Proiectarea Sistemelor Software Complexe

Curs 4 – Arhitecturi de Sistem Software Bazate pe Tehnologii Middleware. Obiecte Distribuite.

Rolul unui arhitect software este foarte asemănător cu cel al unui arhitect de clădiri. Atunci când un arhitect proiectează o clădire, el realizează o serie de desene care arată clădirea din diferite unghiuri. La fel ca și în cazul arhitecturii unui sistem software, desenele realizate pentru o clădire de către un arhitect au la bază o serie de cerințe, cum ar fi: spațiul disponibil, funcționalitatea clădirii (clădire de birouri, biserică, scoală, mall, etc.), estetica, bugetul, etc. Aceste desene nu reprezintă altceva decât o reprezentare abstractă a clădirii. Pentru a putea însă începe construcția clădirii mai trebuie realizate o serie de alte proiectări de detaliu, cum ari fi: proiectarea pereților astfel încât să asigure o anumită rezistență, proiectarea rețelei electrice, a rețelei de țevi care asigură alimentarea cu apă, etc. Pe măsură ce aceste proiectări de detaliu sunt realizate se aleg materialele potrivite pentru a construi clădirea.

Tehnologiile middleware reprezintă pentru un sistem software ceea ce reprezintă pentru o clădire sistemul de țevi de apă sau cel care asigură alimentarea cu energie electrică. Astfel:

- Tehnologiile middleware reprezintă modalități deja testate de conectare a diferitelor componente software dintr-o aplicație. Cu alte cuvinte tehnologiile middleware reprezintă țevile prin care se transmit date între componentele unui sistem software și care pot fi folosite pentru o gamă foarte largă de aplicații.
- Tehnologiile middleware pot fi folosite pentru a lega diverse componente în topologii utile binecunoscute. Conexiunile realizate prin intermediul tehnologiilor middleware pot fi de tipul unu-la-unu, unu-la-mail-multi și mai-mulți-la-mai-mulți.
- Din punctul de vedere al utilizatorului unei aplicații tehnologiile middleware folosite de o anumită aplicație sunt ascunse. Utilizatorul interacționează cu aplicația și nu îl interesează modul în care se realizează schimbul de informație. Cât timp tehnologiile middleware folosite de o aplicație funcționează corect, ele reprezintă o infrastructură invizibilă.
- Utilizatorul unui sistem software devine conștient de existența tehnologiei middleware doar atunci când acesta nu funcționează corespunzător (asemănător cu sistemul de țevi dintr-o clădire).

Așadar tehnologiile middleware reprezintă o infrastructură gata pentru a fi folosită, care permite conectarea componentelor software. Poate fi folosită într-o varietate foarte largă de aplicații, întru-cât a fost proiectată să fie generică și configurabilă astfel încât să poată fi folosită într-o gamă largă de sisteme software.

1.1 Clasificarea Tehnologiilor Middleware

În Fig. 4.1 este prezentă clasificarea tehnologiilor middleware:

- Nivelul transport reprezintă canalele de comunicare de bază prin intermediul cărora se transferă date între componentele software. Aceste canale oferă mecanisme simple care fac ca schimbul de date în cazul sistemelor software distribuite să fie foarte ușor de implementat.

