CORBA 101

GKO 16.11.2000
MIT
Jyväskylän yliopisto

Jonne Itkonen
jonne.itkonen@sonera.com
Software Architectures Laboratory
Mobile & Media Research
Sonera

(2004-8-6: Ylläoleva osoite ei enää toimi, enkä ole enää Soneran palveluksessa, vaan Jyväskylän yliopiston, joten osoite on ji@mit.jyu.fi. Lisäsin myös omniORBin linkin ja korjasin JacORBin linkin. Inprisekin on nykyään taas Borland.)

Yleistä

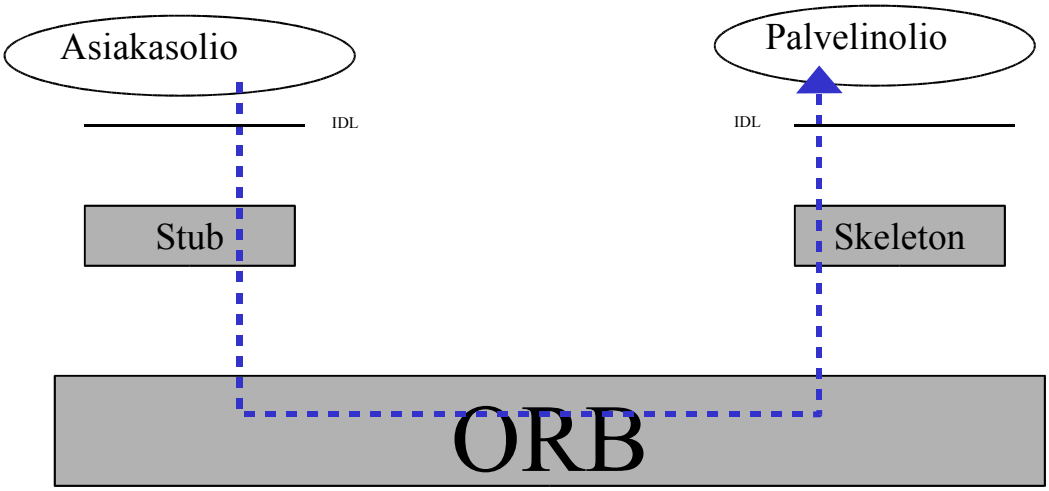
CORBA, eli Common Object Request Broker Architecture on Object Management Groupin määrittelemä kuvaus siitä, kuinka hajautettuja olio-arkkitehtuureja tulisi toteuttaa. CORBA määrittelee yhdyskäytävän olioiden väliselle "viestinnälle". Tämän yhdyskäytävän tarkoituksena on välittää metodikutsut sovellukselta oliototeutukselle, tai tarvittaessa toisinpäin (sovellus voi tietysti olla, ja usein onkin, toinen olio). Yhdyskäytävää kutsutaan Object Request Brokeriksi (ORB).

CORBA määrittelee myös joukon peruspalveluita, jotka tulisi aina olla saatavilla määritykset täyttävässä ORB-toteutuksessa. Näitä peruspalveluita ovat esimerkiksi nimipalvelu, viestinvälityspalvelu, olioidenvälityspalvelu ja aikapalvelu.

Toiminta ja arkkitehtuuri

CORBAn oliot ovat yhteydessä toisiinsa ORBin välityksellä. Asiakasolion kutsuessa palvelinolion operaatiota muunnetaan operaatiokutsu IOP:n (Inter-ORB Protocol) mukaiseksi viestiksi, jonka ORB välittää palvelinoliolle. Palvelinolion ORB taas muuntaa viestin operaatiokutsuksi palvelinoliolle.

