concurrency Control Services: Ofera servicii și interfețe care permit tranzacțiile și controlul concurenței.
• Property Service: Permite utilizatorului să adauge o proprietate a unui obiect deja definit și instanțiat. O proprietate este un șir de legare la orice structură de date.
• Security Service: Ofera servicii care permit securitate în modelul de obiecte distribuite.

- **CORBAfacilities și CORBAdomains**

În timp ce CORBAservices creează o infrastructură bogată pe care un programator o poate utiliza pentru a dezvolta aplicații CORBA, o infrastructură nu este de ajuns pentru un dezvoltator de software pentru a construi componente.

OMG este în procesul dezvoltării specificațiilor CORBAfacilities, care sunt aceste obiecte la nivel de aplicație. CORBAfacilities vor fi puse în aplicare cu ajutorul CORBAservices. Aceste CORBAfacilities sunt un set de API-uri orizontale, ce sunt folosite pentru uz general. CORBAfacilities sunt împărțite în patru domenii de bază:

• User Interface: Pentru interacțiunea cu utilizatorul.
• Information Management: Pentru a gestiona schimbul de informații.
• Systems Management: Pentru gestionarea unor sisteme informatice complexe, tip multivendor
• Task Management: Pentru automatizarea proceselor de lucru.

CORBAdomains vor acoperi piețele verticale specifice. În prezent, OMG are grupuri de lucru pe interfețe pentru medicină, telecomunicații, transport, servicii financiare, producție, comerțul electronic, precum și business objects. Aceste nume sunt auto-explicative, cu excepția grupului business objects. Acest grup lucrează continuu la un standard care va modela oameni, entități, precum și orice obiect activ în domeniul de afaceri.

Ciorobea Cristian Alexandru (433A)

3. Protocolul de control de la distanta al modelului cu componente obiect distribuite (DCOM).

• Introducere

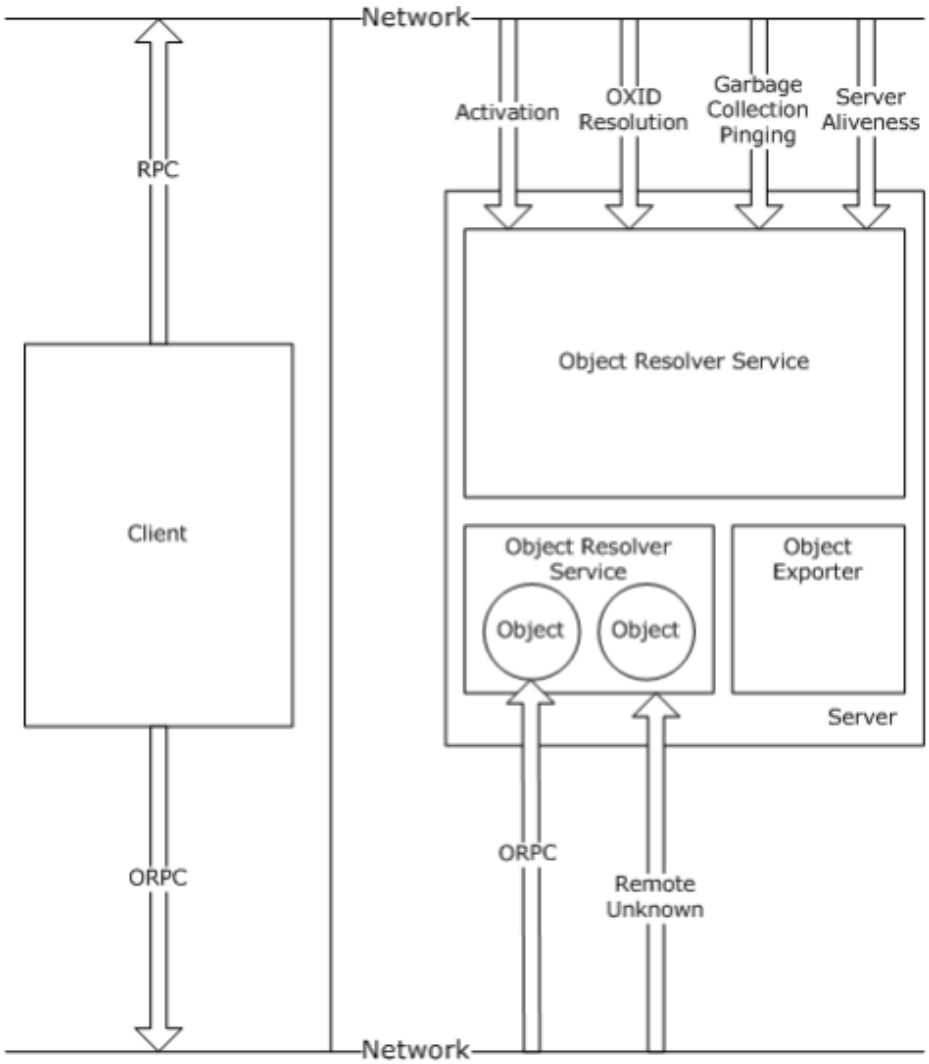
Acest protocol extinde modelul modelul componentei model (COM) prin retea prin acordarea de facilitati pentru a crea si activa obiecte, si pentru a conduce referintele obiect, durata de viata a obiectelor si interogarile interfetei obiect. Acest protocol este construit pe baza extinderii protocolului de apelare a procedurilor de la distanta si se bazeaza pe autentificarea, autorizarea si capacitatea de integrare a mesajelor a acestuia. Diagrama urmatoare arata asezarea pe nivele a protocoalelor. (RPC - Procedura de Remote Call).

