SATAKUNNAN AMMATTIKORKEAKOULU

Minna Seppi

TIEDOSTOPALVELU RMI-TEKNIIKALLA

TIETOJENKÄSITTELYN KOULUTUSOHJELMA

2006

TIEDOSTOPALVELU RMI-TEKNIIKALLA
Minna Seppi
Satakunnan Ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Joulukuu 2006
Juha Stenfors
Asiasanat:
UDK: 004.45, 004.6

Opinnäytetyössä käsiteltiin tiedostopalvelun tekemistä RMI-tekniikalla. Lisäksi työssä toteutettiin pienimuotoinen esimerkkisovellus, jossa tiedostopalvelun toimintaa esiteltiin käytännössä.. Esimerkkisovelluksen tietokannanhallintajärjestelmäksi valittiin MySQL AB:n kehittämä MySQL-tuote (versio 4.0.17).

Työtä tehtäessä huomattiin, ettei tiedostopalvelun toteuttaminen sinänsä ollut vaikeaa. Kyseessä oli yksinkertainen ”lisäpalikka” jo olemassa olevaan järjestelmään. Suurin osa ajasta kului kirjallisen dokumentin tuottamiseen, sillä tarjolla olevasta materiaalista oli välillä vaikeaa löytää tarpeellinen tieto.

Työssä kuvattiin ensin tiedostopalvelun tekninen toteutus ja sitä hyödyntäen lisättiin aiemmin tehtyyn järjestelmään tiedostopalvelu. Esimerkkisovellusta käsittelevässä luvussa kerrottiin tiedostopalvelun toiminnasta.

FILE SERVICE IN RMI TECHNIQUE
Minna Seppi
Satakunta Polytechnic
Degree Programme in Business Information Systems
December 2006
Juha Stenfors
Keywords:
UDC: 004.45, 004.6

This thesis dealt with creating a file service in RMI technique. The thesis also carried out a simple example application that shows how the file service works. For the example's database management system was chosen MySQL AB's MySQL-product (version 4.0.17).

The thesis noticed that creating the file service wasn't actually difficult. It was a simple "additional part" to an existing system. Most of the time was spent on writing the thesis itself because sometimes it was quite difficult to find the necessary information from all of the material available.

The thesis first described the technical implementation of the file service and then by utilising it a file service was added to the existing system. The chapter concerning the example application tells about using the file service.

SISÄLLYS

TERMILUETTELO

abstrakti luokka	Luokka, josta ei suoraan voi luoda olioita. Luokalle on luotava aliluokka, josta voi luoda olioita.
java.awt	Javan luokkakirjaston pakkaus, josta löytyvät luokat käyttäjärajapintojen ja grafiikoiden tekoon.
java.awt.event	Javan luokkakirjaston pakkaus, josta löytyvät rajapinnat ja luokat AWT-komponenttien aiheuttamien tapahtumien hoitamiseksi.
java.io	Javan luokkakirjaston pakkaus, joka tarjoaa järjestelmän inputin ja outputin tietovirroin, sarjallistamisen sekä tiedostojärjestelmän.
java.net	Javan luokkakirjaston pakkaus, joka tarjoaa luokat verkkosovellusten toteuttamiseen.
java.rmi	Javan luokkakirjaston pakkaus, joka tarjoaa RMI-paketin.
java.rmi.server	Javan luokkakirjaston pakkaus, joka tarjoaa RMI:n palvelinpuolen tukemiseen tarvittavat luokat ja rajapinnat.
java.sql	Javan luokkakirjaston pakkaus, joka sisältää tietokannan käsittelyyn käytettäviä luokkia ja rajapintoja.
java.util	Javan luokkakirjaston pakkaus, joka sisältää mm. tietorakenteiden mallintamiseen tarvittavat luokat ja rajapinnat.
javax.swing	Javan luokkakirjaston pakkaus, joka sisältää lukuisia käyttöliittymäkomponenttiluokkia.

laajentaa (extends)	Luoda toiselle luokalle aliluokka, joka perii yläluokkansa kentät ja metodit.
main-metodi	Java-ohjelman suoritus aloitetaan aina main-nimisestä metodista.
rajapinta	Sisältää metodien määrityksiä, muttei niiden toteutuksia. Rajapinnan toteuttava luokka toteuttaa määritellyt metodit.
Remote, Serializable	Rajapinnat, jotka kaikkien etäolioiden täytyy toteuttaa joko suoraan tai välillisesti. Kumpikaan rajapinta ei sisällä metodeita.
RMI	Remote Method Invocation on Javan hajautettujen olioiden toteutustekniikka. Sen avulla Java-olio voi kutsua toisessa JVM-koneessa (Java Virtual Machine) olevan Java-olion metodia. JVM:t voivat olla samassa tietokoneessa tai eri tietokoneissa. Lisäksi Java-olio voi kutsua toisen Java-olion metodia RMI:n kautta, vaikka oliot ovat samassa JVM:ssä.
rmic-kääntäjä	Javassa on rmic-kääntäjä, joka generoi luokalle stub- ja skeleton-luokat. Ennen kuin voi kääntää rmic-kääntäjällä, pitää etäolio olla käännettynä Javakääntäjällä (javac LuokanNimi.java)
staattinen metodi	Luokkakohtainen metodi, jota kutsutaan käyttämällä luokan nimeä.
stub - skeleton	RMI-järjestelmän stub- ja skeleton -oliopari liittää asiakas-olion ja palvelin-olion yhteen. Stub edustaa palvelin-oliota asiakaskoneessa ja välittää metodikutsut palvelinkoneessa olevan skeleton-olion avulla varsinaiselle palvelin-oliolle.

toteuttaa (implements) Luoda luokka, joka perii rajapinnan metodit.

1 JOHDANTO

Tämän opinnäytetyön aihe on tiedostopalvelun tekeminen RMI-tekniikalla. RMI on lyhenne sanoista Remote Method Invocation eli etämetodikutsu. Sen avulla Java-olio voi kutsua toisessa JVM-koneessa (Java Virtual Machine) olevan Java-olion metodia. Esittelen esimerkkisovelluksen avulla RMI:n toimintaa käytännössä. Esimerkkisovelluksen palvelinpuolen luokkakaavio on esitetty liitteessä 1, asiakaspuolen luokkakaavio liitteessä 2.

Olen jo opintojen aikaisemmassa vaiheessa tehnyt pienimuotoisen taloushallintojärjestelmän yksityiselle toiminimelle ja liitän siihen ”lisäpalikkana” tiedostopalvelun. Käyttäjä voi hakea samassa tietokoneessa tai eri tietokoneissa olevilta JVM:ltä erilaisia tiedostoja, muokata niitä ja halutessaan tallentaa muutokset ennen kuin vapauttaa tiedostot.

Tällä hetkellä kolme suosituinta tekniikkaa etämetodikutsujen toteuttamiseen ovat Javasoftin Java/RMI (Java/Remote Method Invocation), Object Management Groupin CORBA (Common Object Request Broker Architecture) ja Microsoftin DCOM (Distributed Component Object Model) (my.execpc.com, [verkkodokumentti]). Näistä kolmesta RMI on puhtaasti Java-perustainen, CORBA ja DCOM tukevat useampia ohjelmointikieliä. Miksi siis valitsin juuri RMI-tekniikan? Suurin syy on se, ettei opinnäytetyön tekemiseen ole kovinkaan paljon aikaa ja RMI on kohtalaisen helppoa oppia. RMI:n huomattava etu esimerkiksi CORBAan verrattuna on se, että se sallii koodin ja datan siirron koneelta toiselle kun yleensä etäkutsumekanismit vaativat parametrien olevan joko alkeistyyppiä tai alkeistyypeistä koottuja rakenteita (www.javacoffeebreak.com, [verkkodokumentti]). Esimerkkisovellus on yksinkertainen ja RMI:n kaltainen kevyt tekniikka riittää hyvin. Jos toteuttaisin tiedostonsiirron isoon yritykseen, valintani ei ehkä silti olisi RMI lähinnä sen kieliriippuvaisuuden takia.