Operaatiokutsun muuntaminen IOP-viestiksi tapahtuu niinkutsutussa "stubissa", joka on palvelinolion rajapinnan IDL-kuvauksesta generoitu olio tai funktiokirjasto asiakasolion ja ORBin välillä. Vastaava muunnos IOP-viestistä operaatiokutsuksi palvelinolion ja ORBin välillä tapahtuu niinikään IDL-kuvauksesta generoidussa "skeleton" oliossa/funktiokirjastossa. Kuva selventänee asiaa:
CORBAssa olioiden rajapinnat määritellään IDL:n (Interface Definition Language) avulla. Luodut määrittelyt käännetään käytettävän toteutuskielen ohjelmiksi erillisellä kääntäjäohjelmalla. OMG on



määritellyt, millaisia rakenteita millekin kielelle syntyy IDL:n lauseista. OMG:n hyväksymiä kielikuvauksia löytyy Adalle, C/C++:lle, COBOLille, Lispille, Javalle ja Smalltalkille, sekä lähiaikoina virallistuva kuvaus Pythonille.

Termiviidakko

ORB	Object Request Broker
CORBA	Common Object Request Broker Architecture
IIOP	Internet Inter-ORB Protocol
IDL	Interface Definition Language
IOR	Interoperable Object Reference
POA	Portable Object Adaptor
(BOA	Bad Object Adaptor formerly known as Basic Object Adaptor)
Stub	"Rajapinta" asiakasohjelman ja ORBin välillä, joka näkyy asiakkaalle olion rajapintana muuttaen operaatiokutsut ORBin välittämään muotoon.
Skeleton	"Rajapinta" olion toteutuksen ja ORBin välillä, joka muuntaa ORBilta tulevat operaatiokutsut olion toteutuksen metodien kutsuiksi.

IDL syntaksi

- `module` *nimi* `{ … };`
  Jokainen rajapinta (interface) kuuluu johonkin moduuliin. Tavallaan moduulia voidaan pitää nimiavaruutena, joka sisältää rajapintojen lisäksi tyyppi- ja poikkeutusmäärittelyjä. Tiedostossa esiteltyyn moduuliin voi myöhemmin lisätä rajapintoja jossakin toisessa tiedostossa.
- `interface` *nimi* `{ … };`
  Tämä on rajapinnan yleisin esittelytapa. Aaltosulkeiden väliin tulee rajapinnan sisältämät operaatiot ja attribuutit.
- `interface` *nimi* `:` s_1`,` s_2`, … { … };`
  Jos rajapinta perii toisia rajapintoja, käytetään tätä muotoa.
- `interface` *nimi*`;`
  Ennaltaesittely tehdään näin.
- `attribute` *tyyppi nimi*`;`
  Tavallisen attribuutin esittely. Attribuutille generoidaan aksessointimetodit/-funktiot kielimäärittelyn mukaan.
- `readonly attribute` *tyyppi nimi*`;`
  Vain luettavissa oleva attribuutti esitellään näin.
- *tyyppi1 op_nimi*`(in|out|inout` *tyyppi2 nimi*`) raises (`*poikkeus1*`,…);`
  Operaatiot esitellään näin, kuten C/C++:ssa. Jos operaatio ei palauta arvoa, annetaan tyyppi$_1$:lle arvoiksi `void`.
- `onway` *tyyppi1 op_nimi*`(…) raises (…);`
  Operaation parametri voi myös olla yksisuuntainen, jolloin sen suoritusta ei jäädä odottamaan (asynchronous).
- `typedef` *tyyppi uusityyppi*`;`
  Tyypinmäärittely.
- `exception` *nimi* `{ … };`
  Poikeutuksen esittely. Aaltosulkeissa voi tarvittaessa esitellä attribuutteja.

Tavallisimmat tietotyypit ovat:

- `short, (long) long, float, (long) double, fixed`
  - myös unsigned versiot, esimerkiksi
    `unsigned long long`
- `char, wchar, string, wstring`
- `boolean, octet, any`
- `enum, union, struct`
- operaation paluuarvon tyyppinä myös `void`
- taulukot tyypinmäärittelyllä, esimerkiksi
  `typedef float` *Vektori*`[3];`
- `sequence<`*tyyppi, koko*`> nimi;`
  - kielestä riippuen luodulle uudelle tietotyypille luodaan operaatiot sen käsittelyyn

Vakiot merkitään avainsanalla const, esimerkiksi `const short foo=175;` Vakioiden sääntöjä ei käsitellä tarkemmin.

