Aplicatii Integrate pentru Intreprinderi
Semestrul de Toamna 2013

Laborator 6

Dezvoltarea unei aplicatii distribuite folosind tehnologia CORBA

Continut

- **CORBA în contextul aplicațiilor distribuite**
- **Arhitectura CORBA**
- **Specificații IDL**
- **Etapele dezvoltării unei aplicații distribuite folosind tehnologia CORBA**
- **Studiu de Caz: Comparație între modelele de programare CORBA**

CORBA in contextul aplicatiilor distribuite

- CORBA – Common Object Request Broker Architecture
  - dezvoltat de OMG (Object Management Group)
  - arhitectura independenta de
    - platforma
    - limbajul de programare
  - destinat aplicatiilor distribuite, orientate obiect
  - permite interactiunea intre colectii de obiecte distribuite aflate pe diferite masini, vizibile intre ele, comunicand prin intermediul unei retele de calculatoare
  - aplicabilitate
    - date distribuite (securitate, separarea datelor) in BD / DD diferite
    - procesare distribuita (prelucrari paralele, servere specializate)
    - utilizatori distribuiti

Caracteristicile sistemelor distribuite

Criteriu	Sisteme Nedistribuite	Sisteme Distribuite
comunicatie	rapidă	lentă
erori	obiectele sunt distruse împreună	obiectele sunt distruse separat
acces concurent	doar prin fire de execuție	da
securitate	da	nu

Caracteristicile sistemelor distribuite (cont'd)

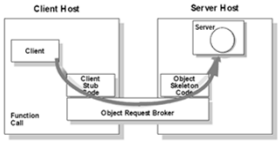
- comunicatie
  - sisteme nedistribuite: comunicatia intre obiecte se realizeaza aproape instantaneu
  - sisteme distribuite: de evitat o interactiune foarte stransa intre obiecte
- erori
  - sisteme nedistribuite: producerea de erori determina distrugerea tuturor obiectelor
  - sisteme distribuite: obiectele exista in contexte independente si trebuie distruse manual

Caracteristicile sistemelor distribuite (cont'd)

- acces concurent
  - sisteme nedistribuite
    - 1 fir de executie: acces secvential
    - mai multe fire de executie: necesitatea implementarii unor mecanisme de sincronizare pentru controlul accesului concurent
  - sisteme distribuite: implicit, resursele pot fi accesate concurent, astfel incat se impune necesitatea implementarii unor mecanisme de sincronizare
- securitate
  - sisteme nedistribuite: un singur sistem de securitate pentru toate resursele
  - sisteme distribuite: trebuie implementate mecanisme de autentificare pentru a se respecta drepturile de accesare ale resurselor

Arhitectura CORBA

- conceptul de **cerere a unui serviciu pus la dispozitie de un obiect distribuit**
  - serviciile pe care le ofera obiectul distribuit sunt specificate in interfata sa, scrisa in limbajul IDL (dezvoltat de OMG)
  - obiectele sunt identificate prin referintele la ele, avand tipurile specificate in interfata respectiva



Arhitectura CORBA (cont'd)

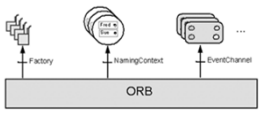
- ❶ clientul detine o referinta catre obiectul distribuit (existent pe server), definit printr-o interfata, realizand o cerere a unui serviciu
- ❷ serviciul ORB (Object Request Broker)
  - localizeaza obiectul in retea
  - transmite cererea catre acesta
  - asteapta rezultatul
  - in momentul in care rezultatul este disponibil il transmite catre aplicatia care l-a solicitat
- transparenta fata de
  - **locatia** unde se afla implementat obiectul care ofera serviciile
  - **limbajul de programare** in care se scrie cererea (diferit de cel in care este implementat serviciul obiectului)

Arhitectura CORBA (cont'd)

- **portabilitate** – definirea unui API pentru a putea rescrie cu usurinta codul pentru accesarea unui serviciu CORBA al altui producator spre a functiona si in alte context similare
  - clientii obiectului distribuit
  - implementarea obiectului CORBA
- **interoperabilitate: IIOP (Internet inter-ORB Procotol)**
  - implementat peste TCP/IP
  - permite clientilor folosind un produs CORBA realizat de un producator sa poata comunica cu obiecte din cadrul altui produs CORBA dezvoltat de alt producator
- !!! interoperabilitatea este mai importanta decat portabilitatea – folosit si in alte sisteme decat cele care implementeaza API-ul CORBA

