

Technologie Internetowe

Laboratorium 2: CORBA w Java

CORBA jest standardem niezależnym od używanego języka programowania. Istnieją implementacje dla większości języków programowania, także oprócz C++ możemy CORBA użyć między innymi w Java, Smalltalk, Ada, Delphi, Python, Cobol, PHP. Istotną kwestią jest fakt, że w jednej rozproszonej aplikacji wykorzystującej CORBA mogą być użyte różne języki programowania. Nie ma zatem przeszkód, aby np. serwer był napisany w C++, a klient w Java. Można również w pierwszych fazach projektu użyć do implementacji serwera języka pozwalającego na szybszą implementację funkcjonalności (np. języki skryptowe), a w drugiej fazie dopiero podmienić serwer na wydajniejszy C++ w sposób zupełnie niezauważalny dla użytkownika.

W Java implementacja CORBA znajduje się w podstawowej dystrybucji.

Wygenerowanie podstawowych plików z IDL

Ten sam plik z opisem interfejsu wykorzystuje się dla każdego z języków. W Java, aby wytworzyć wymagane klasy i interfejsy należy wykonać następujące polecenie:

```
idl -f all kalkulator.idl
```

W wyniku tego polecenia powstanie katalog *ti* (wynikający ze zdefiniowanego modułu), a w nim następujące pliki:

- **KalkulatorOperations.java** – zawiera deklarację interfejsu zdalnego
- **Kalkulator.java** – reprezentuje obiekt CORBA implementujący interfejs Kalkulator; dziedziczy po KalkulatorOperations oraz org.omg.CORBA.Object.
- **KalkulatorPOA.java** - skeleton udostępniający podstawową funkcjonalność serwera; właściwa implementacja serwera dziedziczy po tej klasie
- **KalkulatorHelper.java** – zawiera funkcje pomocnicze (np. metodę *narrow()*).
- **KalkulatorHolder.java** – służy do przekazywania obiektu Kalkulator jako parametru typu *out*.
- **_KalkulatorStub.java** – stub na którym klient wykonuje lokalne wołania; stub przekształca je do transmisji przez CORBA.

Ponadto dla wszystkich zdefiniowanych struktur, wyjątków, itp. tworzone są klasy o tej samej nazwie jak w IDL oraz **Holder* do zwracania obiektów jako parametr typu *out* i **Helper* z funkcjami pomocniczymi.

Wszystkie klasy są w pakiecie wynikającym z modułu w jakim znajdował się interfejs, wyjątek, itp. (w naszym przypadku *ti*).

Stworzenie serwera

Teraz należy stworzyć serwer dla aplikacji, czyli zaimplementować zdefiniowany interfejs oraz udostępnić obiekt dla klientów.

Implementacja parametrów typu *out*

Implementując serwer w C++ było widać, że praktycznie nie ma różnicy, czy przekazywany parametr do metody jest typu *out*, czy *in*. Wynikało to z przekazywania

parametru przez referencję. W Java nie ma tak łatwej możliwości¹, wobec czego używane są obiekty opakowujące typu **Holder*. Mają one publiczne pole *value* z wartością typu, jakiego jest parametr w IDL. Jeżeli parametr jest typu *inout* wartość *value* jest już określona i można na niej bezpośrednio wykonywać działania. Można ją podmienić (np. zwrócić referencję na inny obiekt). Trochę inaczej jest w przypadku parametru typu *out* – tutaj trzeba pierw zainicjować pole *value*, przypisując mu jakiś obiekt (chyba, że jest to typ prymitywny (np. *long*, *double*, ...)), gdyż w tym przypadku wartość pola *value* jest nieokreślona.

Implementacja interfejsu

Najlepiej zacząć od implementacji interfejsu. Zwyczajowo implementującą klasę nazywa się tak samo jak interfejs z przyrostkiem *Impl* -> *KalkulatorImpl*. Klasa implementująca musi dziedziczyć po klasie *KalkulatorPOA*.