Lisäksi käytettävissä on esikääntäjäkomentoja

- `#define` ja `#include`
- `#pragma` (toteutuskohtaisia?)

Pari mielenkiintoista, mutta tämän esittelyn ulkopuolelle jäävää tyyppiä:

- `valuetype`
  Oliot ja muut kuin perustietotyypit välitetään CORBAssa yleensä viitteinä (object reference). Valuetypen avulla voidaan kuitenkin määritellä ja välittää olioita arvosemantiikan mukaan.
- `native`
  Olio toteutetaan ORBille natiivina.

Portable Object Adapter (POA)

Kaikkia olioita ei voi aina pitää ajossa palvelimessa. Tarvitaan järjestelmä, joka osaa automaattisesti käynnistää olion tarvittaessa. Lisäksi kaikki oliototeutukset eivät vaadi kaikkia mahdollisia palveluita alustaltaan, tai nimenomaan tarvitsevat jotain ei-oletuspalvelua. POA toimii näissä molemmissa tapauksissa välittäjänä oliototeutuksen ja sen ajoalustan välillä.

Oliototeutuksesta (“serveripää”)

Oliototeutuksen tärkein tehtävä on – yllätys, yllätys – toteuttaa IDL:llä määritelty olion rajapinta. Jos olion tila halutaan säilyttää käyttökertojen välillä (persistenttiys), saatetaan joissain tapauksissa tätä varten joutua ohjelmoimaan toimintoja olioon.

Yksinkertainen rakenne tälle ohjelmalle on seuraava:

1. Luo oliototeutus.
2. Jos tarvis, lue olion tila säilöstä.
3. Liitä se ORBiin; POA tai suoraan.
4. Rekisteröi olio nimipalveluun, tallenna IOR muiden saataville tai saata olio muuten muiden saataville esimerkiksi rekisteröimällä se Trader-palvelulle.
5. Siirry odottamaan operaatiokutsuja.
6. Kun/jos oliototeutus tuhotaan, tarpeen vaatiessa tallenna olion tila säilöön.

Tapana on luoda erillinen ohjelma, joka luo oliototeutuksen ja liittää sen ORBiin.

Asiakasohjelma

Asiakasohejelmalle jää tehtäväksi löytää viite haluamaansa olioon. Tämä tapahtuu joko muuttamalla merkkijonona oleva IOR olioviitteksi, etsimällä olion (tunnetulla) nimellä nimipalvelusta, tai olion “ominaisuuksien” avulla Trader-palvelusta. Mikä tahansa muukin keino on sallittu, kunhan viite vain jostain saadaan.

Saatu viite joudutaan yleensä muuntamaan tarvittua olioluokkaa vastaavan tyyppiseksi. Olion IDL-määrittelystä on yleensä generoitunut apuluokka – esimerkiksi luokalle Kissa sen nimi voisi olla KissaHelper – josta löytyy operaation narrow() tyyppimuunnoksen suorittamiseksi.
Tämän jälkeen olio käyttäytyy kuin minkä tahansa “tavallinen” olio.

Esimerkkiohjelma

Esimerkkiohjelmana on toteutettu yksinkertaistettu version kaikille tutusta (?) autolaskurisovelluksesta.

Kääntäminen Inprisen Visibroker for Java ja JDK 1.2:lla:

1. Generoidaan java-rajapinnat ja –apuluokat IDL:stä
   `idl2java -idl_strict -no_comments -no_bind -strict Counter.idl`
2. Käännetään oliototeutus
   `vbjc Server.java`
3. Käännetään asiakasohjelma
   `vbjc Client.java`
4. Käynnistetään oliototeutus
   `vbj Server`
5. Käynnistetään asiakasohjelma
   `vbj Client`

Counter.idl:

```
module gko {
      interface Counter {
            readonly attribute long value;
            readonly attribute string name;
            void add();
            void reset();
      };
};
```

