JAVA RMI

Rinnakkaisohjelmoinnin projekti 1 osa C

Tekijät:

Taru Itäpelto-Hu

Jaakko Nissi

Mikko Ikävalko

Table of Contents

1. Yleistä

Hajautettujen ohjelmien tulee voida kommunikoida keskenään jotenkin, jotta ne toimisivat oikein. Yleisesti tällaisia kommunikointitapoja ovat esimerkiksi socketit ja RPC:t. Javan vastine RPC:lle on luokkakokoelma, joka mahdollistaa toisella Javan virtuaalikoneella sijaitsevan luokan metodin kutsumisen samaan tapaan kuin samalla virtuaalikoneella sijaitsevien olioidenkin palveluiden kutsumisen. Tämä Javan tarjoama luokkakirjasto, java.RMI, määrittelee, kuinka nämä eri paikoissa sijaitsevat ohjelmat kommunikoivat, joten ohjelmoija säästyy socketien ohjelmoinnilta. Tämä mahdollistaa Javan idean laajentamisen. Ohjelmoi kerran, käytä missä vain muuttuu muotoon: ohjelmoi kerran, käytä kaikkialla.

Java RMI -luokan tarjoamien palveluiden voima piilee siinä, että metodien lisäksi voidaan lähettää luokkamäärityksiä ja käyttäytymismalleja. RMI lähettää olioita omassa muodossaan, joten olioiden käyttäytyminen ei muutu lähetyksen jälkeen toisella JVM:llä. Tämä mahdollistaa uusien oliotyyppien ja käyttäytymisten esittelyn toisessa JVM:ssä ja siten laajentaa sovelluksen käyttömahdollisuuksia.

RMI sisältää kaksi erillistä ohjelmaa, palvelimen ja asiakkaan.
Palvelin tyypillisesti luo joitakin etäolioita, mahdollistaa näiden olioiden viitteiden saatavuuden ja odottaa asiakkaan herättävän näiden olioiden metodeita. Tyypillisesti asiakas hankkii etäviitteen yhteen tai useampaan etäolioon palvelimella ja sitten kutsuu niiden metodeja. RMI tarjoaa asiakkaan ja palvelimen välisen kommunikoinnin ja tietojen välittämisen niiden välillä.

Hajautettujen sovellusten täytyy paikallistaa etäoliot. Paikallisten sovellusten on siis hankittava etäolioiden viitteet jollakin tavalla ja tähän tarkoitukseen sovelluksilla on erilaisia mekanismeja. Sovellus voi esimerkiksi rekisteröidä etäolionsa RMI:n yksinkertaiseen nimiavaruuteen eli RMI rekisteriin tai sovellus voi lähettää ja vastaanottaa etäolioviitteitä osana etäkutsuja.
Hajautettujen sovellusten täytyy myös pystyä kommunikoimaan etäolioiden kanssa. RMI huolehtii kommunikoinnin yksityiskohdista. Ideana RMI:n kehittämiselle olikin, että ohjelmoijalle etämetodien käyttö olisi yhtä helppoa kuin paikallistenkin metodien kutsuminen ja ohjelmoijalle kommunikaatio näyttääkin samalta kuin normaalit paikalliset metodikutsut.

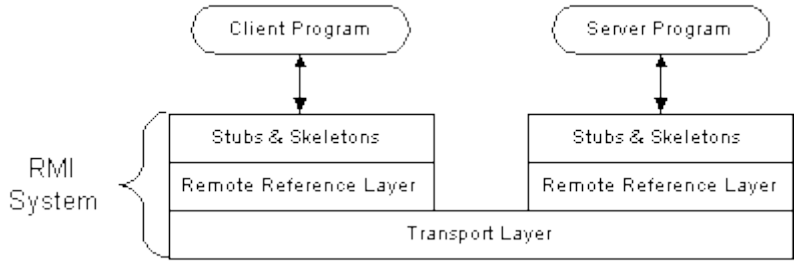
2. Arkkitehtuuri

Kuten muutkin Java-sovellukset, hajautetut sovelluksetkin, jotka on tehty käyttäen Java RMI:tä, koostuvat rajapinnoista ja luokista. Rajapinnat esittelevät luokat ja luokkien palvelut. Luokat toteuttavat nämä esitellyt palvelut. Hajautetussa sovelluksessa toteutukset voivat sijaita joissain JVM:issä, mutta eivät välttämättä toisissa.

