Laboratorul 05

Dezvoltarea de aplicații distribuite folosind tehnologia CORBA

Aplicaţii Integrate pentru Întreprinderi (AIPI)
Semestrul de Toamnă 2014

Departamentul de Calculatoare

Conținut

- **CORBA – aspecte generale**
- **Arhitectura tehnologiei CORBA**
- **Specificarea funcționalității unui obiect distribuit în IDL**
- **Etapele dezvoltării unei aplicaţii distribuite folosind tehnologia CORBA**
  - **Proiectarea serverului**
  - **Proiectarea clientului**
- **Comparaţie între modelele de programare CORBA implementate în Java**

CORBA – aspecte generale

- CORBA – Common Object Request Broker Architecture
  - dezvoltat de OMG (Object Management Group)
  - arhitectură independentă de
    - platformă
    - limbajul de programare
  - destinat aplicațiilor distribuite, orientate obiect
  - permite interacțiunea între colecții de obiecte distribuite aflate pe diferite mașini, vizibile între ele, comunicând prin intermediul unei rețele de calculatoare
  - aplicabilitate
    - date distribuite (securitate, separarea datelor) în BD / DD diferite
    - procesare distribuită (prelucrări paralele, servere specializate)
    - utilizatori distribuiți

Caracteristicile sistemelor distribuite

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

Caracteristicile sistemelor distribuite (cont'd)

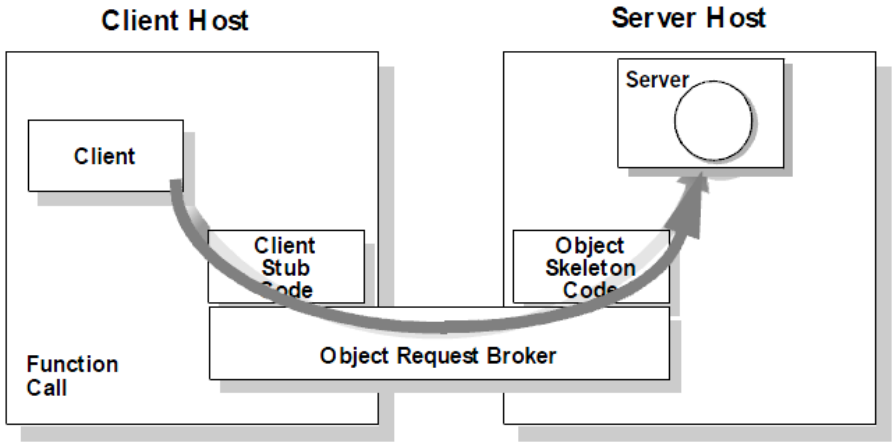
- comunicație
  - sisteme nedistribuite: comunicația între obiecte se realizează aproape instantaneu
  - sisteme distribuite: de evitat o interacțiune foarte strânsă între obiecte
- erori
  - sisteme nedistribuite: producerea de erori determină distrugerea tuturor obiectelor
  - sisteme distribuite: obiectele există în contexte independente și trebuie distruse manual

Caracteristicile sistemelor distribuite (cont'd)

- acces concurent
  - sisteme nedistribuite
    - 1 fir de execuție: acces secvențial
    - mai multe fire de execuție: necesitatea implementării unor mecanisme de sincronizare pentru controlul accesului concurent
  - sisteme distribuite: implicit, resursele pot fi accesate concurent, astfel încât 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 dispoziție de un obiect distribuit**
  - serviciile pe care le oferă obiectul distribuit sunt specificate în interfața sa, scrisă în limbajul IDL (dezvoltat de OMG)
  - obiectele sunt identificate prin referințele la ele, având tipurile specificate în interfața respectivă



Arhitectura CORBA (cont'd)

- ❶ clientul deține o referință către obiectul distribuit (existent pe server), definit printr-o interfață, realizând o cerere a unui serviciu
- ❷ serviciul ORB (Object Request Broker)
  - localizează obiectul în rețea
  - transmite cererea către acesta
  - așteaptă rezultatul
  - în momentul în care rezultatul este disponibil îl transmite către aplicația care l-a solicitat
- transparență față de
  - **locația** unde se află implementat obiectul care oferă serviciile
  - **limbajul de programare** în care se scrie cererea (diferit de cel în care este implementat serviciul obiectului)