Aplicatii/Protocol de nivel inalt
DCOM
RPC

[Tabelul 2 – Asezarea pe nivele a protocoalelor]

Aplicatiile de nivel inalt folosesc clientul DCOM pentru a obtine referinte obiect si pentru a face apeluri ORPC asupra obiectului. La randul sau, clientul DCOM se foloseste de extinderea protocolului de apelare a procedurilor de la distanta pentru a comunica cu serverul obiect.
Serverul obiect constituie un serviciu de decizie obiect si unul sau mai multi exportatori obiect. Obiectele sunt continute de exportatorii obiect. Obiectele sunt tintele apelurilor ORPC de la client.

Urmatoarea figura ofera o prezentare generala a protocolului.



[Figura 5 – Prezentare generala a protocolului DCOM]

- **Activarea**

Activarea este un termenl general folosit pentru descrierea actului de creere (sau uneori gasire) a unui obiect DCOM deja existent sau a unei fabrici de clase. Doua interfete RPC din protocolul de la distanta DCOM sunt folosite pentru activarea obiectelor:

-Metoda IActivation

-Metoda IRemoteSCMActivator

La un nivel rudimentar, activarea conta in trimiterea catre serviciul de activare a obiectului din unitatea remote a urmatoarelor:

-Un indentificator de clasa (**CLSID**)

-Unul sau mai multe **IID**-uri

-Optional, o referinta initiala a stocarii

CLSID-ul identifica clasa obiectului ce urmeaza sa fie creat. IID-urile identifica interfetele cerute de client din obiectul nou creat si, daca este specificat, referinta de stocare identifica un depozit persistent cu care noul obiect sa fie initializat dupa creare.

Activarea returneaza referintele obiect catre aplicatia client. Aceasta poate, de asemenea, sa trimita sau sa primeasca referinte obiect ca urmare a apelurile ORPC.

- # Referintele Obiect

Acestea sunt aranjate ca tipuri OBJREF. Cand un tip OBJREF este aranjat in protocolul de distanta al DCOM, Reprezentarea datelor de retea (Network Data Representation (NDR)) informeaza executia DCOM sa scrie un OBJREF infasurat inauntrul unui MInterfacePointer in cererea/raspunsul executiei unitatii de date protocol (PDU). Datele aranjate contin informatiile cerute de catre client pentru a crea legatura inapoi catre obiect al RPC-ului. Similar, cand un tip OBJERF este nearanjat in protocolul de distanta al DCOM, NDR informeaza executia DCOM sa construiasca referinta obiectului folosind datele aranjate continute de executie. Protocolul de distanta DCOM returneaza referinta obiect catre aplicatie.

- **Exportatorul de obiecte**

Un exportator de obiecte este uncontainer conceptual unde obiectele sunt create, apelate si eliberate. Un obiect trebuie sa fie continut de un singur exportator de obiecte si nu trebuie sa fie cuprins in mai multi exportatori de obiect. Protocolul este imprecis in a specifica cu exactitate ce presupune un exportator de obiect. Acesta poate fi un thread, un proces sau un mecanism. Clientii nu trebuie sa-si asume detalii de implementare asupra exportatorilor de obiect. De exemlu, daca doua obiecte apartin aceluiasi exportator de obiect, clientii trebuie sa presupuna ca ambele obiecte apartin in acelasi thread, proces sau mecanism.