Olioita, joilla on metodeja, joita voidaan kutsua toisista JVM:ista, kutsutaan etäolioiksi. Etäolio toteuttaa etärajapinnan (remote interface), joka laajentaa java.rmi.remote-rajapintaa ja jokainen rajapinnan metodi määrittelee poikkeuksen (java.rmi.Exception).

RMIn toteuttavat luokat voivat sallia rinnakkaisuuden ja säikeiden käytön. RMI ei kuitenkaan takaa säikeiden oikeaa toimintaa. Koska samaan etäolioon voi kohdistua rinnakkaisia etämetodikutsuja, on etäolion toteutuksessa varmistettava, että se on säieturvallinen.

2.1. Java RMI kerrosarkkitehtuuri



Javan RMI on toteutettu kolmena kerroksena, joita ovat Stub ja Skeleton kerros, etäviittauskerros ja kuljetuskerros.
Java2 SDK:ta käyttävissä sovelluksissa skeletonia ei tarvitse enää toteuttaa. Lisäksi arkkitehtuuriin voidaan laskea aiemmin mainittu sovelluskerros. Sovelluskerroksessa sijaitsevat asiakas ja palvelinohjelmat. Asiakkaalla on käytössään rajapinta, palvelimella rajapinta sekä toteuttavat luokat. Sovelluskerroksen alapuolella sijaitsee RMI.

2.1.1. Tynkä (stub)

Tynkäkerros on rajapinta sovelluskerroksen ja RMI järjestelmän välillä. Kerroksen tehtävänä on välittää dataa seuraavana alapuolella olevalle etäviittauskerrokselle ns. marshal-streamien (marshal streams, marshaling, järjestäminen) avulla. Näin toimitaan myös toiseen suuntaan: kerros välittää dataa ylemmälle kerrokselle (sovellukselle), unmarshaloi datan. Marshaling tarkoittaa prosessia jolla korkean tason dataa muutetaan muotoon, joka sopii sellaisenaan tiedonsiirtoon. Unmarshaling tarkoittaa prosessia joka jäsentää vastaanotetun tiedon jälleen samaksi korkean tason dataksi. RMI:ssä marshalling mekanismi käyttää hyväksi Javan olioiden sarjallistamismekanismia (object serialization), joka mahdollistaa olioiden siirtämisen eri osoiteavaruuteen. (Pastila 2003)

Java RMI kohtelee etäolioita erilailla kuin paikallisia olioita, kun niitä lähetetään JVM:ltä toiselle. Asiakkaalle lähetetään olion kopion sijasta tynkä, joka toimii olion paikallisena edustajana ja toimii itse asiassa etäviitteenä asiakkaalle. Asiakas kutsuu metodia paikallisessa tyngässä ja tynkä huolehtii metodin kutsumisesta etäoliossa. Tynkä toteuttaa etäoliolle samat etärajapinnat kuin etäolio toteuttaa. Tämä ominaisuus mahdollistaa minkä tahansa etäolion toteuttaman rajapinnan kutsumisen tyngän avulla. Kuitenkin, vain ne metodit, jotka on määritelty etärajapinnassa ovat kutsuttavissa vastaanottavasta JVM:stä.

2.1.2. Etäviittauskerros

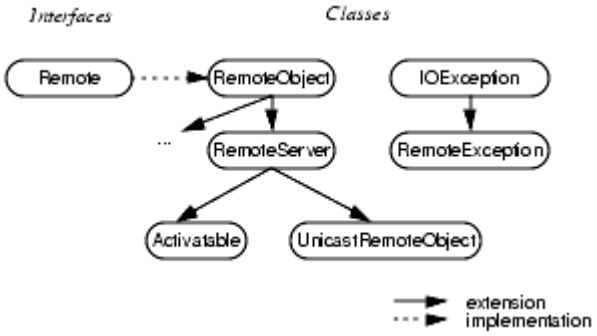
Etäviittauskerros (remote reference layer) on vastuussa erityisen etäviittaus protokollan toteutuksesta. Tämä protokolla on riippumaton asiakkaan stubeista ja palvelimen skeletoneista. Etäviittauskerros käyttää kuljetuskerroksen palveluita.