Przykładowa implementacja klasy może wyglądać następująco:

```
public class KalkulatorImpl extends ti.KalkulatorPOA {
    public KalkulatorImpl() {
    }

    public int dodaj(int x1, int x2) {
        return x1 + x2;
    }

    public void odejmij(org.omg.CORBA.IntHolder x1, int x2) {
        x1.value -= x2;
    }

    public void podziel(double x, double y, org.omg.CORBA.DoubleHolder wynik)
        throws ti.DzieleniePrzezZero
    {
        if ( y == 0 )
            throw new ti.DzieleniePrzezZero();

        wynik.value = x / y;
    }

    public ti.Zespolona dodajZ(ti.Zespolona z1, ti.Zespolona z2) {
        ti.Zespolona wynik = new ti.Zespolona();

        wynik.rzeczywista = z1.rzeczywista + z2.rzeczywista;
        wynik.urojona = z1.urojona + z2.urojona;

        return wynik;
    }

    public void odejmijZ(ti.ZespolonaHolder z1, ti.Zespolona z2) {
        ti.Zespolona z1Value = z1.value;

        z1Value.rzeczywista -= z2.rzeczywista;
        z1Value.urojona -= z2.rzeczywista;
    }
}
```

Implementacja serwera – udostępnienie obiektu

Pozostało jeszcze utworzyć obiekt serwera i udostępnić go klientom. Można to zrobić następująco:

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.io.*;
import org.omg.CosNaming.*;
```

¹ Wszystkie obiekty są co prawda przekazywane przez referencję, ale można zmienić samej referencji, czy zawartości zmiennej użytej przy przekazywaniu wartości *long*.

```

public class KalkulatorServer {

    /** Creates a new instance of KalkulatorServer */
    public KalkulatorServer() {
    }

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

            // pobranie referencji do RootPOA (głównego "kontenera"
            // udostępnianych obiektów)
            // POA = Portable Object Adapter
            POA rootpoa = POAHelper.narrow( orb.resolve_initial_references("RootPOA") );
            rootpoa.the_POAManager().activate();

            // Stworzenie obiektu serwera
            KalkulatorImpl kalkImpl = new KalkulatorImpl();
            rootpoa.activate_object( kalkImpl );

            // Pobranie referencji do obiektu CORBA
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference( kalkImpl );
            ti.Kalkulator kalkulator = ti.KalkulatorHelper.narrow( ref );

            // zapisanie referencji do pliku
            PrintWriter out = new PrintWriter( new FileWriter( "kalkulator.ref" ) );
            out.println( orb.object_to_string( kalkulator ) );
            out.close();

            // zacznij przyjmować zapytania do kalkulatora
            System.out.println( "Start kalkulatora" );
            orb.run();
        }

        catch( Exception e ) {
            System.err.println( "ERROR: " + e );
            e.printStackTrace( System.err );
        }

        new File( "kalkulator.ref" ).delete();
        System.out.println( "Koniec kalkulatora ..." );
    }
}

```

Powyższa implementacja zakłada wymianę referencji do obiektu poprzez plik *kalkulator.ref*. Jednakże tak nie musi być – do odnalezienia referencji można użyć usługi nazw, ale to na kolejnych laboratoriach.

Implementacja klienta

Poniżej przedstawiona została przykładowa implementacja klienta. Po początkowej części, w której uzyskiwana jest referencja do obiektu, dalsza część przebiega zupełnie w sposób niezauważalny, iż obiekt znajduje się na innym komputerze (należy jedynie pamiętać o klasach Holder).

