Pemrograman Jaringan 12

CORBA

CORBA (Common Object Request Broker Architecture)

- CORBA (www.corba.org) adalah cara lain untuk melakukan pemrograman jaringan terdistribusi dan open system, dimana obyek yang dipanggil tidak hanya berasal dari program yang dibuat dengan bahasa Java saja tetapi juga bisa dibuat dengan bahasa lain.
- Corba di buat oleh OMG (Object Management Group – www.omg.org), suatu organisasi yang mengurusi teknologi berbasis obyek. OMG berdiri tahun 1989 dan juga mengurusi tentang UML.
- Corba dikatakan merupakan standar sistem terdistribusi (*distributed sistem standard*) karena dengan menggunakan corba, sistem secara keseluruhan dapat saling terhubung dan berkomunikasi antar platform (sistem operasi dan hardware) yang berbeda.

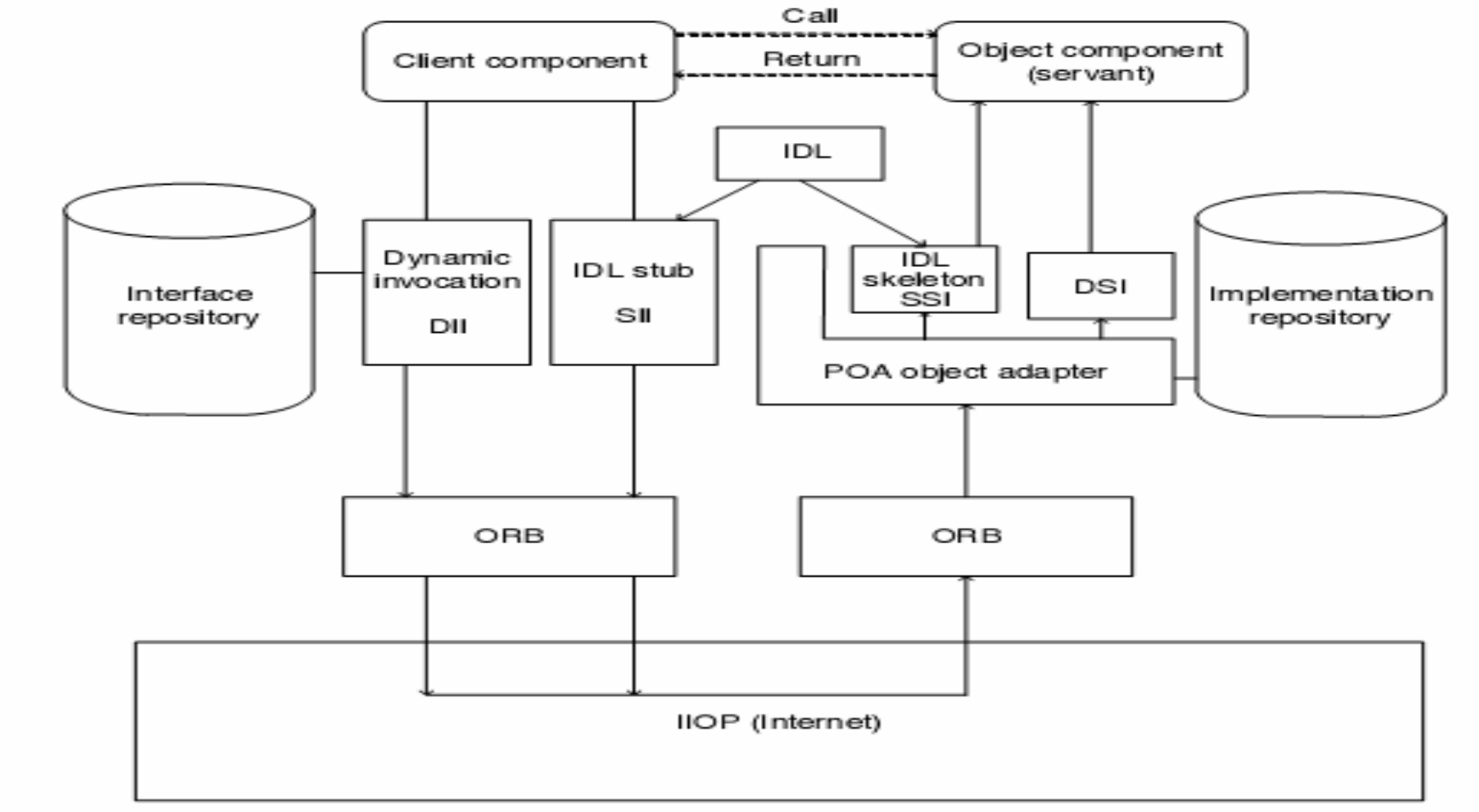
CORBA (2)