2.1.3. Kuljetuskerros

Kuljetuskerros luo varsinaisen tietoliikenneyhteyden kahden virtuaalikoneen välille. Kaikki yhteydet ovat stream-perustaisia TCP/IP yhteyksiä. Vaikka nämä virtuaalikoneet sijaitsisivat fyysisesti samalla tietokoneella, ovat ne
yhteydessä kyseisen tietokoneen TCP/IP -protokollapinon kautta.

3. Java RMI API

Rajapinnat ja luokat määrittelevät RMI-systeemin toiminnan. Oheinen kuva esittää, kuinka keskeisimmät RMI-luokan rajapinnat ja luokat liittyvät toisiinsa.



3.1. Remote

RMI:ssä remote-rajapinta esittelee metodit, joita voidaan kutsua toisita JVM:istä. Etärajapinnan tulee ainakin laajentaa, joko suoraan tai epäsuorasti java.rmi.remote-rajapintaa.
Jokaisen etärajapinnan tai sen ylirajapinnan metodin tulee sisältää java.rmi.Exception tai sen yliluokan poikkeusmäärittely. Etämetodimäärittelyssä, jos metodin parametrina tai paluuarvona on olio, on tämä olio määriteltävä rajapintana, ei toteuttavana luokkana.

Remote-rajapinta ei määrittele mitään metodeja.

Etärajapinnan on laajennettava ainakin java.rmi.Remote -rajapintaa. Etärajapinta voi myös laajentaa paikallista rajapintaa, jos kaikki tämän toisen rajapinnan metodit täyttävät etämetodien määrittelyjen vaatimukset.

Esimerkiksi etärajapinta Beta laajentaa sekä paikallista Alpha-rajapintaa sekä java.rmi.Remote-rajapintaa:

Alpha

```
public interface Alpha {
        public final String okay = "constants are okay too";
        public Object foo(Object obj)
                throws java.rmi.RemoteException;
        public void bar() throws java.io.IOException;
        public int baz() throws java.lang.Exception;
}
```

```
public interface Beta extends Alpha, java.rmi.Remote {
        public void ping() throws java.rmi.RemoteException;
}
```

(http://java.sun.com/javase/6/docs/platform/rmi/spec/rmi-objmodel5.html)

3.2. RemoteException

RemoteException -luokka on RMI:n etäkutsujen suoritusaikaisten poikkeusten yliluokka. Jokaisen etämetodin on määriteltävä java.rmi.RemoteException tai joku sen yliluokista esim. java.io.IOException omassa throws-lauseessaan.
Poikkeus heitetään, jos etämetodikutsu epäonnistuu jostakin syystä. Näitä syitä voivat olla esimerkiksi yhteysvirhe, parametrin tai paluuarvon marshalloinnin epäonnistuminen tai protokollavirheet.

3.3. RemoteObject -luokka

RemoteObject-luokka ja sen aliluokat java.rmi.server.RemoteServer, java.rmi.server.UnicastRemoteObject ja java.rmi.activation.Activatable tarjoavat RMI:n palvelimen toiminnot.

4. RMI koodin keinoin.

Yksi javan merkittävimmistä ominaisuuksista on mahdollisuus ladata Java-koodia mistä tahansa URL:stä ja suorittaa sitä paikallisessa VM:ssä. Tämä ominaisuus on ollut käytössä jo kauan selaimissa, joissa ajettu VM voi ladata verkon yli mitä tahansa luokan java.applet.Applet aliluokkia ja suorittaa ne. Myös java RMI hyödyntää tätä ominaisuutta ja mahdollistaa luokkien lataamisen ja suorittamisen missä tahansa VM:ssä.