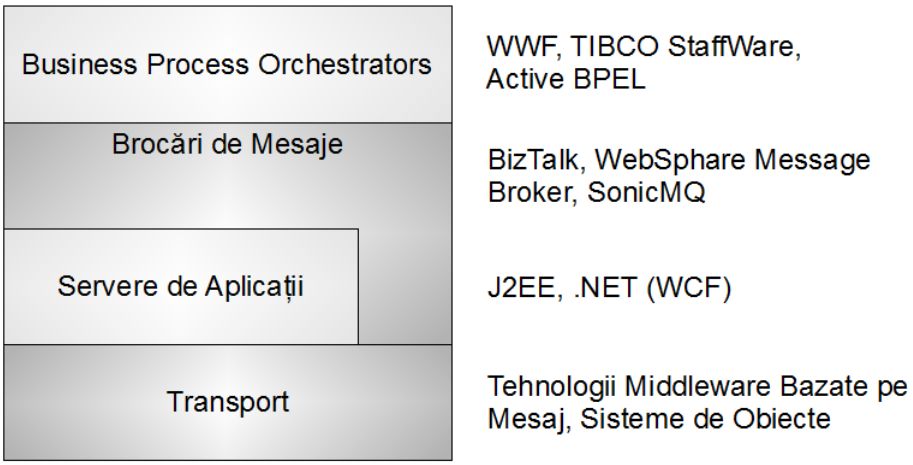

Fig. 4.1. Clasificarea tehnologiilor middleware.

- Serverele de aplicații sunt construite peste nivelul transport. Acestea oferă funcționalități suplimentare precum: suport pentru tranzacții, securitate etc. De asemenea oferă suport pentru dezvoltarea sistemelor software multi-fir bazate pe conceptul de server.
- Brocării de mesaje sunt construiți fie peste nivelul transport fie peste cel al serverelor de aplicații. Ei reprezintă module specializate în procesarea mesajelor. Oferă suport pentru procesarea rapidă a mesajelor, transformarea acestora, dispun de instrumente de programare de nivel înalt pentru a permite specificarea modului în care vor fi interschimbate, direcționate și manipulate mesajele între diferitele componente ale unui sistem software.
- Business process orchestrators (BPOs) sunt tehnologii middleware dezvoltate peste nivelul brocărilor de mesaje. Aceste tehnologii middleware oferă suport pentru dezvoltarea aplicațiilor de tip workflow. În astfel de aplicații un proces poate să dureze ore chiar zile datorită faptului că este nevoie ca anumite persoane să finalizeze anumite taskuri. BPOs oferă suport pentru descrierea workflow-ului unei astfel de aplicații, pentru execuție și pentru gestionarea stărilor intermediare până când procesul se finalizează.

1.2 Obiecte Distribuite

Tehnologiile middleware bazate pe obiecte distribuite au fost folosite încă de la începutul anilor ’90. Cea mai reprezentativă tehnologie pentru middleware-uri distribuite este reprezentată de CORBA (Common Object Request Broker Architecture).

În Fig. 4.2 este prezentat un exemplu de client care trimite o cerere și primește un răspuns prin object request broker (ORB). În CORBA interfața unui obiect servitor este exprimată cu ajutorul limbajului IDL (interface description language). Interfețele IDL definesc metodele pe care un obiect server le suporta; sunt specificați atât parametrii cât și tipul datelor returnate. Un exemplu simplu de interfață IDL este:

```
module ServerExample{
    interface MyObject
   {
       string isAlive();
   };
}
```

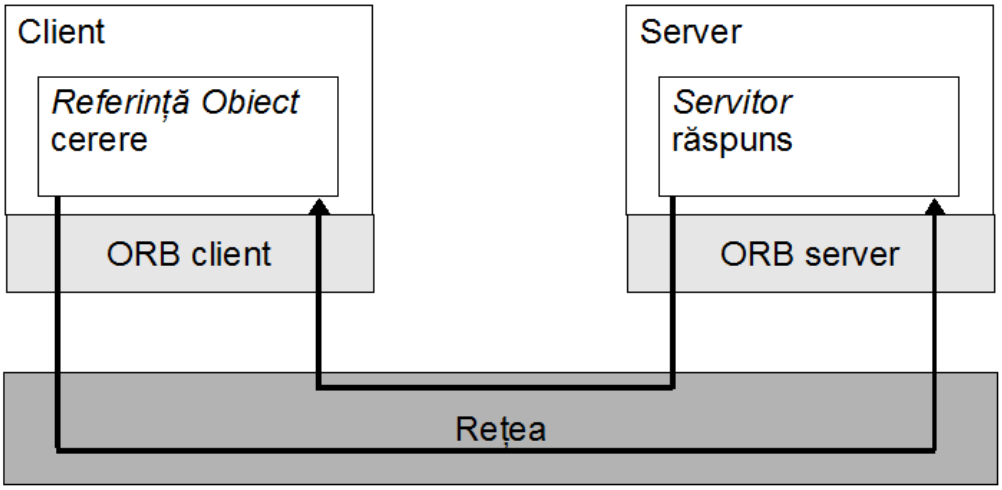

Fig. 4.2. Obiecte distribuite utilizând CORBA.