RMI:n haittapuolina voidaan mainita esimerkiksi turvallisuusriskit koodin etäsuorituksessa aikana sekä turvallisuusmääräysten aiheuttamat rajoitukset toiminnallisuudessa (www.javacoffeebreak.com, [verkkodokumentti].) Lisäksi ongelmia aiheutuu kieliriippuvuudesta koska tällä hetkellä RMI toimii vain Javan kanssa. RMI:tä kuitenkin kehitetään koko ajan, jotta siitä tulisi kilpailukykyisempi. Tekniikka nimeltä RMI-IIOP yhdistää CORBAN vahvuudet RMI:n joustavuuteen (www-128.ibm.com 12.3.2002, [verkkodokumentti]). Tämä opinnäytetyö ei kuitenkaan ota kantaa RMI-IIOP:iin.

2 TIEDOSTOPALVELUN TEKNINEN TOTETUS

2.1 RMI:stä yleisesti

RMI (Remote Method Invocation) on Javan hajautettujen olioiden toteutustekniikka. RMI:n avulla Java-olio voi kutsua toisessa JVM-koneessa (Java Virtual Machine) olevan Java-olion metodia. JVM:t voivat olla samassa tietokoneessa tai eri tietokoneissa. Lisäksi Java-olio voi kutsua toisen Java-olion metodia RMI:n kautta, vaikka oliot ovat samassa JVM:ssä (myy.helia.fi, [verkkodokumentti].)

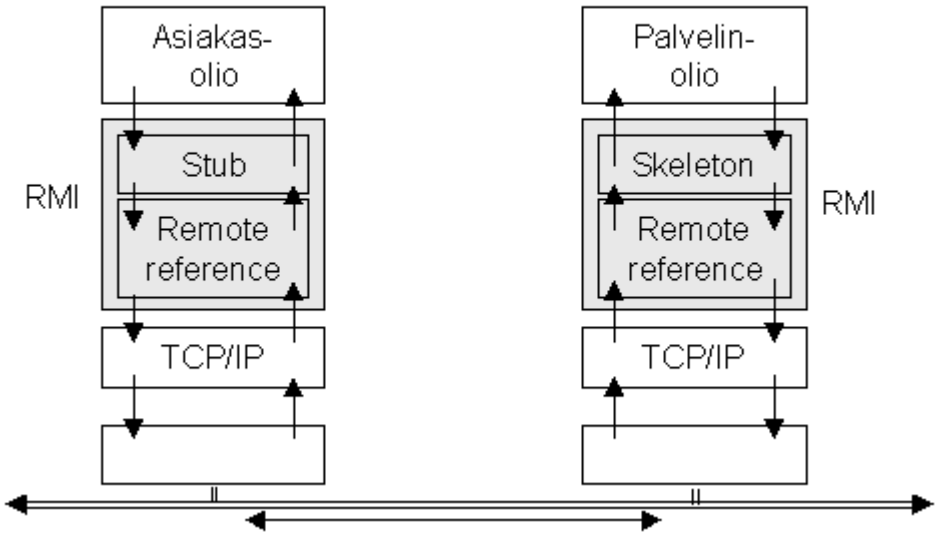
RMI-sovellukset koostuvat usein kahdesta erillisestä ohjelmasta, palvelimesta ja asiakkaasta. Tyypillinen palvelinsovellus luo ensin useita etäolioita. Sen jälkeen se tekee viittaukset näihin etäolioihin mahdollisiksi ja odottaa asiakkaiden kutsuvan näiden etäolioiden metodeja. Tyypillinen asiakassovellus saa etäviittauksen yhteen tai useampaan palvelimen etäolioon ja kutsuu sitten niiden metodeja. RMI tarjoaa mekanismin, jonka avulla palvelin ja asiakas voivat kommunikoida ja välittää tietoa toisilleen.

Yleensä etäkutsuja suorittavat järjestelmät sallivat vain yksinkertaisten tietotyyppien tai määriteltyjen rakennelmien välittämisen metodien välillä, mutta RMI sallii käytettävän mitä tahansa Javan oliotyyppiä - vaikka asiakas tai palvelin ei olisikaan aiemmin törmännyt siihen (www.javacoffeebreak.com [verkkodokumentti]). RMI sallii RMI-kutsuissa välitettävien parametrien, paluuarvojen ja poikkeusten olla mitä tahansa sarjallistettuja olioita (näihin kuuluvat alkeistyypit, etäoliot ja rajapintaa java.io.Serializable toteuttavat tavalliset oliot). RMI käyttää objektien sarjallistamismekanismia tiedon välittämiseen yhdeltä virtuaalikoneelta toiselle. Lisäksi se lisää kutsuvirtaan asiaankuuluvan tiedon sijainnista, jotta vastaanottaja voi ladata luokan määritystiedostot (class definition files). RMI-järjestelmä lataa dynaamisesti luokat parametreja tai paluuarvoja varten, jos ne eivät ole saatavilla paikallisesti. Lisäksi RMI sallii sekä asiakkaan että palvelimen ladata dynaamisesti uusia oliotyyppejä tarpeen mukaan

(www.javacoffeebreak.com [verkkodokumentti]).

2.2 RMI:n rakenne

RMI:n rakenne ja sijoittuminen TCP/IP-protokollaan nähden on esitetty kuvassa 1. RMI koostuu kahdesta kerroksesta: stub/skeleton -kerros ja remote reference -kerros. Itse RMI käyttää Internetin TCP/IP-protokollaa. Etäolion metodia kutsuva asiakasolio kommunikoi Stub-olion ja palvelin-olio Skeleton-olion kanssa. RMI:n kerros TCP/IP-protokollaan päin on Remote reference-kerros (myy.helia.fi, [verkkodokumentti].)



Kuva 1. RMI:n rakenne ja sijoittuminen TCP/IP-protokollaan nähden (myy.helia.fi, [verkkodokumentti])