- Corba juga dikatakan sebagai *open system* karena teknologi corba merupakan suatu standar yang terbuka bagi siapa saja yang ingin menerapkannya. Dengan corba kita dapat membangun aplikasi yang dapat saling berkomunikasi walau satu sama lain menggunakan bahasa pemrograman dan platform yang berbeda.
- CORBA memiliki Interface Definition Interface yang mendukung mapping ke suatu bahasa pemrograman tertentu.
- CORBA menyediakan API untuk berkomunikasi antar obyek secara remote.
- Beberapa yang sudah menerapkan spesifikasi corba adalah:
  - Borland (VisiBroker): C++ dan Java Mapping
  - IONA (Orbix) : C++, Java, dan SmallTalk Mapping
  - Sun Microsystem (JavaIDL)

Perbandingan RMI & CORBA

Perbandingan	CORBA	RMI
Bahasa pemrograman yang didukung	Java, C++, VB, Delphi, Perl dan Phyton	Java
Service	Naming, Lifecycle, Transaction Event	Naming
Kemudahan	Mudah	Mudah
Mendukung sistem terdistribusi	Sangat baik	Baik
Standard organisasi terbuka	Ya	Tidak

CORBA Architecture



ORB

- Bertindak sebagai broker (perantara) antara client dan server yang berjalan pada tiap mesin yang berisi API untuk mencari obyek dan menerima request.
- ORB mengkomunikasikan hubungan antar obyek menggunakan sistem IIOP (Internet Inter-ORB Protocol)
- ORB tersedia untuk beberapa platform yang berbeda-beda.
- ORB mencari obyek, merequest remote method melalui interface CORBA, dan mengembalikannya ke client.
- Menangani secara menyeluruh terhadap suatu permintaan (request) dari client ke object atau sebaliknya (response) dari obyek ke client.
- ORB harus tersedia di sisi server dan client.

ORB (2)

- Pada sisi client, ORB memiliki fungsi:
  - Menghubungkan ke interface repository / IR (penyedia definisi interface).
  - Membantu client dalam menyusun suatu permintaan (invocation) ke object server secara dinamis dengan menggunakan DII (Dynamic Invocation Interface).
- Pada sisi server, ORB berfungsi:
  - Selain bertanggung jawab untuk mengirimkan response dari server ke client yang dituju, ORB juga membantu untuk memulai dan menghentikan operasi terhadap object server yang diminta.

Stub dan Skeleton

- Digunakan untuk marshalling dan Unmarshalling remote method invocation.
  - Marshalling: encoding, to pack all information about remote method invocation to be sent to the remote destination.
  - Unmarshalling: unpack and decode the message
  - Stub marshall the method request, and Skeleton unmarshall the request and forward to actual remote method.
  - Stub berkomunikasi dengan remote object.
- Ada 2 cara menghasilkan kode stub pada client dan kode skeleton pada server:
  - Static: SII (static invocation interface) dan SSI (static skeleton interface), digenerate saat kompilasi IDL.
  - Dynamic: DII (dynamic invocation interface) dan DSI (dynamic skeleton interface)

Object Adapter

- Menerima permintaan dari client.
- Berfungsi sebagai dispatcher (menentukan object servant mana yang dituju).
- Membuat suatu remote objek referensi terhadap setiap objek servant CORBA yang terdaftar padanya. Setiap obyek CORBA akan diberi nama unik, dan setiap nama menunjuk pada suatu obyek servant.
- Dapat mengaktifkan dan menonaktifkan suatu objek servant.
- Mengatur security, method invocation dari object servant
- Melakukan pemanggilan terhadap sebuah object servant, yaitu dengan cara statik, yaitu melalui Static Skeleton Interface (SSI), atau secara dinamis dengan menggunakan Dynamic Skeleton Interface (DSI).
- Nama object Adapter untuk CORBA 2.2 ke atas disebut dengan Portable *Object Adapter (POA)*, dan untuk spesifikasi CORBA 2.1 ke bawah disebut dengan *Basic Object Adapter (BOA)*.

