Using CORBA Technology on the Java Platform - JACORB

Lect.dr. Marian CRISTESCU
Universitatea „Lucian Blaga”, Sibiu

The idea behind CORBA is to model distributed resources as objects that provide a well-defined interface, and to invoke services through remote invocations (RPCs). Since the transfer syntax for sending messages to objects is strictly defined, it is possible to exchange requests and replies between processes running program written in arbitrary programming languages and hosted on arbitrary hardware and operating systems. Target addresses are represented as Interoperable Object References (IORs), which contain transport addresses as well as identifiers needed to dispatch incoming messages to implementations.
Keywords*: CORBA, JacORB, Interoperable Object References, Object Request Broker, Interface Description Language, stub, Object Management Architecture.*

Introducere

Pentru a simplifica programarea aplicaţiilor distribuite şi a permite realizarea de software bazat pe componente, s-a standardizat un model distribuit orientat pe obiecte, CORBA.
Dezvoltată ca o alternativă la modul clasic de acces la bazele de date, maparea obiect/relaţional presupune definirea unei corespondenţe între structura unei tabele şi un obiect Java. Câmpurile tabelei şi relaţiile cu alte tabele sunt mapate pe proprietăţile unui obiect. Operaţiunile simple – crearea, modificare şi ştergerea datelor presupun manipularea unor obiecte fără a mai recurge la SQL. Motorul de mapare generează automat declaraţiile SQL corespunzătoare. În cazul motorului Hibernate este oferit şi un limbaj puternic de interogare, orientat obiect şi independent de SGBD-ul utilizat.

Arhitectura generală CORBA

CORBA este un model de programare distribuita, orientat pe obiecte, propus de către OMG (Object Management Group www.omg.org).
În centrul modelului arhitectural CORBA se afla conceptul de Object Request Broker (ORB), care are un rol de "magistrala software" care intermediază interacţiunea transparenta a unor obiecte, atât local cat si la distanta. Un obiect CORBA este reprezentat printr-o interfaţă si un set de metode. O instanţă a unui obiect este identificata printr-o referinţă la obiect. Clientul unui obiect CORBA obţine referinţa obiectului si o utilizează ca un handle în realizarea apelurilor metodelor ca si cum obiectul ar fi localizat in spaţiul de adrese al clientului. ORB este responsabil de mecanismul necesar găsirii implementării obiectului, pregătirii acesteia pentru primirea unei cereri, comunicarea cererii si transmiterea rezultatului înapoi spre client. Implementarea obiectului interacţionează cu ORB prin intermediul lui Object Adapter (OA) sau prin ORB Interface.
Modelul CORBA este bazat pe comunicaţii de tip client-server. Când cere un serviciu, un client invocă o metodă implementată de un obiect aflat la distanţă, care se comportă ca un server din modelul client-server. Serviciul oferit de server este încapsulat ca un obiect şi *interfaţa obiectului* este descrisă într-un limbaj numit *Interface Description Language (IDL)*. Interfaţa definită în IDL serveşte ca un contract între un server şi clienţii săi. Clienţii interacţionează cu serverul prin invocarea metodelor descrise în IDL. Implementarea obiectului este ascunsă clienţilor. La nivelul IDL sunt prezente unele caracteristici ale programării orientate pe obiecte, cum ar fi: încapsularea, polimorfismul şi moştenirea simplă sau multiplă. De asemenea se permite specificarea excepţiilor. Un avantaj important al OMG IDL este independentă de limbaj. Deoarece OMG IDL este un limbaj declarativ, nu un limbaj de programare, el forţează definirea interfeţelor separat de implementă-

rile obiectelor. Aceasta permite ca obiectele să fie construite folosind diferite limbaje de programare şi totuşi să poată comunica între ele**.** Interfețele independente de limbaj sunt importante în cadrul sistemelor heterogene, unde nu orice limbaj de programare este suportat sau disponibil pe orice platformă. Interacțiunea între un proces client şi un obiect server este implementată în stilul apelului de procedură la distanță - RPC. Figura 1 prezintă structura tipică pentru RPC.