Interfața IDL prezentată mai sus definește un obiect CORBA care conține o singură metodă, *isAlive*, care returnează un obiect de tipul string și nu are nici un argument. Un compilator este folosit pentru a procesa interfața. Compilatorul generează un obiect *skeleton* într-un anumit limbaj țintă (ex.: C++, Java). Obiectul *skeleton* implementează mecanismul prin care metodele implementate pe server pot fi invocate. Programatorul trebuie apoi să scrie codul pentru a implementa fiecare metodă de pe server într-un anumit limbaj de programare:

```
class MyServant extends _MyObjectImplBase{
    public String isAlive(){
        return "It is alive...";
    }
}
```

Procesul server trebuie să creeze o instanță a clasei servitor și să ofere acces la metodele implementate de respectiva clasă prin ORB:

```
ORB orb = ORB.init(args, null);
MyServant objRef = new MyServant();
Orb.connect(objRef);
```

Procesul client trebuie să inițializeze un client ORB și să obțină o referință la un obiect servitor care este găzduit de un proces server. De obicei referințe la obiectele distribuite sunt ținute în directoare. Clienții pot astfel să localizeze obiectele distribuite printr-un nume logic.

```
ORB orb = ORB.init(args, null);
MyServant servantRef = lookup("Myservant");
String reply = servantRef.isAlive();
```

Invocarea unei metode aflate la distanță este asemănătoare ca și sintaxă cu cea a unei metode sincrone a unui obiect local. În spate însă mecanismul ORB trimite cererea împreună cu parametrii obiectului aflat pe server. Metoda se execută pe server iar rezultatul este trimis înapoi către client.

Din punctul de vedere al arhitecturii următoarele sunt câteva aspecte care trebuie considerate în faza de proiectare:

- invocarea metodelor aflate la distanță sunt relativ costisitoare (lente), cererile traversează atât nivelul ORB cât și cel rețea, ceea ce va avea un impact asupra performanței; trebuie proiectate interfețe astfel încât să se minimizeze numărul de apeluri la distanță;
- ca orice altă aplicație distribuită, serverul poate să fie indisponibil pentru o anumită perioadă de timp datorită unor defecțiuni apărute la rețea sau la mașina care găzduiește obiectul server; trebuie gândite strategii pentru a trata astfel de situații și pentru a reporni serverul;
- dacă obiectul server menține o sesiune pentru a stoca date pe durata comunicării cu obiectul client, trebuie gândite strategii de recuperare a sesiunii în cazul în care obiectul server eșuează și trebuie repornit.

1.2.1 Java Remote Method Invocation

În mod normal un sistem software bazat pe RMI constă din două programe: un program server și un program client. Programul server creează așa numitele obiecte *aflate la distanță*, asigură faptul că referințele către aceste obiecte sunt accesibile și așteaptă clienții care invocă metode ale obiectelor *aflate la distanță*. Programul client obține o referință la unul din obiectele aflate la distanță de pe server și invocă metode ale acelui obiect.

O aplicație care folosește obiecte distribuite trebuie să realizeze următoarele acțiuni:

- Localizarea obiectului aflat la distanță – pot fi folosite diferite mecanisme pentru a se obține referințe la obiecte aflate la distanță. De exemplu: o aplicație poate înregistra obiectele aflate la distanță utilizând facilitatea simplă de localizare pusă la dispoziție de RMI, altă aplicație poate returna referințe la obiecte aflate la distanță ca și rezultat al unui alt apel la distanță;
- Comunicarea cu obiectele aflate la distanță – detaliile comunicării între obiectele aflate la distanță sunt tratate la nivelul infrastructurii RMI. Pentru programator invocarea unei metode ce aparține unui obiect aflat la distanță este similară cu invocarea unei metode Java obișnuite.
- Încărcarea claselor care sunt transmise – RMI oferă mecanisme prin care se poate transmite definiția unei clase sau datele corespunzătoare unui obiect.