Un exportator de obiect primeste informatii din retea cu ajutorul protocoalelor RPC.

Un exportator de obiect contine un obiect necunoscut la distanta, care sprijina urmatoarele interfete ORPC:

- *Interfata IRemUnknown*: o interfata ORPC care contine metodele folosite pentru apelul QueryInterface, AddRef si Release pe obiecte la distanta.

- *Interfata IRemUnknown2*: o interfata ORPC care extinde functionalitatea lui IRemUnknown.

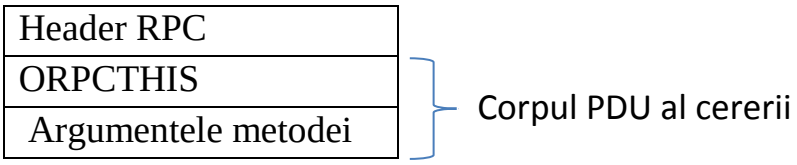
Clientul utilizeaza metodele AddRef si Release pentru a gestiona durata de viata a obiectelor continute in exportatorul de obiecte. Clientul utilizeaza metoda QueryInterface pentru a obtine referinte obiect pentru tipuri de interfete aditionale implementate de un obiect.

Un exportator de obiect este identificat printr-un identificator numit OXID. Cand un client primieste un OXID ca parte a unui obiect de referinta, trebuie sa stabileasca informatiile RPC obligatorii necesare pentru a comunica cu obiectul necunoscut la distanta al exportatorului de obiect. Clientul utilizeaza mecanismul de rezolutie al OXID pentru a realiza acest lucru.

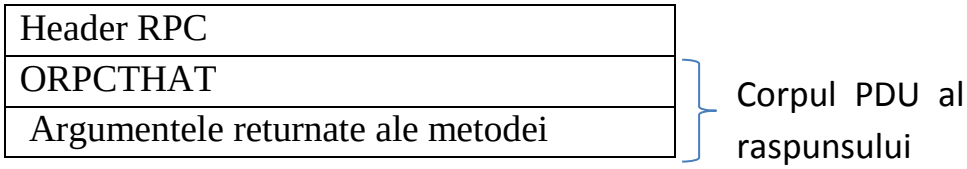
- **Apelurile ORPC**

Un apel ORPC este echivalent si detine o corespondenta unu la unu cu apelurile RPC. Apelurile ORPC se deosebesc de apelurile RPC prin continutul campului UUID al obiectului din headerul RPC. In protocolul DCOM, campul UUID al obiectului contine un pointer identificator de interfata specificand interfata tinta a unui apel ORPC asupra unui obiect.

Apelurile ORPC sunt deosebite de apelurile RPC prin faptul ca primele contin implicit de fiecare data parametrii aditionali prezenti in bufferele de cerere si raspuns pentru fiecare apel. Acesti parametrii aditionali sunt mentionati ca ORPCTHIS si ORPCTHAT. Acesti parametrii sunt pozitionati conceptual si sintactic inaintea altor valori in corpul PDU al RPC.



[Tabelul 3 - Apeluri RPC obiect si corpul PDU al cererii]



[Tabelul 4 - Apeluri RPC obiect si corpul PDU al raspunsului]

Argumentele ORPCTHIS si ORPCTHAT sunt folosite pentru a oferi versiunilor, cauzalitatea informatiei si capabilitatea de a trimite data out-of-band specifice aplicatiilor.

- **Identificatori de cauzalitate**

Fiecare apel ORPC contine cu el, in structura ORPCTHIS, un GUID cunoscut ca identificator de cauzalitate (CID). CID-ul leaga un lant de apeluri ORPC care sunt familiare. Exportatorii de obiect POT folosi CID-ul pentru a asigura sincronizarea apelurilor ORPC. De asemenea, pot folosi CID-ul pentru a preveni blocaje in apelurile ORPC.

Daca un apel ORPC nou este facut de catre un client care deja executa un apel ORPC, noului apel ii trebuie alocat acelas CID ca apelului existent. Daca un nou apel ORPC este facut de catre un client care nu executa un alt apel ORPC, atunci un nou CID trebuie alocat pentru el.