Servicii distribuite CORBA

- COS (CORBA Services / Object Services)
  - integrarea obiectelor distribuite
  - implementate ca obiecte distribuite CORBA, definite prin interfete IDL



Servicii distribuite CORBA (cont'd)

Serviciu	Descriere
ciclu de viată al obiectului	defineşte procesele de creare, ştergere, mutare şi copiere a obiectelor CORBA
serviciu de nume	defineşte modul în care se pot asocia obiectelor CORBA nume simbolice
serviciu de evenimente	decuplează comunicarea între obiecte CORBA
serviciu de legături	stabileşte legături (cu multiplicităţi şi tipuri asociate) între obiecte CORBA
serviciu de externalizare	realizează transformarea obiectelor CORBA în / din surse externe
serviciu de tranzactii	coordonează atomicitatea accesului la obiecte CORBA
control al accesului concurent	oferă servicii de blocare a accesului la obiecte CORBA pentru asigurarea mecanismului de serializare
serviciu de proprietăti	stabileşte asocierea perechilor de tip nume-valoare cu obiecte CORBA
serviciu de identificare	găseşte obiecte CORBA pe baza proprietăţilor care descriu serviciul oferit
serviciu de interogări	permite realizarea de interogări cu obiecte CORBA

Versiuni CORBA

- exista implementari CORBA pentru mai multe limbaje de programare
- in cazul Java
  - Java ORB – distribuita impreuna cu SDK-ul de Java (unele functionalitati lipsa)
  - VisiBroker for Java (Borland)
  - OrbixWeb (Iona Technologies)
  - WebSphere (server de aplicatii – IBM)

Specificatii IDL

- IDL – Interface Definition Language
  - limbaj de definire a interfetelor dezvoltat in cadrul OMG, **independenta de limbajul de programare**
  - specificarea obiectelor distribuite prin operatiile suportate, fara a oferi detalii cu privire la modul in care sunt realizate
    - nu se specifica stari ale obiectelor / algoritmi
  - contract (garantare a functionalitatii prin parametrii de intrare/iesire) intre
    - codul care foloseste obiectul
    - codul care implementeaza obiectul
- corespondente intre IDL si C/C++, Ada, COBOL, SmallTalk, Objective C, Lisp, Java

Specificatii IDL (cont'd)

- construirea interfete modularizate pentru obiecte, specificand
  - atribute
  - metode suportate
    - valori intoarse
    - parametri
    - exceptii ce pot fi generate
- tipuri de date
  - de baza: boolean, char, octet, (unsigned) short, (unsigned) long, (unsigned) long long, float, double, string
  - construite: struct, union, enum, sequence
  - tipul any – tipul de date stabilit dinamic

Specificatii IDL (cont'd) Corespondente IDL ↔ Java

- tipare inversa (*eng.* marshaling) – transformarea unei structuri de date specifice unui limbaj de programare intr-un format CORBA IIOP

IDL	Java
boolean	boolean
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	String

IDL	Java
module	package
interface	interface
operation	method
attribute	pair of methods
exception	exception

Specificatii IDL (cont'd) Exemplu

```
module Reservation {
    struct Date {
        long day;
        long month;
        long year;
    };
    struct Moment {
        long hour;
        long minute;
    };
    struct Time {
        Date d;
        Moment m;
    };
    struct Interval {
        Time start;
        Time end;
    };
    typedef sequence<Interval> Intervals;
    exception UnspecifiedTimeTable {};
    interface ReservationService {
       Intervals getTimeTable() raises(UnspecifiedTimeTable);
       long getAvailableSeats(in Interval interval) raises(UnspecifiedTimeTable);
       boolean makeReservation(in long customerId, in Interval interval,
           in Interval numberOfSeats) raises(UnspecifiedTimeTable);
       boolean cancelReservation (in long customerId, in Interval interval)
           raises(UnspecifiedTimeTable);
    };
};
```

Specificatii IDL (cont'd)

