Rosa Ariani Sukamto
Email: rosa_if_itb_01@yahoo.com
Website: http://www.gangsir.com

Common Object Request Broker Architecture (CORBA)
(Minggu 4 – Praktikum 1)

1. Sekilas CORBA

Common Object Request Broker Architecture **(CORBA)** adalah teknologi yang dipergunakan untuk *heterogeneous computing* (sistem komputer dengan berbagai macam lingkungan). CORBA pada dasarnya menggunakan arsitektur client-server dimana klien dan server berupa objek. CORBA mendukung apa yang disebut interoperabilitas, yaitu kemampuan saling bekerjasama antar sistem computer.

CORBA berbeda dengan RMI, berikut perbedaan CORBA dengan RMI:

1. CORBA adalah dapat diimplementasikan dengan sembarang bahasa pemrograman.
2. CORBA terdiri dari beberapa mekanisme dimana RMI dapat termasuk di dalamnya.
3. Pada RMI tidak menggunakan ORB (*Object Request Broker*).

Kenapa ada CORBA?

- Dapat menangani keberagaman lingkungan antara klien dan server (dapat diimplementasikan pada bahasa pemrograman yang berbeda). Hal ini karena CORBA menggunakan apa yang disebut antarmuka (*interface*) untuk menjembatani dua buah lingkungan yang berbeda.

Object Request Broker **(ORB)** merupakan inti dari CORBA dan bertanggung jawab untuk menjalankan semua mekanisme yang dibutuhkan, yaitu:

1. Menemukan implementasi objek untuk memenuhi suatu *request*
2. Menyiapkan implementasi objek untuk menerima suatu *request*
3. Melakukan komunkasi data untuk memenuhi suatu *request*

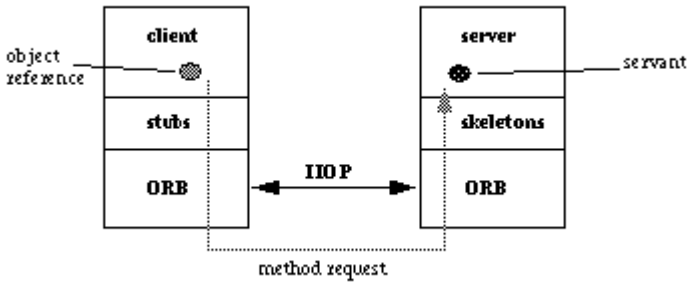
Sebuah permintaan (*request*) yang dikirimkan suatu *client* ke suatu *object implementation* akan melewati ORB. Dengan ORB, yang terdiri dari *interface*, suatu *client* dapat berkomunikasi dengan *object implementation* tanpa adanya batasan *platform*, teknologi jaringan, bahasa pemrograman, dan letak objek. Dengan menggunakan ORB, objek *client* bias meminta sebuah *method* pada sebuah *object server* yang bisa saja terdapat dalam satu mesin maupun jaringan yang berbeda. ORB menerima panggilan dan menemukan objek yang bisa mengimplementasikan permintaan, mengirim parameter, *invoke method*, dan mengembalikan hasil yang diperoleh.

Objek-objek CORBA dispesifikasikan menggunakan *interface*, yang merupakan penghubung anatara *client* dan server. *Interface Definition Language* (IDL) digunakan untuk mendefinisikan *interface* tersebut. IDL menentukan tipe-tipe suatu objek dengan mendefinisikan *interface-interface* objek tersebut. Sebuah *interface* terdiri dari kumpulan operasi dan parameter operasi tersebut. IDL hanya mendeskripsikan *interface*, tidak mengimplementasikannya. Meskipun sintaks yang dimiliki oleh IDL menyerupai sintaks bahasa pemrograman C++ dan Java., perlu diingat, IDL bukan bahasa pemrograman.

CORBA mendefinisikan IIOP *(Internet Inter-ORB Protocol)* untuk mengatur bagaimana objek berkomunikasi melalui jaringan. IIOP merupakan *open protocol* yang berjalan diatas TCP/IP.

Pada Java, CORBA merupakan pelengkap untuk menyediakan *framework* distribusi objek, *services* pendukung *framework* itu, dan kemampuan antar operasi dengan bahasa pemrograman lainnya. CORBA untuk *client-server* menggunakan protokol IIOP (*Internet InterORB Protocol*) untuk komunikasi antara server dan klien.