```

import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import ti.*;

public class KalkulatorClient {

    /** Creates a new instance of KalkulatorClient */
    public KalkulatorClient() {
    }

    public static void main( String args[] ) {
        try {

```

```

// Zainicjuj CORBA
ORB orb = ORB.init( args, null );

// wczytaj referencję z pliku
BufferedReader in = new BufferedReader( new FileReader( "kalkulator.ref" ) );
String ref = in.readLine();
in.close();

// Zamień referencję na obiekt kalkulator
org.omg.CORBA.Object cobj = orb.string_to_object( ref );
Kalkulator kalkulator = KalkulatorHelper.narrow( cobj );

// inicjacja zmiennych
int x1 = 10, x2 = 3;
IntHolder x1Holder = new IntHolder();
DoubleHolder iloraz = new DoubleHolder();
Zespolona z1 = new Zespolona(), z2 = new Zespolona();
ZespolonaHolder z1Holder = new ZespolonaHolder();

z1.rzeczywista = 15;
z1.urojona = -3;

z2.rzeczywista = 4;
z2.urojona = 3;

// wykonaj działania
System.out.println( x1 + " + " + x2 + " = " + kalkulator.dodaj( x1, x2 ) );

x1Holder.value = x1;
// wynik zapisany do x1Holder ; wartość x1 jest niezmiennona
kalkulator.odejmij( x1Holder, x2 );

System.out.println( x1 + " - " + x2 + " = " + x1Holder.value );
x1 = x1Holder.value;

try {
    System.out.print( x1 + " / " + x2 + " = " );
    kalkulator.podziel( x1, x2, iloraz );
    System.out.println( iloraz.value );

    System.out.print( x1 + " / " + x2 + " = " );
    kalkulator.podziel( x1, 0, iloraz );
    System.out.println( iloraz.value );
} catch ( DzieleniePrzezZero e ) {
    System.out.println( "dzielenie przez 0" );
}

System.out.println( zespolona2str( z1 ) + " + " + zespolona2str( z2 )
    + " = " + zespolona2str( kalkulator.dodajZ( z1, z2 ) ) );

System.out.print( zespolona2str( z1 ) + " - " + zespolona2str( z2 ) + " = " );
z1Holder.value = z1;
kalkulator.odejmijZ( z1Holder, z2 ); // wynik jest zapisywany do z1Holder
System.out.println( zespolona2str( z1Holder.value ) );

} catch( Exception e ) {
    System.out.println( "ERROR : " + e );
    e.printStackTrace();
}
}

private static String zespolona2str( Zespolona z ) {
    return z.rzeczywista + "+" + z.urojona + "i";
}
}

```

Kompilacja i uruchomienie

W przypadku Java kompilacja odbywa się w dokładnie taki sam sposób, jak każdy inny program Java.

javac -d classes KalkulatorClient.java KalkulatorServer.java
oraz uruchomienie

```
java -cp classes KalkulatorServer &  
java -cp classes KalkulatorClient
```

Inne uwagi

- Do rzutowania klas CORBA nie powinno się używać operatora (), lecz statycznej metody *narrow()* odpowiedniej klasy Holder
- W Java można dziedziczyć tylko po jednym przodku, natomiast klasa implementująca interfejs musi dziedziczyć po odpowiedniej klasie *POA. Zatem przy dziedziczeniu należy albo przekopiować implementację z klasy nadrzędnej albo stworzyć równoległe klasy implementujące właściwą hierarchię dziedziczenia, a w samych udostępnianych obiektach zastosować delegację. Drugi mechanizm, choć bardziej przejrzysty i tak nie zadziała w przypadku wielodziedziczenia (które jest możliwe w CORBA, a w Java tylko dla interfejsów).
- Stworzony przykład można łączyć w dowolnej konfiguracji z wcześniejszym przykładem w C++ (serwer C++, klient Java ; serwer Java , klient C++).

Dodatkowe informacje

Sporo dodatkowych źródeł informacji o CORBA oraz linków do innych stron można znaleźć pod adresem:

http://www.eti.pg.gda.pl/katedry/kask/dydaktyka/Obiektowe_systemy_rozproszone/