2.2.1 Stub - Skeleton

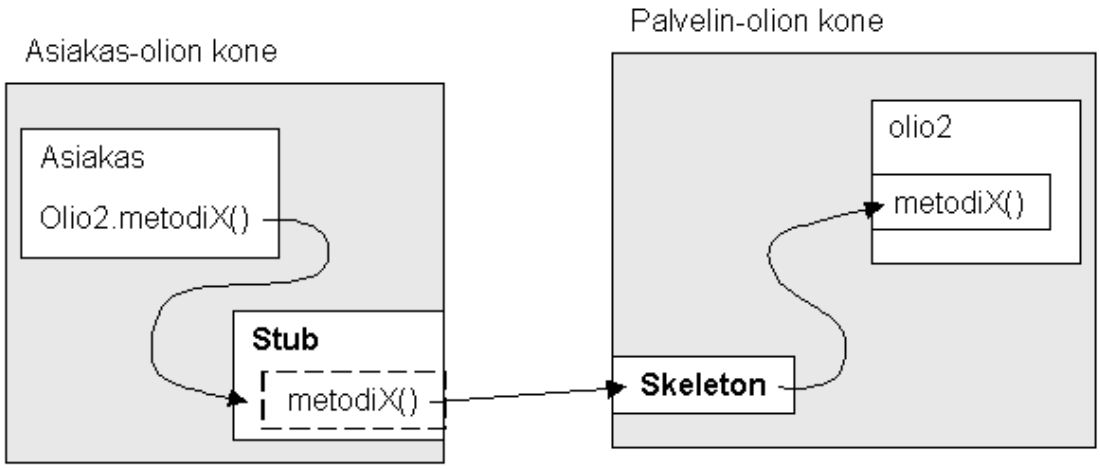
RMI-järjestelmän stub- ja skeleton -oliopari liittää asiakas-olion ja palvelin-olion yhteen. Stub edustaa palvelin-oliota asiakaskoneessa ja välittää metodikutsut palvelinkoneessa olevan skeleton-olion avulla varsinaiselle palvelin-oliolle. Etäolion etäkutsuttavat metodit muodostavat tietyn Interface-rajapinnan, jotka ovat edustettuna myös stub-oliossa (myy.helia.fi, [verkkodokumentti].)

Asiakas-olion kutsuessa etäoliota (kuva 2) kutsu menee etäoliosta asiakaskoneessa olevalle stub-oliolle, joka

- alustaa yhteyden etäkoneeseen ja koodaa lähetettävän metodikutsun tiettyjen sääntöjen mukaan (koodattuun kutsuun tulee etäolion tunniste ja etämetodin nimi),
- hallinnoi (marshals) parametrien välittämisen etämetodille laiteriippumattomalla tavalla,
- lähetettyään metodikutsun skeleton-olion avulla palvelinoliolle jää odottamaan metodin palautetta (joka voi olla joko olio tai perustietotyyppisen muuttujan arvo) ja
- palautteen tullessa välittää sen kutsuvalle metodille.

Kun skeleton vastaanottaa metodikutsun, se

- purkaa stub-olion koodaaman metodikutsun (purkamiseen kuuluu välitettävien parametrien takaisin koodaaminen ja sarjallistamisen purkaminen),
- välittää sen palvelinoliolle ja
- ottaa vastaan metodin palautteen ja hallinnoi (marshals) sen lähettämisen stub-oliolle.

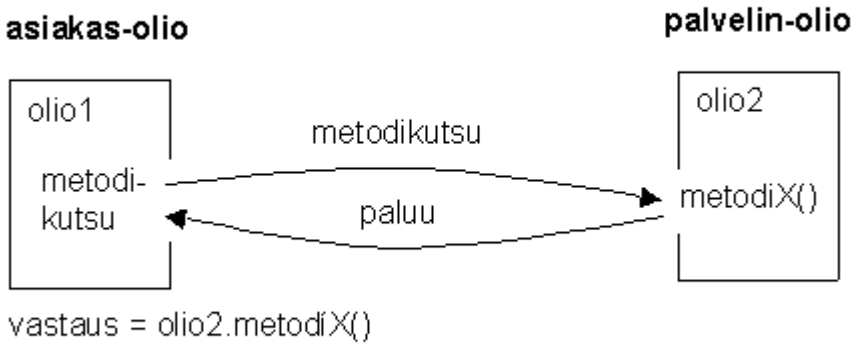


Kuva 2. Etäolion kutsu (myy.helia.fi, [verkkodokumentti])

2.2.2 Etäolion luominen ja etämetodin kutsuminen

RMI:n suunnittelun tavoite oli tehdä RMI:n käyttö ohjelmoijalle läpinäkyväksi. Etäolion metodikutsu on syntaktisesti samanlainen kuin paikallisen olion metodikutsu, mutta etäolion ollessa kyseessä "pinnan alla" tapahtuu paljon lisätoimenpiteitä verrattuna paikalliseen metodikutsuun.

Etäolio (remote object) on Java-olio, jonka yhtä tai useampaa metodia voi kutsua RMI:n avulla. Etäolion tyyppi kuvataan yhdellä tai useammalla etärajapinnalla, jotka määrittelevät etäolion metodit. Etämetodi (remote method) on etäolion metodi, jota voi kutsua RMI:n avulla samassa tietokoneessa tai eri tietokoneissa sijaitsevasta JVM:stä. RMI eli etämetodikutsu on toiminta, jossa etärajapinnan metodeja kutsutaan. RMI-terminologiassa etäolion metodia kutsuvaa oliota kutsutaan asiakasolioksi (client object). Palvelinolioksi (server object) kutsutaan etäoliota, jonka metodia kutsutaan. Jos asiakasolio ja palvelinolio ovat eri JVM-koneissa, puhutaan asiakaskoneesta ja palvelinkoneesta. Kuvassa 3 on esitelty RMI:n asiakas/palvelin-periaate (client/server). Tämä pätee vain yhteen RMI-metodikutsuun ja seuraavassa metodikutsussa roolit voivat olla päinvastoin.



Kuva 3. RMI:n asiakas/palvelin-periaate (myy.helia.fi, [verkkodokumentti])

Etäoliota luotaessa on ensimmäiseksi määriteltävä yksi tai useampi etärajapinta, joka sisältää toisesta JVM:stä kutsuttavissa olevat metodit. Etärajapinnan täytyy vähintään laajentaa rajapintaa java.rmi.Remote (tai jotain muuta etärajapintaa, joka laajentaa rajapintaa java.rmi.Remote). Etärajapinta voi kuitenkin laajentaa ei-etärajapintaa seuraavan ehdon täyttyessä:

- Etärajapinta voi laajentaa myös toista ei-etärajapintaa niin kauan kuin kaikki laajennetun rajapinnan metodit (jos niitä on) täyttävät etämetodin määritelmän vaatimukset (requirements of a remote method declaration), joita ovat:
    - Sovelluskohtaisten poikkeusten lisäksi etämetodimääritelmän tulee throws-osiossaan määrittää poikkeus java.rmi.RemoteException (tai jokin sen yläluokista kuten java.io.IOException tai java.lang.Exception).
    - Sovelluskohtaisten poikkeusten ei tarvitse laajentaa luokkaa java.rmi.RemoteException.

Etämetodin määritelmässä parametriksi tai paluuarvoksi määritelty etäolio täytyy määritellä etärajapinnaksi, ei kyseisen rajapinnan toteutusluokaksi. Koska etäolio periytetään joko luokasta RemoteServer tai luokasta UnicastRemoteObject, sekin toteuttaa rajapinnan Remote. Etämetodeissa voi syntyä poikkus RemoteException. Etäolion konstruktorit voivat synnyttää saman poikkeuksen. Etäolion voi halutessaan rekisteröidä rmiregistry-nimipalveluun.

Palvelinohjelma synnyttää etäolion ja rekisteröi sen rmiregistry-rekisteriin metodilla Naming.rebind(). Asiakas-ohjelma kutsuu etäolion etämetodeja ja sen alussa tulee rmiregistry-nimipalvelusta hakea etäolion stub-olio. Stub toimii etäolion edustajana (proxy) asiakaskoneessa. Siitä löytyy kaikki ne rajapinnat, jotka sen isäntä-luokka toteuttaa. Javassa on rmic-kääntäjä, joka generoi luokalle stub- ja skeleton-luokat. Ennen kuin luokkia voi kääntää rmic-kääntäjällä, pitää etäolio olla käännettynä Javakääntäjällä (javac LuokanNimi.java). rmic-kääntäjä käynnistetään komennolla rmic LuokanNimi. Käännöksessä syntyy LuokanNimi_Stub.class ja LuokanNimi_Skel.class -tiedostot. Java 2 SDK:n versiosta 1.2 lähtien ei skeleton-oliota enää tarvita, joten käännöksen tuloksena syntyy vain stub-luokka.

