MEMBANGUN APLIKASI TERDISTRIBUSI DENGAN CORBA DAN JAVA

Emha Taufiq Luthfi

Abstraksi

Arsitektur aplikasi terdistribusi yang memodelkan semua fungsionalitas sistem dalam bentuk obyek memberikan banyak fleksibilitas. Fleksibilitas di dapat dengan pendefinisian komponen antarmuka yang menjelaskan layanan dalam komponen serta bagaimana pemanfaatannnya. CORBA menyediakan framework untuk membangun dan menjalankan aplikasi terdistribusi. Artikel ini menjelaskan langkah membangun aplikasi terdistribusi beserta contoh dengan CORBA dan java.

Pendahuluan

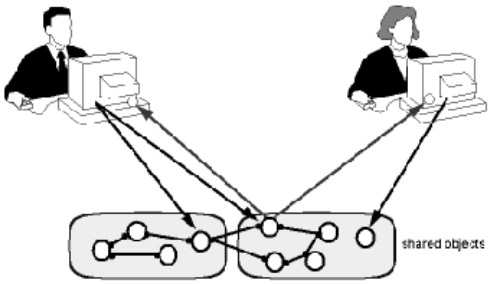
Dalam membangun suatu aplikasi, terkadang di dapatkan situasi tanpa pilihan bahwa aplikasi harus terdistribusi dikarenakan alasan :

- Data yang digunakan aplikasi berada pada beberapa lokasi berlainan.
- Komputasi dilakukan secara terdistribusi.
- Pengguna aplikasi berada pada lokasi yang berlainan.

Beberapa aplikasi harus dijalankan pada beberapa komputer dikarenakan data yang dibutuhkan oleh aplikasi harus berada di beberapa komputer untuk alasan administrasi dan kepemilikan data tersebut. Pemilik mungkin mengijinkan data untuk diakses dari lokasi lain tetapi tidak untuk disalin untuk akses secara lokal. Atau mungkin data tidak dapat dipindahkan dan harus berada pada beberapa sistem yang heterogen karena alasan tertentu.

Beberapa aplikasi dijalankan pada beberapa komputer dikarenakan kebutuhan untuk mendapatkan keuntungan komputasi dengan beberapa prosesor secara pararel untuk menyelesaikan beberapa fungsi. Aplikasi lain mungkin di jalankan pada beberapa komputer untuk mendapatkan keuntungan terhadap fitur khusus dari suatu sistem.

Beberapa aplikasi dijalankan pada beberapa komputer dikarenakan pengguna aplikasi berada pada lokasi berbeda, saling berkomunikasi dan berinteraksi melalui aplikasi. Setiap pengguna menjalankan sebagian dari aplikasi terdistribusi pada komputer mereka, berbagi fungsi yang biasanya dijalankan pada satu atau beberapa server.



Gambar 1 Ilustrasi pengguna terdistribusi

Sistem Terdistribusi

Arsitektur aplikasi terdistribusi dikembangkan dari konsep multitier client/server. Jika multitier client/server fokus pada pemisahan business logic dengan data access, sistem terdistribusi memodelkan semua fungsionalitas sistem dalam bentuk obyek yang mana suatu obyek dapat memanfaatkan layanan dari obyek lain dalam sistem yang sama atau bahkan layanan obyek dari sistem lain. Arsitektur ini sedikit mengaburkan batasan antara client dan server karena komponen client dapat membentuk obyek yang berlaku seperti suatu server.

Sistem terdistribusi memberikan fleksibilitas dengan adanya pendefinisian komponen antarmuka (*interface*). Antarmuka dari komponen akan memberikan spesifikasi kepada komponen lain tentang layanan (*services*) yang dapat diberikan komponen tersebut serta cara penggunaannya.