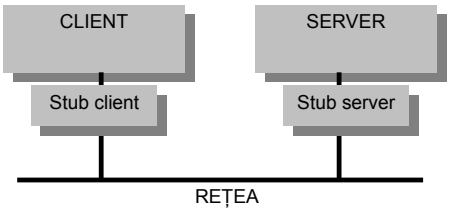

Fig. 1. Structura clasică RPC

Pentru a invoca o funcție la distanță, clientul face un apel la funcția stub-client. Stub-ul împachetează parametri de apel într-un mesaj pe care îl transmite folosind protocolul de rețea către stub-ul server. Pe partea server-ului, protocolul de rețea recepționează mesajul şi îl transmite stub-ului server, care despachetează mesajul şi apelează funcția care trebuie să efectueze cererea. În terminologie CORBA, stub-ul client poartă denumirea de stub, dar stub-ul server este denumit *skeleton*. CORBA este componenta de bază a unui model mai extins elaborat de OMG, modelul OMA (Object Management Architecture). Figura 2 prezintă componentele modelului de referință OMA.

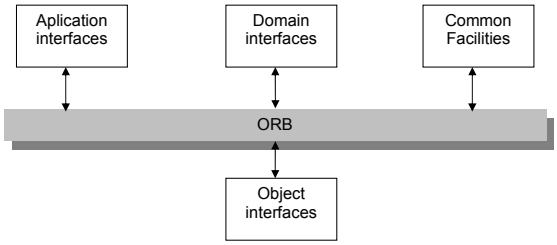

Fig. 2. Componentele modelului OMA

Principala componentă a modelului OMA este Object Request Broker (ORB). Acesta este responsabil cu facilitarea comunicării între clienți şi obiecte.

Celelalte 4 categorii de interfețe, care utilizează componenta ORB, sunt: *Object* Services, Common Facilities, Domain Interfaces şi Application Interfaces.

- Object Services: sunt interfețe independente de domeniu, folosite de majoritatea programelor; serviciile specificate de CORBA sunt: Concurrency Control Service, Event Service, Naming Service, Transaction Service, etc;
- Common Facilities: sunt utile în construcția aplicațiilor din mai multe domenii; principalele categorii de *common facilities* sunt: user interface facilities, information management facilities, system management facilities, task management facilities;
- Domain Interfaces: au un rol asemănător cu *object services* şi *common facilities*, dar sunt orientate înspre anumite domenii de aplicații (financiare, medicale, business, telecomunicații);
- Application Interfaces: sunt interfețe dezvoltate special pentru anumite aplicații; acestea nu sunt încă standardizate

O cerere de la client pentru o implementare a unui obiect se reprezentă ca în figura 3.

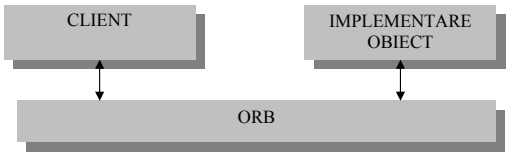

Fig. 3. Traseul unei cereri de la client la server

Clientul este o entitate care doreşte să realizeze o operație şi *Implementarea Obiectului* este codul şi datele care implementează obiectul. ORB asigură mecanismele necesare pentru găsirea implementării, pregătirea obiectului pentru primirea mesajului şi comunicarea datelor. Interfața pe care o vede clientul este complet independentă de locul unde este localizat obiectul, de limbajul de programare în care este implementat acesta sau de orice alt aspect care nu este reflectat în interfața obiectului.

ORB livrează cererile către obiecte şi returnează răspunsurile către clienți. Principalul scop al ORB este de a asigura transparența comunicării client-obiect. ORB ascunde următoarele informații despre obiectul țintă (server):

- *locația obiectului*: clientul nu cunoaşte unde se afla obiectul țintă (target object); acesta poate să fie într-un proces pe o maşină

din rețea, pe aceeaşi maşină ca şi clientul dar în alt proces sau în acelaşi proces cu acesta;

- *implementarea obiectului*: clientul nu cunoaşte cum este implementat obiectul, în ce limbaj de programare a fost scris, nici pe ce sistem de operare sau hardware funcționează;
- *starea execuției obiectului*: când face o cerere, clientul nu ştie dacă obiectul apelat este curent activat şi gata să accepte cereri; este sarcina ORB-ului să pornească serverul, dacă e necesar, înainte de a-i livra cererea;
- *mecanismul de comunicare*: clientul nu ştie ce fel de mecanism (de exemplu: TCP/IP, memorie partajată, apel local de procedură) foloseşte ORB pentru a livra cererea şi a returna răspunsul.