Arsitektur CORBA adalah sebagai berikut:



Skeletons adalah bagian kode yang dibangin pada kode implementasi server pada antarmuka (*interface*). Stub adalah bagian kode yang membuat antarmuka (*interface*) dapat diakse (*available*) oleh klien

Java menyediakan ORB (Object Request Broker) yang mendukung teknologi CORBA. ORB adalah komponen *runtime* yang dapat digunakan untuk *distributed computing* menggunakan komunikasi IIOP. OMG (Object Management Group) adalah industri yang membuat spesifikasi dan mempublikasikan CORBA.

Java IDL (*Interface Definition Langauge*) merupakan sebuah teknologi untuk distribusi objek yang berinteraksi antar platform. CORBA menggunakan IDL untuk membuat antarmuka (*interface*). Java IDL adalah implementasi dari teknologi antarmuka pada CORBA. Model pemrograman IDL atau biasa disebut Java IDL terdiri dari ORB CORBA dan kompiler idlj yang memetakan OMG IDL ke Java dengan menggunakan Java CORBA ORB. Aplikasi CORBA dibuat dengan menggunakan IDL untuk mendefinisikan antarmuka objek agar dapat diakses oleh klien maupun server.

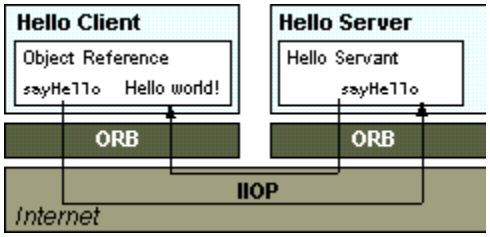
2. Praktikum

a. Persiapan

- Membuat direktori kerja dengan nama SI319-P4-1-Kelas-NIM misalnya SI319-P4-1-A-23507024
- Di dalam direktori di atas, buat direktori Hello untuk menyimpan file-file yang akan dibuat.

b. Hello World

Arsitektur CORBA untuk Hello World



i. IDL

Membuat file dengan nama Hello.idl pada direktori Hello sebagai berikut:

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

Nama modul (HelloApp) di atas tidak boleh dipakai lagi untuk nama antarmuka (unik) agar saat aplikasi dijalankan bisa benar-benar menghasilkan hasil yang benar (tidak bentrok). File .idl menggenerasi stub dan skeletons seperti pada arsitektur umum CORBA.

ii. Server

Server terdiri dari dua buah kelas yaitu HelloImpl dan HelloServer:

- HelloImpl merupakan implementasi dari antarmuka IDL yang berfungsi sebagai servant. Sebuah servant terdiri dari sebuah metode untuk setiap operasi IDL seperti metode sayHello() pada praktikum ini.
- HelloServer memiliki metode main yang melakukan proses-proses berikut:
    a. Membuat instans ORB
    b. Membuat instans servant (implementasi dari sebuah objek Hello dengan CORBA) dan memberitahu ORB mengenai servant

c. Mengambil referensi objek CORBA untuk penamaan context untuk meregister objek baru CORBA
d. Meregister objek baru pada penamaan context dengan nama Hello
e. Menunggu invokasi objek baru

Langkah-langkah yang harus dilakukan untuk server dengan CORBA pada Java adalah sebagai berikut:

1. Membuat file HelloServer.java

   Nama file : HelloServer.java

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

import java.util.Properties;

class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implementasi metode sayHello()
  public String sayHello() {
      // isi dengan NIM kode di bawah ini,
      //misalnya menjadi "\nHello world !! : [1206001]\n"
    return "\nHello world !! : [isi dengan NIM]\n";
  }

  // implementasi shutdown()
  public void shutdown() {
    orb.shutdown(false);
  }
}


public class HelloServer {

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

      // membuat referensi ke rootpoa dan mengaktifkan POAManager
      POA rootpoa =
        POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
```

```
      // membuat servant dan meregisternya ke ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // mengambil objek referensi dari servant
      org.omg.CORBA.Object ref =
         rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // mengambil root naming context
      // NameService memanggil layananan penamaan
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Menggunakan NamingContextExt yang merupakan
      // bagian dari Interoperable
      // Spesifikasi Naming Service (INS).
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // bind objek yang diacu dengan menggunakan penamaan
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);