Antarmuka mendefinisikan protokol komunikasi antara dua komponen sistem yang terpisah (komponen dapat berupa proses terpisah, obyek terpisah, pengguna dan aplikasi serta berbagai entity lain yang terpisah dan butuh untuk melakukan komunikasi). Antarmuka menjelaskan layanan yang disediakan oleh suatu komponen serta protokol untuk menggunakan layanan tersebut. Sistem terdistribusi sesungguhnya merupakan sistem multitier client/server dengan jumlah client dan server dimungkinkan sangat banyak. Salah satu perbedaan penting bahwa sistem terdistribusi biasanya menyediakan layanan tambahan seperti *directory services* yang memungkinkan komponen dari aplikasi ditemukan oleh lainnya. Layanan lain berupa *transaction monitor services* yang memungkinkan komponen untuk melakukan transaksi dengan lainnya.

CORBA

CORBA atau *Common Request Object Broker Architecture* merupakan arsitektur standar untuk sistem obyek terdistribusi. CORBA memungkin berbagai obyek heterogen yang terdistribusi untuk saling berhubungan. Sebelum populer penggunaannya pada *World Wide Web* serta secara khusus pada bahasa pemrograman Java. CORBA merupakan solusi obyek terdistribusi yang digunakan oleh pengembang C++.

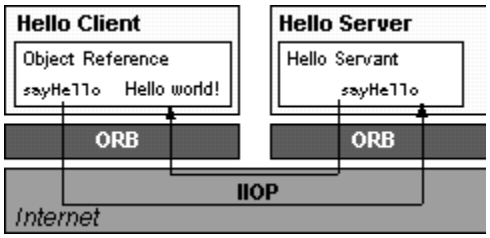
Spesifikasi CORBA dikendalikan oleh Object Management Group (OMG) sebagai konsorsium terbuka dari kurang lebih 700 perusahaan yang bekerja bersama mendefinisikan standar untuk komputasi obyek. Obyek CORBA dapat ditulis dengan beberapa bahasa pemrograman yang didukung oleh perusahaan software CORBA seperti C, C++, Java, Ada atau SmallTalk. Obyek juga dapat berada pada berbagai platform yang didukung oleh perusahaan software CORBA seperti Solaris, Window 95/NT, OpenVMS, Digital Unix, HP-UX dan AIX dan beberapa lainnya. Ini berarti bahwa kita dapat memiliki aplikasi Java yang berjalan pada Windows 95 yang secara dinamik menjalankan dan menggunakan obyek C++ yang diletakkan melalui Internet pada server web pada mesin unix.

Fleksibilitas dalam bahasa dimungkinkan dengan pembuatan antarmuka dari obyek dengan menggunakan Interface Description language (IDL). IDL memungkinkan semua obyek CORBA untuk didefinisikan dengan cara yang sama. Yang dibutuhkan hanya perantara antar bahasa yang berbeda (C / C++, COBOL, Java) dan IDL. Obyek CORBA saling berkomunikasi menggunakan Object Request Broker (ORB) sebagai perantara, dan dapat berkomunikasi melalui berbagai protokol jaringan yang ada (TCP/IP atau IPX/SPX). ORB dari vendor yang berbeda berkomunikasi melalui TCP/IP menggunakan Inter-ORB Protocol (IIOP), yang merupakan bagian dari CORBA 2.0.

Saat ini, third-party ORB telah tersedia untuk beberapa bahasa pemrograman (C++, Smaltalk, Java dan Ada95). Ketika terdapat suatu bahasa pemrograman yang berkembang, maka vendor CORBA akan menyediakan ORB untuk bahasa tersebut. OMG awalnya mendefinisikan Object Management Architecture (OMA) pada 1990 untuk mendefinisikan bagaimana aplikasi dapat saling berhubungan. Untuk memenuhi kebutuhan tersebut, diperlukan standar yang menjelaskan bagaimana bagian atau obyek dari aplikasi dapat berhubungan, lahirlah CORBA. OMA mendefinisikan empat bagian utama dari CORBA :

1. Object Request Broker (ORB), berfungsi sebagai software perantara bagi obyek untuk saling berhubungan.

2. CORBAServices, mendefinisikan layanan pada level sistem yang ditambahkan pada ORB seperti Security, Naming dan Transactions.
3. CORBAFacilities, mendefinisikan layanan level aplikasi seperti penggabungan dokumen dan fasilitas lain.
4. Business Object mendefinisikan obyek sesungguhnya.