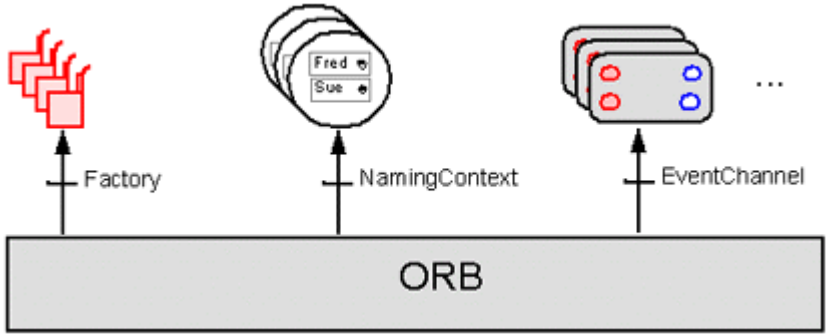
Arhitectura CORBA (cont'd)

- **portabilitate** – definirea unui API pentru a putea rescrie cu ușurință codul pentru accesarea unui serviciu CORBA al altui producător spre a funcționa și în alte contexte similare
  - clienții obiectului distribuit
  - implementarea obiectului CORBA
- **interoperabilitate: IIOP (Internet inter-ORB Procotol)**
  - implementat peste TCP/IP
  - permite clienților folosind un produs CORBA realizat de un producător să poată comunica cu obiecte din cadrul altui produs CORBA dezvoltat de alt producător

!!! interoperabilitatea este mai importantă decât portabilitatea – IIOP folosit și în alte sisteme decât cele care implementează API-ul CORBA

Servicii distribuite CORBA

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



Servicii distribuite CORBA (cont'd)

SERVICIU	DESCRIERE
ciclu de viaţă 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 tranzacţii	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 proprietăţi	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

- există implementări CORBA pentru mai multe limbaje de programare
- limbajul de programare Java
  - Java ORB – distribuită împreuna cu SDK-ul de Java (unele functionalități lipsesc)
  - VisiBroker for Java (Borland)
  - OrbixWeb (Iona Technologies)
  - WebSphere (server de aplicații – IBM)

Specificarea funcționalității unui obiect distribuit în IDL

- IDL – Interface Definition Language
  - limbaj de definire a interfețelor dezvoltat în cadrul OMG, **independent de limbajul de programare**
  - specificarea obiectelor distribuite prin operațiile suportate, fără a oferi detalii cu privire la modul în care sunt realizate
    - nu se specifică stări ale obiectelor / algoritmi
  - contract (garantare a funcționalității prin parametri de intrare/ieșire) între
    - codul care folosește obiectul
    - codul care implementează obiectul
- corespondențe cu C/C++, Ada, COBOL, SmallTalk, Objective C, Lisp, Java

Specificarea funcționalității unui obiect distribuit în IDL (cont'd)

- construirea de interfețe modularizate pentru obiecte, specificând
  - atribute
  - metode suportate
    - valori întoarse
    - parametri
    - excepții ce pot fi generate
- tipuri de date
  - de bază: `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

Specificarea funcționalității unui obiect distribuit în IDL Corespondențe IDL ↔ Java

- tipare inversă (*eng.* marshalling) – transformarea unei structuri de date specifice unui limbaj de programare într-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

Specificarea funcționalității unui obiect distribuit în IDL Exemplu


```
module reservation {
    struct Date {
        long day;
        long month;
        long year;
    };
    struct Moment {
        long hour;
        long minute;
    };
    struct Time {
        Date date;
        Moment moment;
    };
    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 long numberOfSeats)
           raises(UnspecifiedTimeTable);
       boolean cancelReservation (in long customerId, in Interval interval) raises(UnspecifiedTimeTable);
    };
};
```

Specificarea funcționalității unui obiect distribuit în IDL (cont'd)

- tipuri de parametri: `in`, `out`, `inout`
- excepții
  - definite prin cuvântul-cheie `exception`
  - asociate unei metode prin cuvântul-cheie `raises`
- `idlj` – compilator ce convertește interfața IDL în reprezentarea corespunzătoare limbajului de programare
  - `idlj [flags] filename.idl`
  - flag-uri:
    - `-fserver` (fișierele corespunzătoare serverului);
    - `-fclient` (fișierele corespunzătoare clientului) – implicit;
    - `-fall` (fișierele pentru server și pentru client)
  - în directorul `bin` al SDK-ului de Java