- tipuri de parametri: in, out, inout
- exceptii
  - definite de cuvantul-cheie exception
  - asociate unei metode prin cuvantul-cheie raises
- idlj – compilator ce converteste interfata IDL in reprezentarea corespunzatoare limbajului de programare
  - idlj [flags] filename.idl
  - flag-uri: -fserver (fisierele corespunzatoare serverului); -fclient (fisierele corespunzatoare clientului) – implicit; -fall (fisierele pentru server si pentru client)
  - in directorul bin al SDK-ului de Java

Specificatii IDL (cont'd)

Fişier	Semnificatie
ReservationService.java	Interfaţa IDL reprezentată ca intefaţă Java
ReservationServiceHelper.java	Implementează operaţiile de tipare pentru interfaţă (insert, extract, read, write, narrow, unchecked_narrow)
ReservationServiceHolder.java	E folosit pentru operaţii cu parametri de tip out şi inout.
ReservationServiceOperations.java	Conţine descrierile operaţiilor implementate de interfaţă (ReservationService este derivată din ReservationServiceOperations)
_ReservationServiceStub.java	Implementează un obiect local reprezentând obiectul CORBA „la distanţă" ce transmite cererile spre obiectul distribuit

Etapele dezvoltarii unei aplicatii distribuite folosind CORBA

- ❶ definirea interfetei IDL care stabileste functionalitatea obiectului distribuit astfel incat programele server / client sa poata fi implementate in orice limbaj de programare compatibil CORBA
- ❷ compilarea interfetei care contine functionalitatea obiectului distribuit
  - versiunea interfetei corespunzatoare limbajului de programare respective
  - clasele ce contin infrastructura de conectare la ORB
    - schelet (*eng.* skeleton) – pentru server
    - ciot (*eng.* stub) – pentru client

Etapele dezvoltarii unei aplicatii distribuite folosind CORBA (cont'd)

- ❸ dezvoltarea serverului pe baza claselor schelet generate
  - mecanismul de pornire al serviciului ORB si asteptarea invocarilor de la client
  - Implementarea metodelor specificate in interfata
- ❹ dezvoltarea clientului pe baza claselor ciot generate
  - mecanismul de pornire al serviciului ORB
  - cautarea serverului folosind serviciul de nume oferit de interfata pentru a obtine o referinta catre obiectul distribuit si a-i apela metodele
- ❺ rularea: serviciului de nume, serverului, clientului

Serviciul de nume CORBA

- referintele catre obiectele distribuite sunt publicate de catre server, devenind disponibile clientilor care le pot folosi pentru a apela metodele specificate in interfata
- orbd – contine serviciile
  - de intializare (Bootstrap Service)
  - de nume tranzitoriu (Transient Name Service)
    - tnameserv se poate folosi ca alternative la orbd, fiind lansat pe portul 900 (implicit)
  - de nume permanent (Persistent Name Service)
  - proces de gestiune pentru server (Server Manager)
  - #orbd -ORBInitialPort 1100&
  - > start orbd -ORBInitialPort 1100

Sistemul de exceptii CORBA

- **exceptii sistem**
  - generate in cazul in care se produc evenimente care tin de aplicatie la modul general
    - apelul unor metode neimplementate pe server
    - problema de comunicare
    - sistemul ORB neinitializat corespunzator
  - subclasa a `RuntimeException`, nu trebuie incluse intr-un bloc `try {...} catch`
    - tratarea unor astfel de exceptii poate beneficia de avantajul obtinerii unui cod de eroare (care difera de la producator la producator)
- **exceptii utilizator**
  - generate in cazul unor erori in cadrul unor metode
  - subclasa a `Exception`, trebuie incluse obligatoriu intr-un bloc `try {...} catch`

Proiectarea serverului

- implementarea obiectului distribuit se face pe baza clasei schelet obtinuta la compilarea interfetei IDL
- mecanismul de acces la nivelul obiectului distribuit
  - despachetarea datelor primite
  - apelarea metodei ce implementeaza operatia solicitata
  - impachetarea rezultatelor transmise
- operatii
  - ❶ crearea si initializarea instantei ORB
  - ❷ obtinerea unei referinte catre obiectul radacina POA si activarea `POAManager`
  - ❸ instantierea servantului
  - ❹ obtinerea unei referinte catre contextul de nume in care se va inregistra obiectul distribuit
  - ❺ inregistrarea obiectului distribuit in contextul de nume
  - ❻ asteptarea invocarilor obiectului distribuit de catre client