Yksi dynaamisen koodin latauksen keskeisistä RMI:n ominaisuuksista on mahdollisuus ladata myös luokkamäärittelyt, jos luokkaa ei ole määritelty vastaanottajan JVM:ssä. Kaikki oliotyypit ja niiden käyttäytyminen, mitkä ovat aiemmin olleet saatavilla vain paikallisesti yhdeltä JVM:ltä, voidaan lähettää toiselle, mahdollisesti toisella koneella sijaitsevalle JVM:lle.

Seuraavaksi rakennetaan esimerkki, jossa asiakasohjelma lataa dynaamisesti rmirekisteristä Laskin luokan, jolla se suorittaa muutamia triviaaleja laskutoimituksia. Tämän yksinkertaisen esimerkin pohjalta voidaan kuitenkin kirjoittaa mitä monimutkaisempia etämetodeihin nojaavia ohjelmia lisäämällä luokkiin ominaisuuksia.

4.1. Etäkutsuttavat metodit sisältävä rajapinta.

Laskin.java

```
/**
 * Laskin-rajapinta, joka toimii niin palvelimen, kuin
 * asiakkaankin rajapintana rmirekisteristä saatavalle
 * Laskintoiminnot tarjoavalle luokalle.
 * Laajentaa rajapintaa java.rmi.Remote, joka on yleinen
 * ylirajapinta rajapinnoille, joiden metodeja voidaan
 * suorittaa ei-paikalliselta virtuaalikoneelta.
 * Luokan jokaisen metodin tulee heittää java.rmi.RemoteException,
 * joka on yleinen yliluokka useille etäyhteyteen liittyville
 * poikkeuksille.
```

```
 */

public interface Laskin
  extends java.rmi.Remote {
  //Yhteenlasku
  public long summaa(long a, long b)
    throws java.rmi.RemoteException;

  //Vähennyslasku
  public long vähennä(long a, long b)
    throws java.rmi.RemoteException;

  //Kertolasku
  public long kerro(long a, long b)
    throws java.rmi.RemoteException;

  //Jakolasku
  public long jaa(long a, long b)
    throws java.rmi.RemoteException;
}
```

4.2. Tynkärajapinnan toteuttava luokka.

LaskinImpl.java

```
/**
 * LaskinImpl-luokka toteuttaa Laskin-rajapinnan,
 * eli suorittaa itse laskutoimitukset pyydettäessä.
 * Laajentaa luokkaa java.rmi.server.UnicastRemoteObject,
 * jonka avulla luokka linkittyy javan rmi järjestelmään.
 */
public class LaskinImpl
  extends java.rmi.server.UnicastRemoteObject
  implements Laskin {

  /*
   * Konstruktori täytyy määritellä java.rmi.RemoteExceptionin
   * heittäväksi, vaatimus jonka UnicastRemoteObject asettaa.
   */
  public LaskinImpl()
    throws java.rmi.RemoteException {
    /* Yliluokan konstruktorin kutsuminen suorittaa
     * UnicastRemoteObjectin koodin joka hoitaa linkittämisen
     * rmi järjestelmään, sekä etäolion initialisoinnin.
     */
    super();
  }

  /* Laskutoimitukset suorittavat metodit tulee
   * määrittää palauttamaan java.rmi.RemoteException, joka
   * on yleinen yliluokka useille eri etäyhteyteen liittyville
   * poikkeuksille.
   */
  //Yhteenlasku
  public long summaa(long a, long b)
    throws java.rmi.RemoteException {
    return a + b;
  }

  //Vähennyslasku
```

```
  public long vähennä(long a, long b)
    throws java.rmi.RemoteException {
    return a - b;
  }

  //Kertolasku
  public long kerro(long a, long b)
    throws java.rmi.RemoteException {
    return a * b;
  }

  //Jakolasku
  public long jaa(long a, long b)
    throws java.rmi.RemoteException {
    return a / b;
  }
}
```

☐ **Ilman perintää**

LaskinImpl-luokan ei välttämättä tarvitse laajentaa UnicastRemoteObject-luokkaa, vaan luokan rekisteröinnin rmi-järjestelmään voi hoitaa suorittamalla metodin:
`UnicastRemoteObject.exportObject(this)`