Gambar 2 Ilustrasi Salah Satu metode Berbagi Obyek Terdistibusi antara Client dan Server CORBA

Object Request Broker (ORB)

ORB merupakan bagian dari CORBA yang harus ada dalam membangun aplikasi CORBA. Tanpa ORB aplikasi CORBA tidak akan berfungsi. Fungsi utama ORB CORBA adalah untuk merespon permintaan dari aplikasi atau dari ORB lain. Selama siklus hidup dari aplikasi CORBA yang berjalan, ORB mungkin akan diminta untuk melakukan beberapa hal yang berbeda, antara lain :

- Menemukan dan membentuk obyek pada mesin yang berbeda
- Melewatkan parameter dari satu bahasa pemrograman dari satu bahasa pemrograman (misal C++) ke bahasa pemrograman (misal Java)
- Menangani keamanan dalam mesin lokal
- Mengambil dan mempublish metadata pada obyek pada sistem lokal pada ORB lain.
- Menjalankan metode pada obyek pada lokasi yang berbeda menggunakan metode statik.
- Menjalankan secara otomatis obyek yang belum berjalan

Kekuatan dari ORB adalah seluruh detail implementasi untuk semua fungsi di sembunyikan dari pengembang. Sehingga dengan sederhana dapat digunakan hanya dengan menambahkan pada kode aplikasi proses inisialisasi ORB dan mendaftarkan ORB dari aplikasi tesebut untuk dapat menghubungkan aplikasi dengan jaringan obyek terdistribusi.

Mendefinisikan Obyek dengan IDL

Untuk menjaga agar CORBA bebas vendor dan bebas penggunaan bahasa, harus ada perantara antara server CORBA sebagai contoh dengan C++ dengan sebagai contoh client CORBA dengan Java. Perantara tersebut adalah IDL. Metode-metode yang berhubungan dan property didukung oleh obyek dikumpulkan bersama dalam sebuah IDL. IDL yang telah didefinisikan di kompile menjadi bahasa pemrograman yang diinginkan dengan kompiler yang tersedia. Misal, Java/IDL kompiler terdapat pada Visigenic Visibroker untuk ORB Java atau Java 2 ORB yang terdapat dalam Sun's JDK 2 SDK, C++/IDL kompiler terdapat dalam Visigenic Visibroker untuk ORB C++.

Selengkapnya untuk pengembangan aplikasi, semisal dengan Java IDL diperlukan beberapa tahapan yaitu :

- Pendefinisian Antarmuka (*Interface*)
- Mengkompile Antarmuka
- Penerapan Antarmuka untuk Aplikasi Server
- Penerapan Antarmuka untuk Aplikasi Client

Listing 1 adalah contoh pendefinisian IDL dalam file Hallo.idl dengan nama Modul adalah HaloSayang. Di dalamnya terdapat satu buah antarmuka Hallo.

Listing 1 : Mendefinisikan antarmuka dengan file Hallo.idl

```
module HalloApp {
    interface Hallo {
    string ucapkanHallo();
        oneway void shutdown();
    };
};
```

Untuk mendapatkan versi java dari dari antarmuka diatas maka digunakan kompiler idlj yang terdapat dalam Sun's JDK 2 SDK, dengan perintah kompile :

```
idlj –fall Hallo.idl
```

Kompiler idlj akan menghasilkan beberapa file, untuk kasus ini :

- HalloPOA.java
  Merupakan class abstract yang menyediakan dasar fungsionalitas server CORBA.

- _HalloStub.java
  Merupakan class yang menyediakan fungsionalitas untuk client CORBA

- Hallo.java
  Merupakan antarmuka (*interface*) versi java dari antarmuka yang dibuat dalam file Hallo.idl. Menyediakan fungsionalitas dasar obyek CORBA.

- HalloHelper.java
  Merupakan class yang menyediakan fungsionalitas pendukung. Class ini bertanggung jawab untuk membaca dan menulis tipe data ke streams CORBA.

- HalloHolder.java
  Class Holder memberikan delegasi kepada metode dari class Helper untuk melaksanakan tugas membaca dan menulis.