Aceste proprietăți ale ORB permit celor care dezvoltă aplicații să se concentreze numai asupra domeniului aplicației şi să ignore aspectele de programare distribuită de nivel scăzut.

Principale facilități componente ale CORBA sunt: ORB Core, OMG Interface Definition Language, Interface Repository, Language Mappings, Stubs and Skeletons, Dynamic Invocation and Dispatch, Object Adapters, Inter-ORB Protocols. Majoritatea dintre acestea pot fi regăsite în figura 4, care arată de asemenea şi modul în care componentele CORBA sunt legate unele de altele. Figura 4, prezintă structura unui ORB şi interfețele acestuia. Interfețele ORB-ului sunt reprezentate prin dreptunghiuri, iar săgețile indică dacă ORB-ul este apelat sau efectuează apeluri de tip „up-call”.

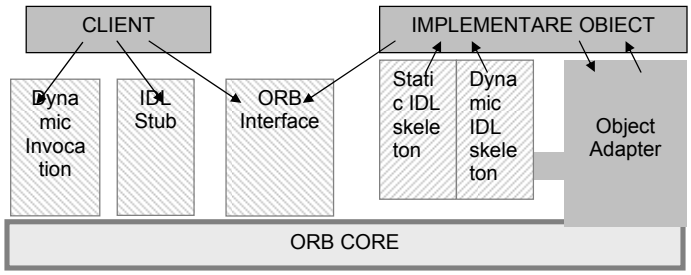

Fig. 4. Facilitățile CORBA şi modul de funcționare

Definirea interfețelor în limbajul OMG IDL

OMG IDL este un limbaj folosit la definirea interfețelor obiectelor CORBA. Obiectele CORBA sunt văzute numai în termenii interfețelor lor. IDL asigură încapsularea implementării unui obiect în spatele unei interfețe care este independentă de limbajul de implementare, de algoritmul folosit la implementare, de localizarea obiectului, de arhitectura maşinii pe care rulează, de sistemul de operare şi tehnologia rețelei. Această separare a interfeței de implementare creează premisele de a vedea CORBA ca o "magistrală software" şi constituie unul din punctele forte ale CORBA.

O interfață a unui obiect specifică operațiile şi tipurile suportate de obiect şi defineşte cererile care pot fi adresate obiectului. Interfețele sunt definite în limbajul IDL. Interfețele sunt similare claselor în C++ şi interfețelor în Java.

Un exemplu de interfață IDL este:

```
// OMG IDL in fisierul matrice.idl
interface Matrice {
   readonly attribute short height;
   readonly attribute short width;
   void set(in short x, in short y, in long value);
   long get(in short x, in short y);
};
```

Interfața `Grid` are două atribute, *height* şi *width*, care definesc dimensiunile tabelei. Deoarece au fost etichetate "read only" nu pot fi modificate direct. Există şi două opera-

ții: *set* şi *get*, care permit modificarea valorii unui anumit element şi respectiv returnează o valoare. Parametri operațiilor sunt etichetați ca parametri *in*, deci sunt parametri de intrare, transmişi de la client la server. De asemenea, mai pot exista parametri de tip *out* (de la server la client) sau *inout* (în ambele direcții).

După cum s-a menționat, OMG IDL este doar un limbaj declarativ, nu un limbaj de programare. De aceea, nu are facilități privind descrierea unor structuri de control, nici nu este folosit direct pentru implementarea unor aplicații distribuite.

Language mapping stabileşte modul cum facilitățile IDL sunt mapate pe facilitățile oferite de diferite limbaje de programare. Există reguli de mapare pentru C, C++, Smalltalk, ADA şi Java.

Mapările IDL pentru Java

În tabelul 1 sunt prezentate corespondențele tipurilor primitive definite de IDL şi cele ale limbajului Java.