4.3. RMI-palvelimen kirjoittaminen.

LaskinPalvelin.java

```
import java.rmi.Naming;

/**
 * LaskinPalvelin rekisteröi Laskin-rajapinnan toteuttavan
 * LaskinImpl luokan olion rmirekisteriin nimellä LaskinPalvelu.
 */
public class LaskinPalvelin {

  /*
   * LaskinPalvelimen konstruktorissa rekistöidään Laskimen toteuttava
   * rmi-palvelu.
   */
  public LaskinPalvelin() {
    try {
      // Luodaan uusi laskimen toteuttava olio sidottavaksi palveluun.
      Laskin c = new LaskinImpl();
      /* Rekisteröidään Laskin-rajapinnan toteuttava LaskinImpl-luokan
       * olio samalla koneella ajettavaan rmirekisteriin nimellä
       * LaskinPalvelu. Jos nimelle on jo rekisteröity palvelu,
       * se korvataan tällä uudella oliolla.
       */
      Naming.rebind("rmi://localhost:1099/LaskinPalvelu", c);
    }
    /* Jos annettu rmirekisterin osoite on epäkelpo url,
     * Naming.rebind heittää MalformedURLExceptionin.
     */
    catch (MalformedURLException mue) {
      System.out.println("MalformedURLException");
      System.out.println(mue);
    }
```

```
    /* Jos rmirekisteriin ei saada yhteyttä, heittää
     * Naming.rebind RemoteException.
     */
    catch (RemoteException re) {
      System.out.println("RemoteException");
      System.out.println(re);
    }
    /* Jos Naming.rebind-metodia kutsutaan ei-paikalliselta isännältä,
     * se heittää AccessExceptionin.
     */
    catch (AccessException ae) {
    System.out.println("AccessException");
    System.out.println(ae);
    }
  }

  /*
   * main-metodi luo vain uuden LaskinPalvelimen, jonka
   * konstruktorissa kaikki työ tehdään.
   */
  public static void main(String args[]) {
    new LaskinPalvelin();
  }
}
```

4.4. RMI-asiakkaan kirjoittaminen.

LaskinAsiakas.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

/**
 * LaskinAsiakas pyytää rmiregistryltä Laskimen tynkäluokkaa ja
 * laskee saamallaan luokalla pari triviaalia laskutoimitusta.
 */
public class LaskinAsiakas {
  public static void main(String[] args) {
    try {
      /* Pyydetään tynkäluokkaa joka on rekisteröity nimellä
       * LaskinPalvelu rmirekisteriltä jota ajetaan samalla
       * koneella kuin asiakasta ja castataan(suomennos?)
       * se luokan Laskin olioksi.
       */
      Laskin c = (Laskin)Naming.lookup("rmi://localhost/LaskinPalvelu");

      //Triviaalit laskutoimitukset.
      System.out.println( c.vähennä(4, 3) );
      System.out.println( c.summaa(4, 5) );
      System.out.println( c.kerro(3, 6) );
      System.out.println( c.jaa(9, 3) );
    }
    /* Jos annettu RMIrekisterin osoite on epäkelpo url,
     * Naming.lookup heittää MalformedURLExceptionin.
     */
    catch (MalformedURLException murle) {
      System.out.println();
      System.out.println("MalformedURLException");
```

```
      System.out.println(murle);
    }
    /* Jos rmirekisteriin ei saada yhteyttä, heittää
     * Naming.lookup RemoteExceptionin.
     */
    catch (RemoteException re) {
      System.out.println();
      System.out.println("RemoteException");
      System.out.println(re);
    }
    /* Jos rmirekisteriin ei ole rekisteröity nimeä, jota
     * pyydetään Naming.lookupilla, Naming.lookup heittää
     * NotBoundExceptionin.
     */
    catch (NotBoundException nbe) {
      System.out.println();
      System.out.println("NotBoundException");
      System.out.println(nbe);
    }
    /* ArithmeticException tulee kopata Laskimen suorittamien
     * laskutoimitusten takia eikä liity itse rmin toteutukseen.
     */
    catch (java.lang.ArithmeticException ae) {
      System.out.println();
      System.out.println("java.lang.ArithmeticException");
      System.out.println(ae);
    }
  }
}
```

4.5. Esimerkin kääntäminen

Aluksi käännetään palvelun rajapinta ja itse palvelun implementaatio;

```
>javac Laskin.java
>javac LaskinImpl.java
```

Seuraavaksi generoidaan etäkäyttöön tarvittavat stub- ja skeleton luokat;

```
>rmic LaskinImpl
```

☐ **rmic vivut**

java 2:n rmic:tä ajettaessa voidaan vivuilla valita mille javan versiolle tynkät/luurangot generoidaan.