Proiectarea serverului (cont'd)

- 2 mecanisme de dezvoltare a serverului pentru implementarea interfetei IDL
  - **modelul mostenirii** – implementarea unei clase care extinde clasa schelet generata de compilator
    - standardul POA – compilatorul genereaza o clasa `*POA.java` care extinde `org.omg.PortableServer.Servant` folosita drept clasa de baza pentru toate implementarile servantului (clasa care implementeaza metodele specificate in interfata IDL pe server);
    - ImplBase – compatibilitatea cu servere scrise in versiuni Java mai vechi (< 1.4)
  - **modelul delegarii**
    - o clasa generata de compilator care mosteneste clasa schelet dar deleaga toate apelurile unei clase care implementeaza metodele;
    - o clasa care implementeaza operatiile generate din interfata IDL (`*Operations.java`).

Proiectarea serverului (cont'd)

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;
import Reservation.*;

public class Server {

    public static void main (String[] args) {

        try {

            ORB orb = ORB.init(args,null);

            POA POARoot =
            POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            POARoot.the_POAManager().activate();

            ReservationServiceImplementation ReservationServiceImpl =
            new ReservationServiceImplementation();
            org.omg.CORBA.Object
            ReservationServiceImplRef =
            POARoot.servant_to_reference(ReservationServiceImpl);
            ReservationService ReservationServiceRef =
            ReservationServiceHelper.narrow(ReservationServiceImplRef);
```

Proiectarea serverului (cont'd)

```
            org.omg.CORBA.Object nameServiceRef =
            orb.resolve_initial_references("NameService");
            NamingContextExt nameContextRef =
            NamingContextExtHelper.narrow(nameServiceRef);

            String serviceName = "ReservationService";
            NameComponent nameComponent =
            new NameComponent(serviceName,"");
            NameComponent path[] = { nameComponent };

            nameContextRef.rebind(path,ReservationServiceRef);

            orb.run();

        } catch (Exception exception) {
            System.out.println ("exception: "+exception.getMessage());
        }
    }
}
```

Proiectarea serverului (cont'd)

- `ReservationServiceImplementation` – servantul care extinde clasa ReservationServicePOA, mostenind functionalitatea CORBA generata de compilatorul IDL
  - contine toate implementarile descrise prin interfata
- obiectul ORB local
  - necesar atat pe server cat si pe client
  - infrastructura de comunicatie este realizata prin intermediul sau
  - metoda `init()` primeste parametrii din linia de comanda prin care se specifica anumite proprietati ale serverului (adresa IP, port)
    - # java Server -ORBInitialPort 1100 -ORBInitialHost localhost&
    - > start java Server -ORBInitialPort 1100 -ORBInitialHost localhost

Proiectarea serverului (cont'd)

- se obtine referinta catre obiectul POA radacina prin `resolve_initial_references` aplicata obiectului ORB local si se activeaza serviciul POAManager
  - metoda `activate()` – schimba starea serviciului in activ astfel incat obiectele asociate vor prelucra cererile provenite de la client
  - POAManager – incapsuleaza starea in care se gaseste prelucrarea fiecarui obiect POA asociat
- se creeaza obiectul servant `ReservationServiceImplemenation` si se obtine referinta catre obiectul distribuit
  - ca obiect CORBA (`org.omg.CORBA.Object`) prin metoda `servant_to_reference()`
  - ca tip al interfetei, prin metoda narrow a clasei `ReservationServiceHelper` (generata de compilatorul IDL)

Proiectarea serverului (cont'd)

- serviciul de nume – face vizibile referintele catre instantele servantului
  - referinta se obtine prin metoda `resolve_initial_references` ce primeste ca parametru sirul `"NameService"` care este definit pentru toate serviciile ORB, indicand faptul ca serviciul de nume e persistent (orbd) /tranzitoriu (tnameserv)
  - transformarea catre tipul corespunzator contextului de nume se realizeaza prin metoda narrow
- se realizeaza o componenta de nume indicandu-se denumirea prin care serviciile puse la dispozitie de obiect pot fi accesate, publicarea propriu-zisa realizandu-se prin metoda rebind
- `orb.run()` – blocheaza serverul in asteptarea invocarilor din contextul clientului
  - trebuie specificata si o conditie de terminare implicand apelul metodei `orb.shutdown()`

Proiectarea clientului

- crearea si initializarea instantei ORB
  - se foloseste metoda init() care primeste parametrii din linia de comanda
    - # java Client -ORBInitialPort 1100 -ORBInitialHost localhost
    - > java Client -ORBInitialPort 1100 -ORBInitialHost localhost
  - necesar pentru impachetarea datelor transmise si prelucrarile IIOP
- obtinerea unei referinte catre serviciul de nume pentru identificarea unei referinte catre obiectul distribuit
  - `orb.resolve_initial_references("NameService")` intoarce referinta ca `org.omg.CORBA.Object`
  - transformarea la tipul corespunzator de date se face prin metoda `narrow`

Proiectarea clientului (cont'd)

- utilizarea serviciului de nume pentru cautarea obiectului distribuit pentru specificarea unei denumiri sub care acesta a fost inregistrat
  - metoda `resolve_str` intoarce o referinta catre obiectul distribuit de tipul `org.omg.CORBA.Object`
  - transformarea la tipul corespunzator se face prin metoda `narrow`
- apelurile metodelor puse la dispozitie de obiectul distribuit – specificate in interfata IDL – arata similar cu apelul unei metode realizata asupra unui obiect local
  - detaliile cu privire la impachetare / transmitere sunt transparente pentru programator

Proiectarea clientului (cont'd)

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import Reservation.*;

public class Client {

    public static void main (String[] args) {
        try {
            ORB orb = ORB.init(args,null);

            org.omg.CORBA.Object nameServiceRef =
            orb.resolve_initial_references("NameService");
            NamingContextExt nameContextRef =
            NamingContextExtHelper.narrow(nameServiceRef);

            String serviceName = "ReservationService";
            ReservationService ReservationServiceRef =
            ReservationServiceHelper.
            narrow(nameContextRef.resolve_str(serviceName));

            // TO DO: apeleaza metode obiect rezervare
        } catch (Exception exception) {
            System.out.println ("exception: "+exception.getMessage());
        }
    }
}
```