Kääntämisen jälkeen käynnistetään rmiregistry. RMI registry on yksinkertainen palvelinkoneessa ajettava nimipalvelu, jonka avulla asiakasolio saa viittauksen etäolioon. Lisäksi käynnistetään palvelin- ja asiakas-ohjelmat. Asiakasohjelman voi käynnistää vasta kun tietoturva-asetukset määräävä policy-tiedosto on olemassa. Policy-tiedostoon tutustutaan tarkemmin luvussa 2.5 Turvallisuus.

2.2.3 Metodin parametrien välitys RMI:ssä

Javassa metodin parametrien ja paluuarvon välitys ilman RMI:tä tapahtuu joko by value- tai by reference-periaatteella. by value – periaate tarkoittaa, että välitetään muuttujan kopio. Näin välitetään perustietotyyppisiä muuttujia. by reference – periaatteessa välitetään viittaus olioon eli olion välitys tapahtuu näin.

Myös RMI:ssä perustietotyypin muuttujat välitetään by value -periaatteella. Jos metodin parametri tai paluuarvo on perustietotyyppinen, muuttujan kopio viedään JVM:stä toiselle standardilla, laiteriippumattomalla tavalla.

Etämetodikutsun parametrina välitetty tai sen tuloksena saatu tavallinen olio välitetään kopiona. Tavallisten olioiden kohdalla ei ole mielekästä välittää viittausta olion, koska viittaus on konekohtainen. RMI:ssä olio välitetään siten, että RMI lähettää koko olion JVM:stä toiseen. Ennen kuin etämetodia kutsutaan, tavallisen olion sisältö kopioidaan. Kun tavallinen olio palautetaan etämetodikutsussa, luodaan uusi olio kutsuvalle virtuaalikoneelle. RMI voi välittää sarjallistettavissa olevan olion eli luokka toteuttaa rajapinnan Serializable tai Externalizable.

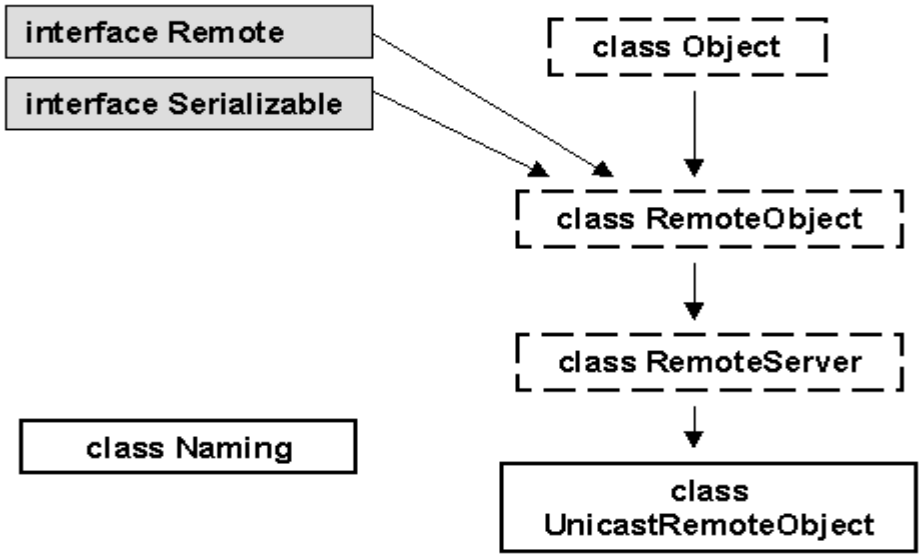
Etämetodin parametrin tai paluuarvon ollessa etäolio, välitetään etäolion stub-olio. Stub-oliossahan on edustettuna etäolion etärajapinnat. Stub-olion saava asiakasohjelma voi sijoittaa Stub-olioviittauksen etärajapintamuuttujaan ja kutsua etäolion etämetodeja.

Jos kaksi viittausta olioon välitetään JVM:stä toiseen parametreina tai paluuarvona yhdessä etämetodikutsussa ja nämä viittaukset viittaavat samaan olioon lähettävässä

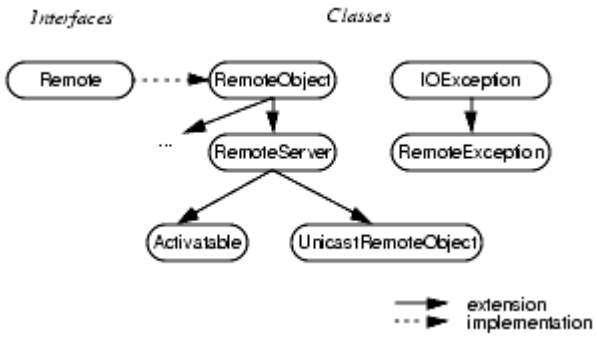
JVM:ssä, ne viittaavat samaan olion kopioon vastaanottavassa JVM:ssä. Yksinkertaisemmin sanottuna: yhdessä etämetodikutsussa RMI-järjestelmä ylläpitää viite-eheyttä kutsun parametreina tai paluuarvona välitettyjen olioiden välillä (java.sun.com [verkkodokumentti].)

2.3 RMI-luokat

RMI-luokat ovat pääasiassa kahdessa pakkauksessa: java.rmi ja java.rmi.server. Tärkeimmät RMI-rajapinnat ja -luokat on esitelty kuvassa 4. Kaikkien etäolioiden täytyy toteuttaa joko suoraan tai välillisesti metodittomat rajapinnat Remote ja Serializable. RMI määrittää abstraktin luokan RemoteObject ja tästä johdetun abstraktin luokan RemoteServer. RemoteObject toteuttaa rajapinnat Remote ja Serializable, RemoteServer toimii yleisenä etäolio-toteutusten kantaluokkana. Siitä on johdettu luokka UnicastRemoteObject. Tämä luokka määrittää etäolion, johon voi viitata palvelinprosessin ollessa käynnissä. Jos olioilla halutaan olevan etäolioiden käyttäytymispiirteitä, ne pitää johtaa luokasta RemoteServert tai oikeammin luokasta UnicastRemoteObject (myy.helia.fi, [verkkodokumentti].) RMI-järjestelmän etäkäyttäytymisestä vastuussa olevat rajapinnat ja luokat löytyvät paketista java.rmi. Kuvassa 5 esitellään useimpien näiden rajapintojen ja luokkien välisiä suhteita.