Tabelul 1. Maparea tipurilor primitive Java pe tipuri IDL

Tip primitiv IDL	Tip primitiv Java
boolean	Boolean
char	Char
wchar	Char
octet	Byte
short / unsigned short	short
long / unsigned long	Int
long long / unsigned long long	Long
float	float
double	double
string, wstring	Java.lang.String
fixed	Java.math.BigDecimal

Definirea interfețelor se face cu declarația *interface*. Aceasta se mapează pe o clasă Java. Dacă se doreşte includerea claselor generate într-un pachet, se adaugă şi declarația *module*.

Exemplu:

```
module demo
{
    module hello {
        interface GoodDay {
            string hello_simple();
            wstring  hello_wide(  in
wstring msg );
        };
    };
};
```

O interfață conține atribute şi operații, analog cu proprietățile şi metodele unei clase Java.

Declararea proprietăților se face cu cuvântul cheie *attribute* urmat de tipul şi numele atributului. Dacă se adaugă cuvântul cheie *readonly* atributul nu poate fi modificat direct.

O declarație de metodă necesită specificarea tipului returnat, numele metodei şi parametrii de intrare/ieşire (în caz că există parametri). Datorită naturii distribuite a obiectelor, la declarare parametrilor se specifică şi modul de transmitere: *in*, *out*, *inout*.

Parametri „*in*” se transmit serverului dar nu sunt returnați (parametri de intrare), cei „*out*” nu sunt transmişi ci doar returnați de server (parametri de ieşire) şi cei „*inout*” sunt şi transmişi şi returnați.

Când se invocă o metodă a unui obiect, obiectul apelant aşteaptă ca metoda să se execute şi să returneze ceva. Se spune că obiectul se *blochează* aşteptând un răspuns. Dacă se doreşte evitarea situației de blocare a apelantului se poate marca metoda „*oneway*” (unidirecțională). În acesta caz apelantul nu mai aşteaptă un răspuns de la metoda apelată şi îşi continuă execuția. Consecința este că o metodă „*oneway*” nu poate returna nici o valoare. De aceea, tipul returnat trebuie declarat *void* şi toți parametri sunt „*in*”.

Concluzii

Utilizarea facilităților oferite de tehnologiile Mozilla, Hibernate şi CORBA oferă posibilități sporite de acces şi manipulare a informațiilor necesare bunei desfăşurări a actului

educaţional şi se constituie într-un instrument modern care oferă câteva facilităţi interesante ce pot uşura procesul de dezvoltare software. Utilizarea efectivă a platformei Mozilla şi a motorului ORM Hibernate permit o evaluare concretă a avantajelor şi dezavantajelor oferite.

Folosirea mapării obiect/relaţional facilitează considerabil manipulare bazelor de date ale aplicaţiilor. Operaţiuni frecvente cum sunt: inserarea, actualizarea, ştergerea înregistrărilor şi interogarea se fac într-un mod care se integrează natural în semantica limbajului Java. Limbajul propriu de interogare implementat de Hibernate se dovedeşte, în practică, mai simplu decât SQL. Singura problemă întâlnită este legată de tratarea excepţiilor Hibernate. Orice operaţie este susceptibilă de generarea unei erori, încărcându-se astfel codul sursă al aplicaţiei cu declaraţii de tipul *try {. . .} catch*. Acest lucru nu reprezintă însă un impediment major, fiind legat doar de lizibilitatea codului sursă.

Bibliografie

[AMBL04] Ambler Scott, "The Fundamentals of Mapping Objects to Relational Databases", http://www.agiledata.org/essays/mappingObjects.html, 2004

[BROS01] Brose Gerald, "Java Programming with CORBA : Advanced Techniques for Building Distributed Applications", John Wiley & Sons, 2001

[CALV00] Calvin Austin, "Advanced Programming for the Java 2 Platform", http://java.sun.com/developer/onlineTraining/Programming/JDCBook/index.html, 2000

[ROSJ97] Jeremy Rosenberger, "Teach Yourself CORBA In 14 Days", Macmillan Computer Publishing, 1997

[SIEG01] Siegel Jon, "Quick CORBA 3", Barnes & Noble, 2001

[XXXX04a] "JacORB Programming Guide", http://www.jacorb.org/releases/2.2/ProgrammingGuide.pdf.gz, 2004

[XXXX03] JavaTM 2 SDK, Standard Edition Documentation Version 1.4.2, 2003

[XXXX04b] "Gecko DOM Reference", http://www.mozilla.org/docs/dom/domref/, 2004