Studiu de Caz: Comparatie intre modelele de programare CORBA implementate in Java

- IIOP – protocol de transport ce asigura interoperabilitatea intre limbaje de programare si aplicatii comercializate de diversi producatori
  - IDL
  - Java RMI
- **modelul de programare IDL**
  - sistem independent de limbajul de programare unde parametrii si valorile intoarse sunt limitate la ceea ce poate fi reprezentat in limbajele de programare in care este realizata implementarea
  - orientarea pe obiecte este limitata la obiecte transmise prin referinta

Studiu de Caz: Comparatie intre modelele de programare CORBA implementate in Java (2)

- **modelul de programare IDL (cont'd)**
  - Java IDL = Java CORBA ORB + idlj (mapare IDL ↔ Java) – componenta a distributiei standard Java
  - interoperabilitate + conectivitate standardizata
  - un obiect ORB este folosit pentru prelucrari distribuite folosind comunicatie bazata pe IIOP
  - definirea interfetelor la distanta folosind IDL, obtinerea implementarilor Java a interfetei, claselor schelet / ciot care asigura comunicatia prin intermediul ORB

Studiu de Caz: Comparatie intre modelele de programare CORBA implementate in Java (3)

- **modelul de programare RMI**
  - interfata si implementarea ei sunt dezvoltate folosind acelasi limbaj de programare
  - obiecte la nivel de limbaj de programare (codul insusi) pot fi transmise de la un process la altul
  - variante
    - Java pur: Java Remote Method Protocol (JRMP)
    - limbaje de programare compatibile CORBA – folosind IIOP
  - face parte din distributia standard Java
    - compilatorul rmic – clase schelet si ciot si a legaturilor la distanta
    - serviciul ORB
  - dezvoltare rapida, flexibilitate prin transmiterea de obiecte serializabile

Studiu de Caz: Comparatie intre modelele de programare CORBA implementate in Java (4)

- IDL / RMI over IIOP – folosite numai pentru interfatarea cu sisteme mostenite (*eng.* legacy systems)
- modelul de programare RMI este preferat
  - portabilitate
  - securitate
  - colectarea memoriei disponibile