```
>rmic -v1.1 LaskinImpl # Generoi tyngät ja lurkit java1:lle.
>rmic -v1.2 LaskinImpl # Generoi tyngät java2:lle.
>rmic -vcompat LaskinImpl # Generoi tyngät ja lurkit molemmille
javaversioille.
```

Lopuksi käännetään palvelin- ja asiakasohjelma;

```
>javac LaskinPalvelin.java
>javac LaskinAsiakas.java
```

Ja sitä myöten esimerkki on valmis ajettavaksi.

5. Ajaminen

1.Käynnistä rmiregistry

Windows:
start rmiregistry
Unix:
rmiregistry &

2.Käännä lähdekoodi
Käännä lähdekoodit seuraavilla komennoilla:
javac *tiedosto*

3.Käynnistä serveri
Käynnistä serveri luokat sisältävässä hakemistossa kirjoittamalla:
java *serveri*

Java-versiosta 5.0 lähtien RMI stubseja ei ole enää tarvinnut kääntää erikseen rmic:llä.

4.Käynnistä asiakas
Asiakas voidaan ajaa paikallisesti, tai joltain muulta koneelta.
Jos ajat asiakkaan paikallisesti käytä seuraavaa komentoa:
java *asiakas* localhost
Jos taas ajat asiakkaan muualta, niin käytä seuraavaa komentoa (korvaa *hostname* oikealla hostnamella)
java *asiakas* *hostname*

6. Yhteenveto

Java RMI tarjoaa suhteellisen helposti käytettävät työkalut hajautetun ohjelman toteuttamiseen. Kuten koodiesimerkeistä huomaamme, yksinkertaisen toteutuksen tekeminen ei ole kovinkaan monimutkaista. RMIn käyttöönoton tekee helpoksi myös se, että se sisältyy Javan kehityspakettiin, joten mitään erillisiä lisäosia ei tarvita. Tämä tarkoittaa luonnollisestsi myös sitä, että RMItä voidaan käyttää ainoastaan Javan kanssa.

Java RMI on muuttunut melko radikaalisti Javan eri versioiden aikana. Jopa koodin kääntäminen tehdään uudemmissa Java-versioissa eri tavalla. Tämä tekee hyvien ohjeiden etsimisen internetistä hankalaksi, ja vanhoille versioille tehtyjä ohjeita löytyy paljon. Harvassa ohjeessa on minkäänlaista mainintaa, mille versiolle ohje on laadittu. Varmin tapa löytää uusinta tietoa on hakea sitä Sunin sivuilta, josta löytyy varmasti oikeat syntaksit.

7. Erot luennolla esitettyyn materiaaliin

Luennolla käsiteltiin Java RMI:n aiempaa versiota (J2SE 5.0), kun tässä olemme käsitelleet Java RMI:n uudempaa versiota (Java SE 6). Uuden version etuna on se, että skeletonit ovat tarpeettomia. Uusi versio käyttää hyväkseen Activable ominaisuutta. Ja näin Java RMI:tä on edelleen pyritty tekemään käyttäjäystävällisemmäksi.

8. Käyttölupa

Lähteet

http://java.sun.com/developer/onlineTraining/rmi/RMI.html#IntroRMI
Seppo Pastila, Java RMI, Helsingin Yliopisto Tietojenkäsittelytieteen laitos, Väliohjelmistot
http://www.cs.helsinki.fi/u/summanen/opetus/2003/middleware/esseet/pastila.doc
http://java.sun.com/docs/books/tutorial/rmi/TOC.html
Timo Aaltonen, Hajautettujen järjestelmien perusteet, Java RMI
http://www.cs.tut.fi/~hajap/luennot/12JavaRMI.pdf
http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html

http://en.wikipedia.org/wiki/Java_remote_method_invocation