Un exportator de obiect trebuie sa utilizeze CID-ul unui apel ORPC primit pentru a detecta daca acesta apartine aceluiasi lant ca unui apel ORPC aflat in executie. Daca CID-ul apelului primit si cel trimis nu sunt identice, exportatorul de obiecte POATE sa nu proceseze apelul ORPC de primit pana cand apelul trimis nu este terminat. In orice caz, daca CID-urile sunt identice, exportatorul de obiect TREBUIE sa proceseze apelul ORPC primit, altfel, apare un blocaj.

- **Calculul referintelor**

Procotolul DCOM foloseste un calcul al referintelor pentru a organiza timpul de viata al obiectelor. Fiecare interfata dintr-un obiect are asociat un calcul de referinta care ii determina timpul de viata. Sunt doua tipuri de calcul de referinta asociate cu o interfata: referintele publice si referintele private. Singura diferenta dintre acestea este aceea ca referintele private pot fi lansate numai de catre clientul ce le-a cerut.

Pentru a asigura recuperarea resurselor obiectului in cazul avariilor, protocolul DCOM contine un mecanism de colectare a ‘gunoiului’ (garbage collection). Acest mecanism este bazat pe ping-ul pentru mentinerea in viata, ce permite unui client sa mentina timpul de viata al referintelor obiectului. Daca un server obiect nu reuseste sa primeasca ping-uri pentru un obiect, atunci, serverul obiect recupereaza obiectul.

- **Serviciul de rezolvare al obiectelor**

Acest serviciu face parte din protocolul DCOM ce face activarea, rezolutia OXID, colectarea gunoiului si testarea vietii serverului. Serviciul de rezolvare al obiectelor implementeaza urmatoarele interfete RPC:

- Metodele IObjectExporter
- IActivation: contine o metoda utilizata pentru a creea obiecte si producatori de clasa.
- IRemoteSCMActivator: contine mai multe metode utilizate pentru a crea obiecte si producatori de clasa.

4. Concluzii

DCOM sau CORBA?

Dacă sistemul funcționează în totalitate pe mașini Windows, atunci DCOM poate fi o soluție bună. Totuși, orice sistem heterogen se va descurca mai bine cu CORBA.

În timp ce DCOM are bune rădăcini în COM, integrarea în rețea este încă imatură. În plus, serverele Windows NT cu DCOM pornit sunt vulnerabile la respingerea de atacuri de servicii din cauza unui bug. Astfel, orice infrastructură DCOM trebuie să stea în spatele unui firewall. Presupunând că în viitor acest bug nu va mai exista, atunci DCOM ca și infrastructură e mult mai fezabil având în vedere că beneficiază de criptare la nivelul pachetelor.

Uneltele de dezvoltare pentru DCOM sunt cele mai bune dintre toate aceste tehnologii și probabil vor rămâne pentru lung timp. Aceasta din cauza angajamentului făcut de Microsoft către DCOM, la fel ca și distribuirea mare pe piață a compilatoarelor și a sistemelor de operare desktop. Totuși, suportul DCOM în afară Windows-ului este mic și nedezvoltat.

Corba este cea mai matură tehnologie, având avantajul că poate lucra cu aplicații moștenite, și rețele heterogene. O problemă sunt uneltele de dezvoltare ale CORBA care nu sunt foarte dezvoltate. Anumite ORB-uri nu împachetează uneltele care se integrează cu compilatoarele existențe. În momentul de față, în orice caz, puțini furnizori au implementat serviciul de securitate, așa că implementarea masivă prin internet poate să nu fie foarte sigură, în special dacă proiectul folosește protocolul public IIOP.

CORBA în acest moment pare a fi cea mai bună soluție a tehnologiilor obiect distribuite, celelalte tehnologii nefiind un rival destul de bun pentru CORBA.

5. Bibliografie

1. http://www.corba.org/

2. F. Chang, B, Annal, F. Grunta - A Large Scale Distributed Object Architecture CORBA & COM for Real Time Systems

3. Don Box - Essential COM

4. http://www.opengroup.org/comsource/

5. http://www.ifindkarma.com/attic/jedi/paper/ - [Figura 1, 2]

6. http://www.oser.org/~hp/ds/node38.html - [Figura 3]

7. http://www.ibm.com/software/network/dce/library/publications/appdev/html/APPDEV20.HTM - [Figura 4]

8. https://msdn.microsoft.com/en-us/library/cc226807.aspx [Figura 5, Tabelul 2,3,4]