Client.java:

```
import java.io.*;

public class Client {
      private static void printValue(gko.Counter counter) {
            System.out.println(
                  "Counter "+counter.name()
                  +" has the value "+counter.value()+".");
      }

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

            LineNumberReader input=
                  new LineNumberReader(
                        new FileReader("carcounter.ior"));
            org.omg.CORBA.Object obj=
                  orb.string_to_object(input.readLine());
            gko.Counter counter=gko.CounterHelper.narrow(obj);

            printValue(counter);
            System.out.println("Add ten cars...");
            for (int i=0; i<10; ++i) counter.add();
            printValue(counter);

            System.out.println("Reset counter...");
            counter.reset();
            printValue(counter);

            System.out.println(
                  "Add five cars (the paranoid check ;) ...");
            for (int i=0; i<5; ++i) counter.add();
            printValue(counter);
      } catch (java.io.IOException ioe) {
            ioe.printStackTrace();
      }
      }
}
```

CounterImpl.java:

```
import gko.*;

public class CounterImpl extends CounterPOA {
      public int value() {
            System.out.println(name+": value is "+value);
            return value;
      }
      public String name() {
            return name;
      }
      public void add() {
            value++;
            System.out.println(name+": new value is "+value);
      }
      public void reset() {
            value=0;
            System.out.println(name+": value reset to zero");
      }
      public CounterImpl(String name) {
            this(name, 0);
      }
      public CounterImpl(String name, int value) {
            this.name=name;
            this.value=value;
            System.out.println(
                  "Created new counter "+name
                  +" with value "+value+".");
      }
      private int value;
      private String name;
}
```

Server.java:

```
import java.io.*;
import org.omg.PortableServer.*;
import gko.*;

public class Server {
      public static void main(String[] args) {
      try {
            System.out.println("Obtain ORB");
            org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init(args, null);

            System.out.println("Resolve RootPOA");
            POA rootPOA=POAHelper.narrow(
                        orb.resolve_initial_references("RootPOA"));

            System.out.println("Create own POA");
            org.omg.CORBA.Policy[] policies={
                  rootPOA.create_lifespan_policy(
                        LifespanPolicyValue.PERSISTENT)
            };
            POA myPOA=rootPOA.create_POA(
                        "CounterPOA",
                        rootPOA.the_POAManager(),
                        policies);

            System.out.println("Instantiate object");
            CounterImpl car_counter=new CounterImpl("Cars");
            byte[] counter_ID="CarCounter".getBytes();
            myPOA.activate_object_with_id(counter_ID, car_counter);
            rootPOA.the_POAManager().activate();

            System.out.println("Get object reference");
            org.omg.CORBA.Object obj=
                  myPOA.servant_to_reference(car_counter);

            System.out.println("Write ior:");
            String ior=orb.object_to_string(obj);
            FileWriter output=new FileWriter("carcounter.ior");
            output.write(ior);
            output.close();
            System.out.println(ior);

            System.out.println("Run...");
            orb.run();
      } catch (Exception e) {
            e.printStackTrace();
      }
      }
}
```

Viitteitä

- OMG http://www.omg.org/
- CORBA http://www.omg.org/technology/documents/formal/index.htm

Kaikki mahdollinen CORBA:sta “tiiviissä” paketissa.

Kaupallisia ORB-toteutuksia

- Inprise http://www.inprise.com/
- Visibroker http://www.inprise.com/visibroker/

Muitakin löytyy, tuossa yksi käytössä hyväksi havaittu.

Vapaita ORB-toteutuksia

- MICO http://www.mico.org/
- JacORB http://jacorb.inf.fu-berlin.de/
- omniORB http://omniorb.sourceforge.net/

Kuten kaupallisia, näitäkin löytyy enemmän kuin tarpeeksi. Yllä pari hyväksi havaittua, muita ovat esimerkiksi ACE+TAO, Orbit, FnORB.

Katso myös

- GNOME http://www.gnome.org/

Gnome-projekti, tuttu Linuxin käyttäjille, käyttää CORBAa. Linuxille on myös useita muita ORB-toteutuksia.