În Fig. 4.3 sunt ilustrate canalele de comunicare care există în cazul unei aplicații care folosește RMI registry pentru a localiza referințe la obiecte aflate la distanță. Serverul comunică cu RMI registry pentru a asocia un nume cu un obiect aflat la distanță (găzduit de server). Clientul comunică cu RMI registry pentru a obține o referință către un obiect aflat la distanță pe baza unui nume. După ce a obținut o referință la un obiect aflat la distanță clientul poate să invoce metode ale acelui obiect. În plus în figură mai apar două servere Web, aceste servere sunt folosite pentru încărcarea claselor pentru obiectele trimise de la client la server, respectiv de la server la client.

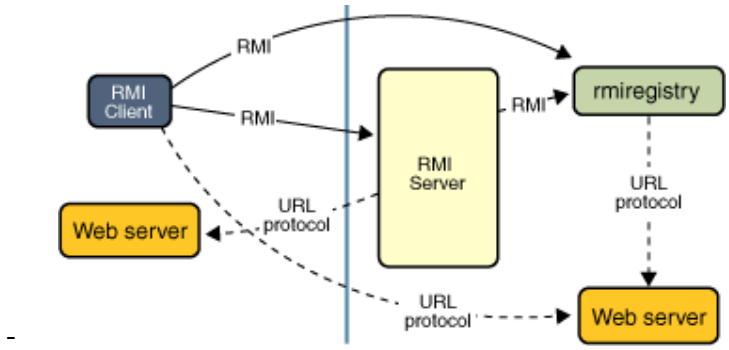

Fig. 4.3. RMI example.

Ca orice alte aplicații Java și aplicațiile RMI sunt create din clase și interfețe. Interfețele declară metode. Clasele implementează metode declarate de interfețe și eventual alte metode. Obiectele care conțin metode ce pot fi invocate din mașini virtuale Java diferite sunt denumite obiecte la distanță. Un obiect devine obiect la distanță dacă:

- implementează o interfața la distanță, o interfață care extinde interfața *Remote*;
- toate metodele declarate de interfață, declară excepția de tipul *java.rmi.RemoteException* în clauza *throws*.

RMI tratează diferit obiectele la distanță atunci când sunt transmise între mașini virtuale diferite față de obiectele Java normale. În loc de a se transmite o copie a obiectului se transmite un *stub*. Un *stub* acționează ca și un *proxy* pentru obiectul aflat la distanță. Atunci când o metodă a unui *stub* este invocată, apelul este transmis către obiectul aflat la distanță. Un stub permite invocarea doar a metodelor care au fost declarate într-o interfață derivată din interfața *Remote*.

Pentru crearea unei aplicații distribuite utilizând RMI trebuie urmăriți următorii pași:

- proiectarea și implementarea componentelor distribuite ale aplicației:
    - definirea interfețelor la distanță – aceste interfețe definesc metodele care pot fi invocate la distanță;
    - implementarea obiectelor la distanță – obiectele la distanță trebuie să implementeze una sau mai multe interfețe la distanță;
    - implementarea clienților – aplicații care invocă metode ale obiectelor la distanță;
- compilarea codului sursă – compilarea codului se realizează cu ajutorul compilatorului javac (înaintea versiunii 5.0 a platformei Java era necesar un pas suplimentar pentru invocarea utilitarului rmic, acest lucru nu mai este necesar în prezent);
- publicarea claselor pe rețea – de obicei publicarea claselor se realizează prin intermediul serverelor Web;
- pornirea aplicației – presupune pornirea utilitarului RMI remote object registry, pornirea serverului și a clientului.

Bibliografie

[1] Ian Gorton. Essential Software Architecture. Editura Springer. 2006.
[2] http://java.sun.com/docs/books/tutorial/rmi/overview.html
[3] http://www.eg.bucknell.edu/~cs379/DistributedSystems/rmi_tut.html