      System.out.println("HelloServer ready and waiting ...");

      // menunggu panggilan klien
      orb.run();
    }

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

      System.out.println("HelloServer Exiting ...");

  }
}
```

Simpan dan tutup file.

2. Memahami HelloServer.java
    - Server pada CORBA sama dengan server pada aplikasi Java pada umumnya dimana diperlukan mengimpor pustaka, mendeklarasikan kelas server, metode main(), dan penanganan eksepsi. Server CORBA membutuhkan objek ORB lokal. Setiap server menginisialisasi sebuah ORB dan meregister objek servant sehingga ORB dapat mendeteksi server saat sebuah invokasi sampai pada server.
    - Deklarasi dan inisialisasi ORB dilakukan di dalam blok try-catch dengan kode sebagai berikut:

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

- Server harus mengambil nilai dari root POA (*Portable Object Adapter*) dan mengaktifkan POAManager. POA adalah sebuah mekanisme yang menghubungkan sebuah *request* menggunakan objek yang diacu dengan kode yang sesuai untuk memenuhi request itu.
- Sebuah server merupakan sebuah proses yang menginstansiasi (menggunakan) satu atau lebih objek servant. Servant mengimplementasikan antarmuka yang digenerasi oleh idlj dan mengerjakan operasi yang ada pada antarmuka itu. Pada praktikum ini sebuah HelloServer membutuhkan sebuah servant.
- HelloServer bekerja dengan naming service yang membuat operasi objek servant dapat diakses oleh klien. Server membutuhkan sebuah referensi objek ke name service sehingga dapat meregister dirinya sendiri dan memastikan invokasi ke antarmuka Hello diteruskan ke objek servant-nya.
- Servant diinstansiasi dalam blok try-catch dengan kode:

  ```
  orb.resolve_initial_references()
  ```

  untuk mendapatkan sebuah referensi objek ke nama server
- Pada kode :

  ```
  org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
  ```

  String “NameService” didefinisikan untuk semua ORB CORBA sehingga ORB dapat mengembalikan objek naming context yang merupakan objek referensi untuk name service.
- objRef adalah objek CORBA secara umum, untuk menggunakannya sebagai objek NamingContext diperlukan proses mengkhususkan objek itu dengan memanggil metode narrow() dengan menuliskan kode :

  ```
  NamingContext ncRef = NamingContextHelper.narrow(objRef);
  ```

- Path dan objek servant dilewatkan ke naming service dan mengaitkan (binding) objek servant ke id “Hello”.

  ```
  ncRef.rebind(path, href);
  ```

  sehingga ketika klien memanggil resolve(“Hello”) pada inisial naming context, naming service akan mengembalikan referensi objek ke servant.
  Kode-kode sebelumnya merupakan kode-kode untuk menyiapkan server, setelahnya server tinggal menunggu invokasi dari klien ke ORB dengan kode

  ```
  Orb.run();
  ```

iii. Client

Langkah-langkah yang harus dilakukan untuk klien dengan CORBA pada Java adalah sebagai berikut:

1. Membuat HelloClient.java

Nama file: HelloClient.java

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

public class HelloClient
{
  static Hello helloImpl;

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

        // mengambil root naming context
        org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
        // menggunakan NamingContextExt yang
        // merupakan bagian dari Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // mengacu Object yang diacu penamaan
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: "
          + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

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

}
```

Simpan pada direktori Hello dan tutup file.

2. Memahami HelloClient.java
    - Klien CORBA membutuhkan ORB lokal untuk proses *marshaling* dan kerja IIOP. Setiap klien menginstansiasi sebuah `org.omg.CORBA.ORB` dan menginisialisasi ORB. ORB dideklarasikan dan diinisialisasi di dalam blok try-catch.