Kuva 4. Tärkeimmät RMI-rajapinnat ja -luokat (myy.helia.fi, [verkkodokumentti])



Kuva 5. RMI-järjestelmän etäkäyttäytymisen määrittävien rajapintojen ja luokkien suhteita (java.sun.com, [verkkodokumentti])

Etärajapintaa toteuttavien luokkien tulee täyttää muutamia yleisiä sääntöjä. Luokka yleensä laajentaa luokkaa java.rmi.server.UnicastRemoteObject ja perii näin etäkäyttäytymisen luokilta java.rmi.server.RemoteObject ja java.rmi.server.RemoteServer. Tarvittaessa etärajapintaa toteuttava luokka voi kuitenkin laajentaa myös muuta luokkaa kuin java.rmi.server.UnicastRemoteObject. Tällöin toteuttavan luokan tulee itse huolehtia luokasta java.lang.Object periytyvien metodien hashCode, equals, ja toString oikeasta toiminnasta etäolioiden kanssa. Etäolioiden toteutukset voivat laajentaa luokkia java.rmi.server.UnicastRemoteObject ja java.rmi.activation.Activatable, jotka RMI tarjoaa helpottamaan etäolioiden toteutusta (C:\j2sdk142\docs 2003, [verkkodokumentti]). Luokka voi toteuttaa useita etärajapintoja tai se voi myös laajentaa toista etätoteutusluokkaa. Lisäksi luokka voi määritellä metodeja, joita ei ole etärajapinnalla. Näitä metodeja voi käyttää vain paikallisesti, eivätkä ne ole saatavilla etäkäyttöä varten.

Jotta asiakas voi kutsua etäolion metodia, sen on ensin saatava viittaus kyseiseen olioon. Viittaus etäolioon saadaan yleensä metodikutsun parametrina tai paluuarvona. Tätä varten RMI tarjoaa luokan java.rmi.Naming. Sen avulla voidaan tallentaa etäolioiden viittauksia rmiregistry-rekisteriin (remote object registry) ja hakea tallennettujen olioiden viittauksia nimen avulla. Luokka Naming tarjoaa seuraavat URL-pohjaiset (Uniform Resource Locator) metodit:

- bind(String name, Remote obj), joka yhdistää annetun nimen (name) etäolioon obj
- list(String name), joka listaa rmiregistry-rekisteriin tallennetut nimet

- lookup(String name), joka palauttaa etäolion viittauksen (stub-olion) nimen perusteella
- rebind(String name, Remote obj), joka yhdistää annetun nimen (name) uuteen etäolioon
- unbind(String name), joka poistaa rmiregistry-rekisterista nimi-etäolioviitaus – parin (myy.helia.fi, [verkkodokumentti].)

2.4 Remote- ja Serializable-luokat

Java RMI:ssä voidaan käyttää kahdenlaisia luokkia, Remote ja Serializable. Remote-luokan ilmentymiä voidaan käyttää etäältä, Serializable-luokan ilmentymiä voidaan kopioida osoiteavaruudesta toiseen.

Remote-luokan olioon voidaan viitata kahdella tavalla: luontisoiteavaruuden sisällä olio on tavallinen olio ja sitä voidaan käyttää kuin mitä tahansa oliota, muissa osoiteavaruuksissa olioon voidaan viitata olioviitteen eli ”kahvan” (object handle) avulla. Olioon verrattuna kahvan käytössä on rajoituksia, mutta suurimmaksi osaksi sitä voi käyttää tavallisen olion tapaan.

Serializable-luokan ilmentymää kutsutaan sarjallistetuksi olioksi (serializable object) ja näitä voidaan hallinnoida. On kuitenkin huomioitava, ettei tällä ole mitään tekemistä tietokannanhallintajärjestelmän (database management systems) käsitteen serializability kanssa. Jos sarjallistettu olio välitetään etämetodikutsun parametrina tai paluuarvona, olion arvo kopioidaan osoiteavaruudesta toiseen. Jos taas etäolio välitetään parametrina tai paluuarvona, kahva kopioidaan osoiteavaruudesta toiseen.

Teoriassa luokka voi olla sekä Remote että Serializable, mutta käytännössä näiden kahden sekoittaminen keskenään saa aikaan vain vaikeasti ymmärrettävän suunnittelun.

2.4.1 Serializable-luokat

Serializable-luokka on helpompi suunnitella kuin Remote. Luokka on Serializable (sarjallistettavissa), jos se laajentaa rajapintaa java.io.Serializable. Myös Serializable-luokan aliluokat ovat automaattisesti Serializable. On mahdollista sisällyttää ei-sarjallistettavia objekteja sarjallistettavien sekaan, mutta tämä ei ole järkevää. Yleensä Serializable-luokan sisältämän tiedon tulisi olla Serializable.

Sarjallistettavan olion käyttö etämetodikutsussa on suoraviivaista. Olio välitetään parametrin avulla tai paluuarvona, jonka tyyppi on Serializable-luokka. Sekä asiakkaalla että palvelimella on oltava pääsy kaikkiin käytettävissä oleviin Serializable-luokkien määritystiedostoihin. Jos asiakas ja palvelin pyörivät eri koneilla, Serializable-luokkien määritystiedostot täytyy ehkä ladata koneelta toiselle. Tällainen lataaminen voi vahingoittaa järjestelmän turvallisuutta.

2.4.2 Remote-luokat

Remote-luokka on vaikeampi määritellä kuin luokka Serializable. Remote-luokalla on kaksi osaa: rajapinta ja itse luokka. Remote-rajapinnan on oltava public ja sen tulee laajentaa rajapintaa java.rmi.Remote. Lisäksi rajapinnan jokaisen metodin tulee määritellä poikkeus java.rmi.RemoteException (tai jokin sen yläluokista kuten java.io.IOException tai java.lang.Exception). Poikkeus java.rmi.RemoteException esiintyy kun etämetodikutsu jostakin syystä epäonnistuu. Syitä voivat esimerkiksi olla protokollavirheet tai yhteydenpito voi epäonnistua, koska etäpalvelin kieltäytyy ottamasta yhteyttä. Myös muita poikkeuksia kuin java.rmi.RemoteException voi esiintyä.

Remote-luokan tulee toteuttaa rajapinta Remote ja sen tulisi laajentaa luokkaa java.rmi.server.UnicastRemoteObject. Tällaisen luokan oliot ovat olemassa palvelimen osoiteavaruudessa ja niitä voidaan kutsua etäältä. Remote-luokalla voi olla metodeita, joita sen Remote-rajapinnalla ei ole. Niitä voidaan kutsua vain paikallisesti (www.ccs.neu.edu 1998 [verkkodokumentti].)

Toisin kuin Serializable-luokkien kohdalla sekä asiakkaalla että palvelimella ei tarvitse olla pääsyä Remote-luokan määritystiedostoihin. Palvelin tarvitsee määritystiedostot sekä luokkaan että rajapintaan Remote, mutta asiakas käyttää vain rajapintaa Remote. Karkeasti sanottuna, rajapinta Remote edustaa olioviitteen tyyppiä ja luokka Remote olion tyyppiä. Jos etäoliota käytetään etäältä, tyypiksi on määriteltävä rajapinta Remote eikä luokkaa Remote.

2.5 Turvallisuus

Yleisin ongelma RMI:n käytössä liittyy turvallisuusrajoitusten aiheuttamiin epäonnistumisiin. Etämetodin kutsussa voi olla parametreina olioita, jotka välittyvät oliolle, jonka metodia on kutsuttu. Myös etämetodin kutsun paluuarvona voi olla olio. Molemmissa tapauksissa ohjelmaan tulee uutta, mahdollisesti tuntematonta koodia toiselta verkkokoneelta. Sen mukana voi tulla viruksia tai muuta haitallista ohjelmakoodia.