- HalloOperations.java
  Merupakan antarmuka yang mengandung metode ucapkanHalo() dan shutdown. Kompiler IDL ke java meletakkan semua operasi yang didefinisikan dalam antarmuka IDL pada file ini.

Selanjutnya pembuatan server, dalam kasus ini terdiri dua class yaitu servant dan server. Servant, HalloImp yang merupakan penerapan dari antarmuka IDL Hallo. Setiap instance Hallo di implementasikan oleh HalloImpl. Servant mengandung sebuah metode untuk setiap operasi dalam IDL, ucapkanHallo() dan shutdown().
Class server memiliki metode main(), yang melakukan proses :

- Membuat dan menginisialisasi instance ORB
- Membuat referensi ke root POA dan mengaktifkan POAManager
- Membuat instance Servant dan memberitahukan ORB tentang hal tersebut
- Mendapatkan referensi dari obyek CORBA untuk penamaan yang dalam rangka melakukan register obyek CORBA baru.
- Melakukan register pada obyek baru dengan nama Hallo
- Menunggu permintaan obyek baru dari client

Listing 2 : Mendefinisikan server dengan file HalloServer.java

```
import HalloApp.*;
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 HalloImpl extends HalloPOA {
  private ORB orb;
  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implementasi metode ucapkanHallo()
  public String ucapkanHallo() {
    return "\nHallo Jogjakarta !!\n";
  }

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

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

      // referensi ke rootpoa & mengaktifkan POAManager
      POA rootpoa = POAHelper.narrow(
                    orb.resolve_initial _references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // buat servant dan register-kan ke ORB
      HalloImpl HalloImpl = new HalloImpl();
      HalloImpl.setORB(orb);

      // dapatkan referensi obyek dari servant
      org.omg.CORBA.Object ref =
                        rootpoa.servant_to_reference(HalloImpl);
      Hallo href = HalloHelper.narrow(ref);

      // dapatkan root naming context
      org.omg.CORBA.Object objRef =
               orb.resolve_initial_references("NameService");

      NamingContextExt ncRef =
               NamingContextExtHelper.narrow(objRef);

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

      System.out.println("HalloServer aktif dan
                              menunggu permintaan ...");
```

```
      // tunggu permintaan dari client
      orb.run();
    }

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

      System.out.println("HalloServer dimatikan ...");

  }
}
```

Kompile dilakukan dengan perintah :

```
Javac HalloServer.java HalloApp/*.java
```

Listing 3 : Mendefinisikan Client dengan file HalloClient.java

```
İmport HalloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HalloClient
{
  static Hallo HalloImpl;
  public static void main(String args[])
    {
      try{
        // membuat dan menginisalisasi ORB
       ORB orb = ORB.init(args, null);
        // ambil root naming context
        org.omg.CORBA.Object objRef =
               orb.resolve_initial_references("NameService");
         NamingContextExt ncRef =
               NamingContextExtHelper.narrow(objRef);

        String name = "Hallo";
        HalloImpl = HalloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("penanganan obyek server : " +
                           HalloImpl);
        System.out.println(HalloImpl.ucapkanHallo());
        HalloImpl.shutdown();
```

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

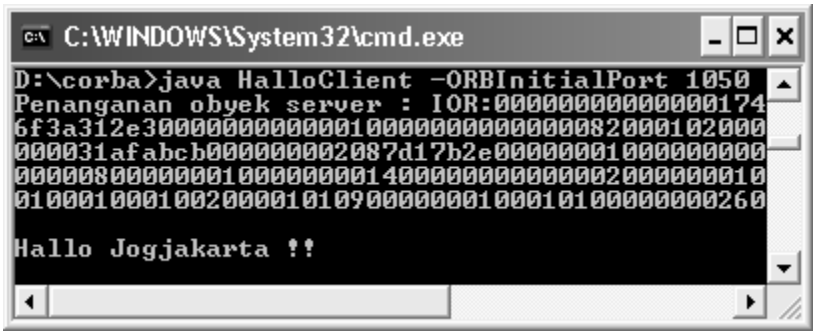
Kompile dilakukan dengan perintah :

```
Javac HalloClient.java
```

Dan jalankan aplikasi dengan langkah :

- jalankan orbd
  start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
- jalankan server
  start java HalloServer -ORBInitialPort 1050 -ORBInitialHost localhost
- jalankan client
  java HalloClient -ORBInitialPort 1050 -ORBInitialHost localhost

Gambar 3 HalloServer aktif dan siap menerima permintaan



Gambar 4 HalloClient memanggil metode dari obyek HalloServer

Alternatif Obyek Terdistribusi dengan DCOM dan RMI

Rata-rata pengembangan java telah lebih dahulu familiar dengan pesaing-pesaing utama CORBA, DCOM dan RMI.