Specificarea funcționalității unui obiect distribuit în IDL (cont'd)

FIȘIER	SEMNIFICAȚIE
`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 dezvoltării unei aplicații distribuite folosind CORBA

- ❶ definirea interfeței IDL ce stabilește funcționalitatea obiectului distribuit astfel încât programele server / client să poată fi implementate în orice limbaj de programare compatibil CORBA
- ❷ compilarea interfeței care conține funcționalitatea obiectului distribuit
  - versiunea interfeței corespunzatoare limbajului de programare respective
  - clasele ce contin infrastructura de conectare la ORB
    - schelet (*eng.* skeleton) – pentru server
    - ciot (*eng.* stub) – pentru client

Etapele dezvoltării unei aplicații distribuite folosind CORBA (cont'd)

- ❸ dezvoltarea serverului pe baza claselor schelet generate
  - mecanismul de pornire al serviciului ORB și așteptarea invocărilor de la client
  - implementarea metodelor specificate în interfață
- ❹ dezvoltarea clientului pe baza claselor ciot generate
  - mecanismul de pornire al serviciului ORB
  - căutarea serverului folosind serviciul de nume oferit de interfață pentru a obține o referință către obiectul distribuit și pentru a-i apela metodele
- ❺ rularea: serviciului de nume, serverului, clientului

Serviciul de nume CORBA

- referințele către obiectele distribuite sunt publicate de către server, devenind disponibile clienților care le pot folosi pentru a apela metodele specificate în interfață

- `orbd` – conține serviciile
  - de inițializare (Bootstrap Service)
  - de nume tranzitoriu (Transient Name Service)
    - `tnameserv` se poate folosi ca alternativă 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 excepții CORBA

- **excepții sistem**
  - generate în cazul în care se produc evenimente care țin de aplicație la modul general
    - apelul unor metode neimplementate pe server
    - probleme de comunicare
    - sistemul ORB neinițializat corespunzător
  - subclasă a `RuntimeException`, nu trebuie incluse într-un bloc `try {...} catch`
    - tratarea unor astfel de excepții poate beneficia de avantajul obținerii unui cod de eroare (care diferă de la producător la producător)
- **excepții utilizator**
  - generate în cazul unor erori în cadrul unor metode
  - subclasa a `Exception`, trebuie incluse obligatoriu într-un bloc `try {...} catch`

Proiectarea serverului

- implementarea obiectului distribuit se face pe baza clasei schelet obținută la compilarea interfetei IDL
- mecanismul de acces la nivelul obiectului distribuit
  - despachetarea datelor primite
  - apelarea metodei ce implementează operația solicitată
  - împachetarea rezultatelor transmise
- operații
  - ❶ crearea și inițializarea instanței ORB
  - ❷ obținerea unei referințe către obiectul rădăcină POA și activarea `POAManager`
  - ❸ instanțierea servantului
  - ❹ obținerea unei referințe către contextul de nume în care se va înregistra obiectul distribuit
  - ❺ înregistrarea obiectului distribuit la contextul de nume
  - ❻ așteptarea invocărilor obiectului distribuit de către client

Proiectarea serverului (cont'd)

- 2 mecanisme de dezvoltare a serverului pentru implementarea interfeței IDL
  - **modelul moștenirii** – implementarea unei clase care extinde clasa schelet generată de compilator
    - standardul POA – compilatorul generează o clasă `*POA.java` care extinde `org.omg.PortableServer.Servant` folosita drept clasă de bază pentru toate implementările servantului (clasa care implementează metodele specificate în interfața IDL pe server);
    - ImplBase – compatibilitatea cu servere scrise in versiuni Java mai vechi (< 1.4)
  - **modelul delegării**
    - o clasă generată de compilator care moșteneste clasa schelet dar deleagă toate apelurile unei clase care implementeaza metodele;
    - o clasă care implementează operațiile generate din interfața 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`, moștenind funcționalitatea CORBA generată de compilatorul IDL
  - conține toate implementările descrise prin interfață
- obiectul ORB local
  - necesar atât pe server cât și pe client
  - infrastructura de comunicație este realizată prin intermediul său
  - metoda `init()` primește 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 obține referința către obiectul POA rădăcină
  prin `resolve_initial_references` aplicată obiectului ORB local și
  se activeaza serviciul `POAManager`
  - metoda `activate()` – schimbă starea serviciului în activ astfel încât obiectele asociate vor prelucra cererile provenite de la client
  - `POAManager` – încapsulează starea în care se gasește prelucrarea fiecărui obiect POA asociat