Ohjelmaan täytyy asentaa tietoturvaa hoitamaan SecurityManager-luokka, jos ohjelmalla halutaan olevan SecurityManager. Ilman määrittelyä ohjelmalla ei ole lainkaan SecurityManageria. RMI:ssä on RMISecurityManager-luokka, joka voidaan asettaa huolehtimaan ohjelman tietoturvasta. Asiakasohjelma asettaa aluksi RMISecurityManager-olion kutsumalla luokan System metodia setSecurityManager. RMI ei lataa mitään luokkia etäkohteista, ellei security manageria ole asetettu. Jotta RMISecurityManageria voisi käyttää, tulee seuraava koodinpätkä lisätä muun koodin sekaan (se on suoritettava ennen kuin RMI voi ladata koodia etäkohteista, joten sen on todennäköisesti oltava sovelluksen main-metodissa): System.setSecurityManager(new RMISecurityManager()); (C:\j2sdk142\docs 2003, [verkkodokumentti]). Tämän lisäksi tarvitaan policy-tiedosto, jossa on kuvattu tietoturva-asioita.

Monet Java-asennukset ovat ottaneet käyttöön turvallisuuspolitiikkoja, jotka sisältävät oletusarvoja enemmän rajoituksia. Tähän on hyvät syyt, eikä näitä turvallisuuspolitiikkoja tulisi ylikirjoittaa huolettomasti. Seuraavaksi kuitenkin keskitytään joihinkin tapoihin, joiden avulla voidaan ylikirjoittaa RMI:n toiminnan estäviä turvallisuuspolitiikkoja. Kuten aiemmin todettiin, turvallisuuspolitiikan saa päälle välittämällä SecurityManager-

tyyppisen olion luokan System metodiin setSecurityManager. On olemassa useita tapoja muokata ohjelman turvallisuuspolitiikkaa. Yksinkertaisin tapa on määritellä luokalle SecurityManager aliluokka ja kutsua System.setSecurityManageria aliluokan olioiden kohdalla. Luokalla SecurityManager on useita check-alkuisia metodeja ja aliluokan määritelmän kohdalla tulisi ylikirjoittaa ne check-metodit, joihin haluaa erilaisen politiikan. Jos ohjelma esimerkiksi kieltäytyy ottamasta yhteyttä registryyn, tulisi ylikirjoittaa checkConnect-metodit.

On olemassa kaksi checkConnect-metodia, checkConnect (String host, int port) ja checkConnect (String host, int port, Object context). Jos kutsu ensimmäiseen checkConnect-metodiin palautuu normaalisti, turvallisuuspolitiikka sallii ohjelman ottaa socket-yhteyden palvelin-socketiin määriteltyyn isäntään ja porttiin. Jos turvallisuuspolitiikka ei salli ottaa yhteyttä tähän isäntään ja porttiin, kutsu heittää poikkeuksen. Tällöin ohjelma yleensä päättyy ilmoitukseen kuten java.security.AccessControlException: access denied(java.net.SocketPermission 127.0.0.1:1099 connect,resolve). Tämä viesti esiintyy kun RMI-palvelin tai - asiakas ei saa yhteyttä RMI-registryyn, joka pyörii samalla koneella kuin palvelin tai asiakas. Toisella checkConnect-metodilla on kolmaskin parametri, joka määrittää pyynnön turvallisuussisällön. Seuraava koodinpätkä osoittaa miten tämä tehdään:

```
System.setSecurityManager (new RMISecurityManager()
{
        public void checkConnect (String host, int port) {}
        public void checkConnect (String host, int port, Object context) {}
};
```

Edellä kirjoitettu koodi käyttää nimetöntä sisäluokkaa (anonymous inner class). Tällainen luokka on käyttökelpoinen kun halutaan, että luokkaa käytetään ainoastaan luomaan olio yhteen paikkaan. Halutessaan voi tietysti myös määritellä luokan RMISecurityManager aliluokan tavalliseen tapaan.

SecurityManagerin määritteleminen ja asentaminen oli Javan alkuperäinen tekniikka turvallisuuspolitiikan määrittämiseen. On kuitenkin erittäin vaikeaa suunnitella luokka, jossa ei olisi lainkaan turvallisuusaukkoja ja tämän vuoksi Java 1.2:ssa esiteltiin uusi

(myös vanhan version kanssa yhteensopiva) tekniikka. Oletusarvoisesti SecurityManagerissa kaikki check-metodit (lukuunottamatta metodia checkPermission) toteutetaan kutsumalla metodia checkPermission. Tarkastettavan luvan tyypin määrittää tyypin Permission parametri, joka välitetään metodiin checkPermission. Esimerkiksi metodi checkConnect kutsuu metodia checkPermission olion SocketPermission kanssa. Oletusarvoisesti metodin checkPermission toteutus kutsuu luokan AccessController metodia checkPermission, joka tarkistaa onko määritelty lupa myönnettyjen lupien listalla. Luokkaa Permissions käytetään myönnettyjen lupien ylläpitämiseen ja sen tiedon tarkistamiseen onko jokin tietty lupa myönnetty.

Edellä kuvatulla mekanismilla SecurityManager tarkistaa luvat, mutta se ei selitä miten turvallisuuspolitiikkaa voi määritellä tai muuttaa. Tätä varten on olemassa oma luokkansa, nimeltään Policy. Kuten SecurityManagerilla, jokaisella ohjelmalla on ajankohtainen turvallisuuspolitiikkansa, johon pääsee käsiksi kutsumalla metodia Policy.getPolicy(). Turvallisuuspolitiikan voi asettaa metodilla Policy.setPolicy, jos siihen on oikeudet. Tyypillisesti turvallisuuspolitiikan määrittää politiikkakonfiguraatiotiedosto (policy configuration file tai lyhyesti policy file), joka luetaan ohjelman käynnistyessä ja aina kun saapuu pyyntö virkistää turvallisuuspolitiikka. Politiikkatiedosto määrittää Policy-objektin sisältämät luvat eikä ole kovinkaan väärin ajatella tätä tiedostoa eräänlaisena Policy-objektin sarjallistamisena (lukuun ottamatta sitä tosiasiaa, että politiikkatiedosto on tarkoitettu sekä ihmisten että koneiden luettavaksi). Oletusarvoisesti SecurityManager käyttää politiikkaa, joka on määritelty politiikkatiedostojen kokoelmassa. Jos käyttäjä haluaa myöntää lisäoikeuksia, hän voi määrittää ne politiikkatiedostoon. Politiikkatiedoston luontiin voidaan käyttää policytoolia, mutta sen voi tehdä myös tekstieditorilla.

3 ESIMERKKISOVELLUKSEN TOTEUTUS JA KÄYTTÖ

3.1 Tietoja esimerkkisovelluksesta

Esimerkkisovelluksen tietokannanhallintajärjestelmäksi on valittu MySQL AB:n kehittämä MySQL-tuote (versio 4.0.17). Sovelluksessa käytettävät luokat ClassFileServer ja ClassServer eivät ole omia luokkiani, vaan ne on löydetty Sunin sivuilta (java.sun.com, [verkkodokumentti]). Palvelinpuolelle kuuluvat luokat ClassFileServer, ClassServer, FileServiceException ja FileServiceImpl sekä rajapinta FileService, asiakaspuolelle kuuluvat luokat DefaultDocumentOperations, DocumentFrame, DocumentInfo ja FileExtensionFilter sekä rajapinta DocumentOperations.

3.2 Luokka ClassFileServer

Luokka ClassFileServer tuo (import) pakkauksesta java.io tarvitsemansa luokat ja poikkeukset. Luokka ClassFileServer laajentaa luokkaa ClassServer, joka lukee luokkatiedostoja tiedostojärjestelmästä. ClassFileServer on pienen web-palvelimen toteutus, jota RMI-sovellus voi käyttää luokkien lataamiseen verkon yli. Tyypillisesti RMI käyttää HTTP-palvelinta tiedostojen lataamiseen. ClassFileServer tarjoaa saman perustoiminnallisuuden, jonka se perii yläluokaltaan ClassServer.