Interface Respiratory

- Database pada sisi server yang berisi semua metadata interface IDL yang telah diregistrasikan ke server, termasuk tipe data, nama method, dan parameternya.

ORB References

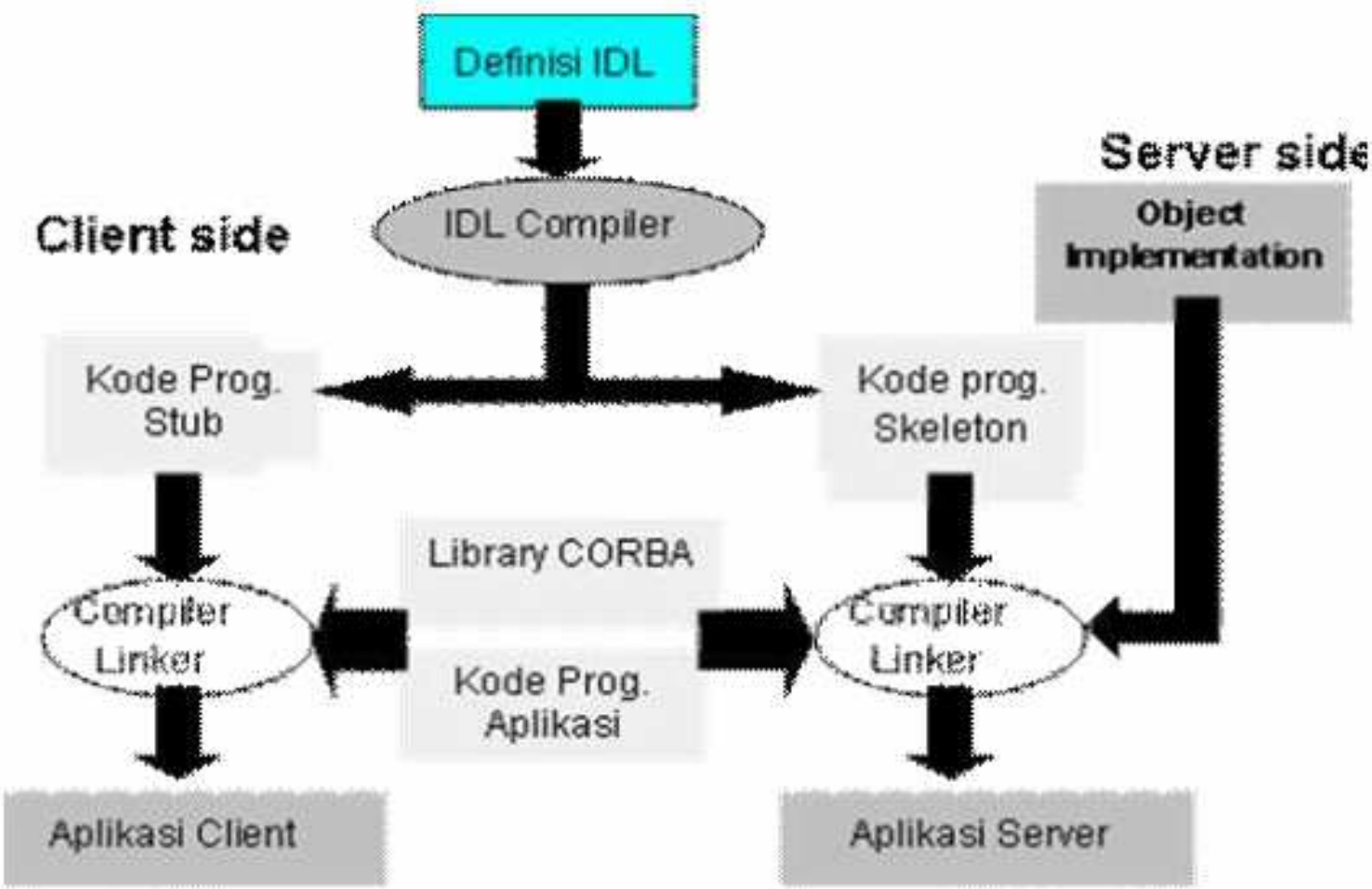
- Berada di sisi client
- Membungkus lokasi dari remote obyek yang akan diakses oleh client.
- Client harus mengambil/mencari OR untuk dapat mengakses remote object.