- CORBA

  CORBA memiliki beberapa keuntungan utama, diantaranya independen dalam bahasa, vendor dan sistem operasi. ORB CORBA tersedia pada beberapa sistem operasi yang banyak digunakan saat ini (Termasuk pada sistem operasi microsoft dibandingkan dengan DCOM). ORB CORBA juga tersedia untuk digunakan secara luas pada berbagai bahasa pemrograman termasuk C++, Ada, COBOL, SmalTalk dan Java. Dengan menggunakan IIOP, ORB CORBA yang dikembangkan oleh suatu vendor dapat diambil atau memanipulasi obyek dari ORB lain yang dikembangkan oleh vendor lain.

- DCOM

  Microsoft Distributed Component Object Model (DCOM) saat ini secara lengkap tersedia hanya pada dua sistem operasi yaitu : Windows 95 dan Windows NT 4. Microsoft bekerja dengan third-party vendors untuk menambahkan DCOM pada sistem operasi lain (termasuk beberapa Unix dan MVS). DCOM seperti CORBA, independen dalam bahasa dan obyek didefinisikan melalui interface yang ada yaitu Microsoft Object Description Language (ODL).

  DCOM memiliki 3 kekurangan dibandingkan dengan CORBA. Pertama, DCOM di definisikan dan dikendalikan oleh seorang vendor (Microsoft) yang secara luas akan mengurangi pilihan bagi pengembang DCOM ketika bekerja (misal dalam tool dan feature). Kedua, DCOM dibatasi pada n platform sehingga mengurangi reusable dari kode pada aplikasi DCOM. Dan yang terakhir, tehnologi yang terdapat ada DCOM masih kurang dibanding dengan tehnologi dari CORBA.

  Misal, Menulis applet java atau aplikasi java untuk mengakses obyek DCOM dari server akan membuat pengembang membatasi software client yang mendukung hanya dengan browser Ms Internet Explorer dan platform pada window 95 / window NT. Batasan ini tentu saja akan mengurangi fleksibilitas aplikasi.

- RMI

  Remote Method Invocation adalah feature yang terdapat mulai versi JDK 1.1. RMI memungkin client java untuk membentuk obyek yang mungkin terletak pada remote server. Seolah seperti CORBA, akan tetapi dalam RMI aplikasi server juga harus ditulis dengan Java dan juga harus menggunakan tools yang disediakan pada JDK 1.1, RMI memiliki batasan.

  Tidak seperti CORBA, RMI tidak memiliki konsep layanan. Selain itu, obyek server yang ditulis dengan RMI java memiliki performance yang kurang karena batasan

penggunaan JVM (CORBA memiliki performance lebih). Singkatnya, RMI merupakan pilihan untuk aplikasi dengan skala kecil yang keseluruhannya ditulis dengan java.

Kesimpulan

Sistem terdistribusi merupakan pengembangan arsitektur multitier client/server yang memodelkan semua fungsionalitas sistem dalam obyek yang dapat saling berinteraksi. Pemanfaatan CORBA dalam sistem terdistribusi akan menghasilkan obyek yang independen terhadap bahasa pemrograman serta platform dengan penggunaan Interface Definition Language (IDL) sebagai antarmuka yang mendefinisikan suatu komponen, layanan yang dimiliki serta penggunaanya.

Pustaka

Jeremy Rosenberger, *Teach Yourself CORBA In 14 Days*, SAMS Publishing

Bryan Morgan, *Distributed objects meet the Web*, http://www.javaworld.com/jw-10-1997/corba/jw-10-corbajava.zip

JDK 2 SDK Documentation