Jotta luokka ClassFileServer toimisi, täytyy kääntää javatiedostot (ClassServer.java ja ClassFileServer.java) ja käynnistää ClassFileServer. Ennen käynnistystä on määriteltävä port (jossa palvelin pyörii) ja classpath (josta palvelin löytää luokkatiedostot). RMI-palvelinta käynnistettäessä sen tarjoaman codebase URL:in on yksinkertaisesti oltava URL, joka sisältää tiedon luokan ClassFileServer isännästä ja portista. RMI-palvelinsovelluksen sisään voi upottaa oman palvelimen sen sijaan, että sitä käyttäisi erikseen. Tämä onnistuu luomalla omalla palvelimellaan ClassFileServer ja tarjoamalla sille asiaankuuluvan porttinumeron ja polun:

```
import ClassFileServer;
//...
new ClassFileServer(port, classpath);
```

Konstruktorissa luodaan ClassFileServer, joka saa yläluokaltaan ClassServer tiedon käytettävästä portista. Lisäksi sillä on parametrina classpath, joka kertoo palvelimelle mistä luokat löytyvät. Konstruktori voi aiheuttaa poikkeuksen IOException.

Metodi getBytes(String path) palauttaa parametrin path määrittelemän luokan bytecodet bytetaulukko -muodossa. Parametri path on pisteellä erotettu luokannimi, josta on poistettu pääte ".class". Jos parametrin path määrittelemää polkua ei voitu ladata, ohjelma heittää poikkeuksen ClassNotFoundException. Meodissa luodaan File-olio, joka sisältää tiedon luettavasta polusta, ja tarkistetaan if-else-rakenteella onko sille annettu arvoa. Jos ei, sovellus heittää poikkeuksen IOException. Tämä poikkeus aiheutuu silloin kun tiedostoa ei voida lukea ja tällöin käyttäjä saa ilmoituksen siitä, ettei luettavaa polkua ole. Jos poikkeuksia ei esiinny, luodaan File-olion arvokseen saava FileInputStream-olio. Lisäksi luodaan DataInputStream-olio, joka saa arvokseen juuri luodun FileInputStream-olion. Tämän jälkeen luetaan bytecodet ja palautetaan ne.

Main-metodi on luokkatiedostoja lukevan luokkapalvelimen luomista varten. Se ottaa kaksi komentoriviparametria: port, johon palvelin hyväksyy pyynnöt sekä classpath. Main-metodissa tarkistetaan if-lauseilla komentorivin parametrit. Jos sekä port että classpath löytyvät, yritetään näillä tiedoilla luoda ClassFileServer. Jos tämä ei onnistu, otetaan kiinni poikkeus IOException ja ilmoitetaan käyttäjälle, ettei ClassServeriä voitu käynnistää. Tarkempia tietoja esiintyneestä ongelmasta käyttäjä saa metodien getMessage() ja printStackTrace() välityksellä.

3.3 Luokka ClassServer

Luokka ClassServer tuo pakkauksista java.io ja java.net tarvitsemansa luokat ja poikkeukset. Luokka ClassServer on abstrakti luokka, joka tarjoaa pienen web-palvelimen perustoiminnallisuuden ja on erikoistunut vain luokkatiedostojen lataamiseen. Luokassa ClassServer luodaan säie (thread), joka kuuntelee socketia ja hyväksyy HTTP GET-pyynnöt. HTTP-vastaus sisältää GET-otsikossa (header) pyydetyn luokan bytecodet. RMI-sovellus voi HTTP-palvelimen sijaan käyttää tämän palvelimen konkreettista aliluokkaa (ClassFileServer) etäluokkien lataamiseen. Aliluokassa on määriteltävä metodi

getBytes, joka on vastuussa luokan bytecodejen noutamisesta. Luokka ClassServer toteuttaa rajapinnan Runnable. Jos luokan ilmentymät on suunniteltu suoritettavaksi säikeessä, sen tulisi laajentaa tätä rajapintaa. Luokassa tulee määritellä parametriton metodi run. Rajapinta Runnable on suunniteltu tarjoamaan yleinen protokolla olioille, joiden halutaan suorittavan koodia aktiivisina ollessaan (C:\j2sdk142\docs 2003, [verkkodokumentti]). Aktiivisena oleminen tarkoittaa yksinkertaisesti sitä, että säie on käynnistetty eikä sitä ole vielä lopetettu.

Konstruktorissa luodaan ClassServer, joka kuuntelee parametria port ja käyttää metodia getBytes luokan bytecodejen saamiseksi. Konstruktorissa luodaan myös kuuntelija (newListener()). Jos ClassServer ei voinut kuunnella parametria port, ohjelma palauttaa poikkeuksen IOException.

Abstrakti metodi getBytes(String path) palauttaa parametrin path määrittelemän luokan bytecodet bytetaulukko -muodossa.. Parametri path on pisteellä erotettu luokannimi, josta on poistettu pääte ".class". Jos parametrin path määrittelemää polkua ei voitu ladata, sovellus heittää poikkeuksen ClassNotFoundException. Poikkeus IOException heitetään silloin kun luokan lukemisen aikana tapahtuu virhe.

Metodissa run() palvelimelle yhteyden hyväksyvä ”listen”-säie parseeraa otsikon saadakseen luokan nimen ja lähettää takaisin luokan bytecodet. Virhe palautuu, jos luokkaa ei löydetty tai vastaus oli väärin muodostettu. Yhteyden muodostuksen aikana voi esiintyä poikkeus IOException, jolloin käyttäjälle palautuu ilmoitus palvelimen kaatumisesta. Tarkempia tietoja esiintyneestä ongelmasta käyttäjä saa metodien getMessage() ja printStackTrace() välityksellä.

Metodissa run() luodaan uusi säie vastaanottamaan seuraava yhteydenottopyyntö (Tämä tehdään luomalla newListener()). Yritetään luoda DataOutputStream-olio, jonka jälkeen yritetään saada polku luokkatiedostoon luomalla DataInputStream-olio sekä hakea bytecodet metodilla getBytes(path) ja lähettää ne vastauksena. Tällöin voi esiintyä poikkeus IOException. Jos DataInputStream-olion luominen ei onnistu, palautuu poikkeus Exception ja käyttäjä saa ”error responsen”. Jos DataOutputStream-olion luominen ei onnistu, palautuu poikkeus IOException ja tieto siitä, että virhe tapahtui vastauksen kirjoittamisen aikana. Tarkempia tietoja esiintyneestä ongelmasta käyttäjä saa metodien get-

Message() ja printStackTrace() välityksellä. Jos kaikki edellä kuvattu on suoritettu onnistuneesti, lopuksi yhteys suljetaan. Tästäkin voi aiheutua poikkeus IOException.

Staattinen metodi getPath(DataInputStream in) palauttaa HTML-otsikon parseeraukses-ta saadun polun luokkatiedostoon. Polku löytyy riviltä, joka alkaa "GET /". Rivejä luetaan do-while-rakenteella niin kauan kuin niitä on. Jos polku löytyy, se palautetaan. Muulloin heitetään IOException ja ilmoitetaan käyttäjälle väärinmuotoillusta otsikosta.