- se creează obiectul servant `ReservationServiceImplemenation` și
  se obține referința către obiectul distribuit
  - ca obiect CORBA (`org.omg.CORBA.Object`) prin metoda `servant_to_reference()`
  - ca tip al interfetei, prin metoda `narrow` a clasei `ReservationServiceHelper` (generată de compilatorul IDL)

Proiectarea serverului (cont'd)

- serviciul de nume – face vizibile referințele către instanțele servantului
  - referința se obține prin metoda `resolve_initial_references()` ce primeste ca parametru șirul de caractere `"NameService"` care este definit pentru toate serviciile ORB, indicând faptul că serviciul de nume e persistent (`orbd`) / tranzitoriu (`tnameserv`)
  - transformarea către tipul corespunzator contextului de nume se realizează prin metoda `narrow()`
- se realizează o componentă de nume indicându-se denumirea prin care serviciile puse la dispozitie de obiect pot fi accesate, publicarea propriu-zisă realizându-se prin metoda `rebind()`
- `orb.run()` – blochează serverul în așteptarea invocărilor din contextul clientului
  - trebuie specificată și o condiție de terminare implicând apelul metodei `orb.shutdown()`

Proiectarea clientului

- crearea și inițializarea instanței ORB
  - se foloseste metoda `init()` care primește 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
- obținerea unei referințe către serviciul de nume pentru identificarea unei referințe către obiectul distribuit
  - `orb.resolve_initial_references("NameService")` întoarce referința ca `org.omg.CORBA.Object`
  - transformarea la tipul corespunzător de date se face prin metoda `narrow`

Proiectarea clientului (cont'd)

- utilizarea serviciului de nume pentru căutarea obiectului distribuit pentru specificarea unei denumiri sub care acesta a fost înregistrat
  - metoda `resolve_str()` întoarce o referință către obiectul distribuit de tipul `org.omg.CORBA.Object`
  - transformarea la tipul corespunzător se face prin metoda `narrow`
- apelurile metodelor puse la dispoziție de obiectul distribuit – specificate în interfața IDL – arată similar cu apelul unei metode realizată asupra unui obiect local
  - detaliile cu privire la împachetare / 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());
        }
    }
}
```

Comparație între modelele de programare CORBA implementate de Java

- IIOP – protocol de transport ce asigură interoperabilitatea între limbaje de programare și aplicații comercializate de diverși producători
  - IDL
  - Java RMI
- **modelul de programare IDL**
  - sistem independent de limbajul de programare unde parametrii și valorile întoarse sunt limitate la ceea ce poate fi reprezentat în limbajele de programare în care este realizată implementarea
  - orientarea pe obiecte este limitată la obiecte transmise prin referință

Comparație între modelele de programare CORBA implementate de Java (2)

- **modelul de programare IDL (cont'd)**
  - Java IDL = Java CORBA ORB + `idlj` (mapare IDL ↔ Java) – componentă a distribuției standard Java
  - interoperabilitate + conectivitate standardizată
  - un obiect ORB este folosit pentru prelucrări distribuite folosind comunicație bazată pe IIOP
  - definirea interfețelor la distanță folosind IDL, obținerea implementărilor Java a interfeței, claselor schelet / ciot care asigură comunicația prin intermediul ORB

Comparație între modelele de programare CORBA implementate de Java (3)

- **modelul de programare RMI**
  - interfața și implementarea ei sunt dezvoltate folosind același limbaj de programare
  - obiecte la nivel de limbaj de programare (codul însuși) pot fi transmise de la un proces la altul
  - variante
    - Java pur: Java Remote Method Protocol (JRMP)
    - limbaje de programare compatibile CORBA – folosind IIOP
  - face parte din distribuția standard Java
    - compilatorul `rmic` – clase schelet și ciot și a legăturilor la distanță
    - serviciul ORB
  - dezvoltare rapidă, flexibilitate prin transmiterea de obiecte serializabile

Comparație între modelele de programare CORBA implementate de Java (4)

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