IDL (Interface Definition Language)

- IDL interface yang berisi kumpulan method yang akan diakses oleh client.
- Language Dependent Text File berekstensi .idl
- Contoh:

```
module MyAccount {
    interface Account{
       attribute long  accountNo;
       void deposit (in double amount);
       void withdraw(in double amount);
       double reportBalance(); }
}
```

Langkah Pengembangan CORBA



Langkah-langkah

- Langkah pertama: mendefinisikan interface yang berisi layanan-layanan yang tersedia oleh obyek server menggunakan bahasa IDL.
- Langkah kedua: mengkompilasi bahasa IDL ke bahasa Java. Kita gunakan IDL-to-Java compiler. Yaitu : idlj. Hasil dari kompilasi ini dihasilkan client stub dan skeleton server.
- Untuk menghasilkan aplikasi client, kita perlu menuliskan aplikasi client dan dikompilasi dan dilink bersama dengan library CORBA serta client stub yang dihasilkan dari IDL compiler.
- Dari sisi server : dari definisi yang sudah didefinisikan perlu dibuat sebuah kelas sebuah kelas yang merupakan implementasi dari interface tersebut. Sehingga sebuah aplikasi server dihasilkan dari kompilasi dan linking antara kode program aplikasi server, implementasi obyek dan skeleton.

IDL (Interface Definition Language)

Tipe data IDL	Tipe Data Java
Boolean	boolean
Char	char
Wchar	java.lang.String
Short	short
String	java.lang.String
Octet	byte
Short	short
unsigned sort	short
Long	int
unsigned long	int
long long	long
unsigned long long	long
float	float
double	double

Contoh IDL

- IDL **konstanta**:

  interface Konstanta{

  const long MyLong = 12345;

  const string nama = “anton”;

  const boolean ada = TRUE;

  }
- Oleh Java akan diterjemahkan menjadi:

  public interface Konstanta{
- public static final int MyLong = (int)(12345);
- public static final String nama = "anton";
- public static final boolean ada = (boolean)(true);

  }

- interface Katalog {
  struct Buku {
  string Judul;
  string Pengarang;
  long Tahun;
  };
  void SimpanBuku(in Buku bkbaru);
- };

- Hasilnya:
- public final class Buku{
- public String Judul = null;
- public String Pengarang = null;
- public int Tahun = (int) 0;
- }

- enum Status { DIPINJAM, HILANG, RUSAK, ADA };

- Hasilnya menjadi:
- private int __value;
- private static int __size = 4;
- private static Status[] __array = new Status [__size];

- public static final int _DIPINJAM = 0;
- public static final Status DIPINJAM = new Status(_DIPINJAM);
- public static final int _HILANG = 1;
- public static final Status HILANG = new Status(_HILANG);
- public static final int _RUSAK = 2;
- public static final Status RUSAK = new Status(_RUSAK);
- public static final int _ADA = 3;
- public static final Status ADA = new Status(_ADA);

Pengembangan Aplikasi CORBA

Buat definisi IDL:

```
module HelloApp {
  interface Hello {
    string sayHello();
  };
};
```

Beri nama hello.idl

Pengembangan Aplikasi CORBA

- **Kompilasi dengan perintah:**
  idlj hello.idl
- **Anda akan mendapatkan direktori HelloApp dan di dalamnya terdapat kelas-kelas:**
  - **HelloHelper.java**
    Bertanggung jawab untuk membaca dan menulis tipe data ke stream CORBA dan menterjemahkan dari tipe Any.
  - **HelloHolder.java**
    Class ini menyimpan public instance dari tipe Hello. Ketika terdapat tipe parameter out atau inout, kelas ini digunakan.
  - **Hello.java**
    Digunakan untuk deklarasi method dan ketika digunakan pada interface lain.
  - **HelloOperations.java**
    Interface ini digunakan untuk pemetaan sisi server dan dishare untuk stub dan skeleton.
  - **HelloPOA.java**
    Class yang memerankan server skeleton. Class server harus menerapkan dari kelas ini.
  - **_HelloStub.java**
    Merupakan class client stub, yang menyediakan fungsi CORBA pada sisi client.