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

- Dipanggil orb.resolve_initial_references() untuk mendapatkan naming context, ditulis dalam blok try-catch dengan kode:

```
org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
```

String “NameService” didefinisikan untuk semua ORB CORBA. Ketika string tersebut dilewatkan maka ORB akan mengembalikan initial naming context yang merupakan referensi objek ke name service

- objRef adalah objek CORBA secara umum, untuk menggunakannya sebagai objek NamingContext diperlukan proses mengkhususkan objek itu dengan memanggil metode narrow() dengan menuliskan kode :

```
NamingContext ncRef =
  NamingContextHelper.narrow(objRef);
```

Komplikasi dari CORBA adalah memaket (*marshaling*) parameter ke koneksi dan menghubungkannya ke ORB pada sisi server, membongkar paket dan memanggil metode pada sisi server seperti yang diminta oleh klien. Klien memanggil metode pada sisi server dan mencetaknya ke layar dengan kode

```
System.out.println(helloImpl.sayHello());
```

iv. Menjalankan aplikasi

- Buka Command prompt, masuk ke direktori dimana file .idl disimpan
- Jalankan kompiler IDL-to-Java (idlj) pada file .idl untuk membuat stub dan skeletons dengan perintah (jika perintah tidak dikenali, set path ke jdk):

```
idlj -fall  Hello.idl
```

idlj akan menggenerasi beberapa file sebagai berikut:

- `HelloPOA.java`
  Merupakan kelas abstrak yang menyediakan fungsi dasar server CORBA. Kelas HelloImpl pada server merupakan turunan dari kelas HelloPOA.
- `_HelloStub.java`
  Merupakan kelas yang merepresentasikan stub pada sisi klien yang menyediakan fungsi CORBA untuk klien yang diimplementasikan oleh antarmuka Hello.java
- `Hello.java`

Antarmuka ini berisi versi Java dari antarmuka IDL. Hello.java merupakan turunan dari `org.omg.CORBA.Object` yang menyediakan fungsi objek standar CORBA

- `HelloHelper.java`
  Merupakan kelas final yang menyediakan fungsi pembantu, seperti metode narrow() yang dibutuhkan untuk melakukan *casting* pada referensi objek CORBA menjadi tipe yang dibutuhkan
- `HelloHolder.java`
  Kelas final yang menangani anggota instans public dari tipe Hello. Kelas ini menyediakan operasi keluar masuknya argumen yang diijinkan CORBA tapi tidak mudah dipetakan pada Java
- `HelloOperations.java`
  Kelas antarmuka ini berisi sebuah metode sayHello(). IDL-to-Java memetakan semua operasi yang didefiniskan pada antarmuka IDL ke dalam file ini dimana dapat diakses oleh server maupun klien

- Ketik perintah:

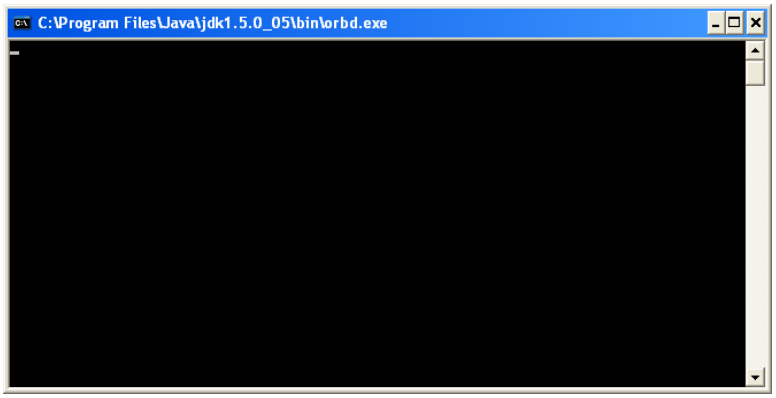
```
javac *.java HelloApp/*.java
```

- Ketik perintah:

```
start orbd -ORBInitialPort 1050
```

yang berarti bahwa port 1050 dipilih untuk naming naming service, port dapat diganti port lain jika port 1050 telah terpakai, jika port tidak dituliskan pada perintah maka port standar yang digunakan adalah 900

hingga muncul jendela berikut:



- ketik perintah

```
start java -cp . HelloServer -ORBInitialPort 1050 -ORBInitialHost
localhost
```

hingga muncul jendela commad prompt dengan tulisan “`HelloServer ready and waiting`”

- ketik perintah

```
java -cp . HelloClient -ORBInitialPort 1050 -ORBInitialHost
localhost
```

catatan:
jika server atau client dijalankan pada mesin yang berbeda maka perintah yang harus dijalankan adalah:

```
java –cp . [HelloServer/HelloClient] -ORBInitialHost
namerserverhost -ORBInitialPort 1050
```

- Matikan ORB dengan perintah (pada Command prompt orbd) dengan menekan tombol ctrl+C