Pengembangan Aplikasi CORBA

Buat definisi class implementasi dari antarmuka Hello, disebut juga kelas Servant yang merupakan turunan dari kelas HelloPOA yang berada di dalam package HelloApp yang terbentuk!

```
import HelloApp.*;
import org.omg.CORBA.*;
class HelloImpl extends HelloPOA {
  private ORB orb;
  public HelloImpl(ORB orb) {
    this.orb = orb;
  }
  public String sayHello() {
    return "\nHello world!!\n";
  }
}
```

Pengembangan Aplikasi CORBA

- Import semua kelas-kelas dan package yang dibutuhkan!
- Buat obyek ORB dan inisialisasi, lihat baris merah
- Buat obyek dari kelas implementasi (POA) atau object servant (BOA)
  - jika diperlukan object ORB yang sudah dibuat dapat dijadikan parameter ke object servant, dengan tujuan agar object servant dapat mengkontrol ORB yang digunakan. Lihat baris biru
- Buat referensi dari root POA dan aktifkan POA Manager, lihat warna hijau
- Dapatkan referensi obyek yang dibuat pada langkah sebelumnya dengan bantuan root POA, lihat warna orange
- Buat koneksi ke Naming Service dengan membuat referensi dari object Naming Service yang digunakan, lihat baris coklat
- Daftarkan referensi object yang didapatkan dari langkah sebelumnya ke Naming Service dengan diwakili sebuah nama, lihat warna biru tua
- Jalankan, tunggulah sampai ada permintaan dari client.

Contoh

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

public class HelloServer {
  public static void main(String args[]) {
    try {
      ORB orb = ORB.init(args, null);
      HelloImpl impl = new HelloImpl(orb);
      POA rootpoa =
          POAHelper.narrow
(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

     org.omg.CORBA.Object ref =
          rootpoa.servant_to_reference(impl);

     Hello href = HelloHelper.narrow(ref);
```

Contoh

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

       String name = "Hello";
        NameComponent path[] = ncRef.to_name( name );
        ncRef.rebind(path, href);

       System.out.println("HelloServer siappp grak...");
        orb.run();
     }
     catch (Exception e) {
       System.out.println("Ada kesalahan sistem!");}
  }
}
```

Pengembangan Aplikasi CORBA

- Buat program client
  - Inisialisasi obyek ORB
  - Ambil referensi obyek dari NameService
  - Kemudian masukkan hasil pengambilan obyek referensi ke suatu variable obyek lokal dan manipulasilah obyek lokal tersebut!

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
class HelloClient{
  public static void main(String args[]){
    try{
      ORB orb = ORB.init(args, null);
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      NamingContext ncRef =
          NamingContextHelper.narrow(objRef);
```

```
        NameComponent nc = new NameComponent("Hello", "");
        NameComponent path[] = {nc};
        Hello helloRef = HelloHelper.narrow(
                                 ncRef.resolve(path));
        System.out.println(helloRef.sayHello());
      } catch(Exception e) {
        System.out.println("ERROR : " + e);
      }
    }
}
```

Pengembangan Aplikasi CORBA

- Kompilasi dengan perintah:
  - javac HelloApp/*.java
  - javac HelloImpl.java
  - javac HelloServer.java
  - javac HelloClient.java
- Jalankan naming service
- Jalankan server
  - java HelloServer -ORBInitialPort 50000
- Hasilnya:
  - HelloServer siappp grak...
- Jalankan client
  - java HelloClient -ORBInitialPort 50000
- Hasilnya:
  - Hello world!!

Praktikum