3.4 Rajapinta FileService

Rajapinta FileService tuo pakkauksesta java.rmi tarvitsemansa rajapinnat ja poikkeukset. Rajapinta FileService laajentaa rajapintaa Remote ja on rajapinta, joka määrittää tiedostopalvelun tarvitsemat metodit. Rajapinta Remote tunnistaa rajapintoja, joiden metodeja voidaan kutsua ei-paikallisilta virtuaalikoneilta. Tätä rajapintaa tulee toteuttaa suoraan tai epäsuorasti jokaisen sellaisen olion, joka on etäolio. Vain rajapintaa java.rmi.Remote laajentavan rajapinnan metodit ovat saatavilla etäältä.

Metodi sendFile(byte[] bytes, String fileName) lähettää tiedoston palvelimelle. Sillä on kaksi parametria, bytetaulukko bytes ja String fileName. Parametri bytes on tiedoston sisältämä data, parametri fileName on tiedoston nimi. Metodi voi aiheuttaa poikkeukset RemoteException ja FileServiceException. Kuten jo aiemmin todettiin, metodin on määriteltävä heittävänsä ainakin poikkeus RemoteException. Myös muut poikkeukset ovat mahdollisia.

Metodi getFile(String fileName) hakee tiedoston palvelimelta. Sillä on yksi parametri, String fileName, joka on tiedoston nimi. Metodi voi aiheuttaa poikkeukset RemoteException ja FileServiceException. Kuten jo aiemmin todettiin, metodin on määriteltävä heittävänsä ainakin poikkeus RemoteException. Myös muut poikkeukset ovat mahdollisia.

3.5 Luokka FileServiceException

Luokka FileServiceException laajentaa luokkaa Exception. Luokka Exception ja sen aliluokat määrittävät poikkeukset, jotka sovelluksissa on hyvä ottaa kiinni. Luokka FileServiceException määrittää virheet, jotka voivat esiintyä esimerkkisovelluksen käytön aikana. Virhe voi aiheutua esimerkiksi silloin kun tiedostopalvelua yritetään käyttää ennen sen käynnistystä, tiedostoon kirjoittamisen tai tiedostosta lukemisen aikana tai kun haluttua tiedostoa ei löydetä. Luokka FileServiceException sisältää staattisia metodeja, joita voidaan kutsu käyttämällä luokan nimeä.

3.6 Luokka FileServiceImpl

Luokka FileServiceImpl tuo pakkauksista java.io, java.rmi ja java.rmi.server tarvitsemansa luokat ja poikkeukset. Luokka FileServiceImpl laajentaa luokkaa UnicastRemoteObject ja toteuttaa rajapinnan FileService. Luokka UnicastRemoteObject määrittää eikopioidun (non-replicated) etäolion, jonka viittaukset ovat voimassa vain palvelinprosessin ollessa elossa/päällä. Lisäksi se tarjoaa pisteestä-pisteeseen tuen (point-to-point) aktiivisille objektiviittauksille (kutsut, parametrit ja tulokset) käyttäen TCP-virtoja (C:\j2sdk142\docs 2003, [verkkodokumentti].)

Luokka FileServiceImpl on luokka tiedostopalvelun muodostamiseen ja tiedostopalvelu alustetaan sen konstruktorissa. Se sisältää kaiken toiminnallisuuden ja on tärkein luokka. FileServiceImpl perii super()-kutsulla yläluokaltaan FileService sen metodit ja kentät. Konstruktorissa voi esiintyä poikkeus RemoteException.

Metodissa sendFile(byte[] bytes, String fileName) yritetään luoda FileOutputStream-olio, jolla kirjoitetaan tiedoston sisältö. Metodi voi aiheuttaa kahdenlaisia FileServiceException-poikkeuksia. Jos tiedostoa ei löydy, otetaan kiinni poikkeus FileNotFoundException, haetaan luokasta FileServiceException sopiva virheilmoitus ja kutsutaan FileServiceException(FileServiceException.ERR_FILE_NOT_FOUND). Jos kirjoittaminen ei onnistu, otetaan kiinni poikkeus IOException, haetaan luokasta FileServiceException sopiva virheilmoitus ja kutsutaan FileServiceException(FileServiceException.ERR_WRITE_FILE).

Metodissa getFile(String fileName) yritetään luoda FileInputStream- ja DataInputStream-oliot, joilla luetaan tiedoston sisältö. Metodi voi aiheuttaa kahdenlaisia FileServiceException-poikkeuksia. Jos tiedostoa ei löydy, otetaan kiinni poikkeus FileNotFoundException, haetaan luokasta FileServiceException sopiva virheilmoitus ja kutsutaan FileServiceException(FileServiceException.ERR_FILE_NOT_FOUND). Jos lukeminen ei onnistu, otetaan kiinni poikkeus IOException, haetaan luokasta FileServiceException sopiva virheilmoitus ja kutsutaan FileServiceException(FileServiceException.ERR_READ_FILE). Molemmat oliot suljetaan operaatioiden jälkeen ja tiedoston sisältö palautetaan käyttäjälle.

Main-metodi luo ja asentaa security managerin, jos se ei löydä aiemmin määriteltyä. Metodissa yritetään luoda FileServiceImpl-olio ja liittää (bind) olioinstanssi nimeen "FileService". Käyttäjä saa ilmoituksen kun FileService on liitetty rekisteriin. Suorituksen aikana voi esiintyä poikkeus Exception, joka otetaan kiinni ja tulostetaan käyttäjälle tieto FileService-virheestä. Tarkempia tietoja esiintyneestä ongelmasta käyttäjä saa metodien getMessage() ja printStackTrace() välityksellä. Main-metodissa voi esiintyä myös poikkeus IOException.

3.7 Luokka DefaultDocumentOperations

Luokka DefaultDocumentOperations tuo pakkauksista java.awt, java.io, java.net, java.rmi ja java.sql sekä javax.swing tarvitsemansa luokat, poikkeukset ja rajapinnat. Luokka DefaultDocumentOperations suorittaa dokumenttioperaatioita ja toteuttaa rajapinnan DocumentOperations. Luokan konstruktorissa voidaan luoda kahdenlaisia DefaultDocumentOperations-olioita. Toinen on parametriton, toinen saa parametrinaan luokkaa käyttävän komponentin.

Metodi view(DocumentInfo docInfo) hakee tiedoston, jos sitä ei ole varattu käyttäjälle. Metodi edit(DocumentInfo docInfo) varaa tiedoston, jos sitä ei ole varattu käyttäjälle. Metodi reserve(DocumentInfo docInfo) varaa tiedoston. Jos tiedosto on jo varattu kyseiselle käyttäjälle, varmistetaan haluaako käyttäjä hakea sen uudelleen. Metodi release(DocumentInfo docInfo) vapauttaa tiedoston ja poistaa sen käyttäjän tiedostoista

varmistettuaan ensin haluaako käyttäjä päivittää tekemänsä muutokset. Metodi importFile(DocumentInfo docInfo) vie tiedoston palvelimelle. Metodi exportFile(DocumentInfo docInfo) hakee tiedoston palvelimelta. Kaikki edellä mainitut metodit voivat aiheuttaa poikkeukset SQLException ja FileServiceException.

Metodi sendFile(DocumentInfo docInfo, File file) lähettää tiedoston palvelimelle. Sillä on kaksi parametria, docInfo ja file. Parametri docInfo on dokumentin tiedot ja file lähetettävä tiedosto. Metodissa yritetään luoda FileService-, FileInputStream- ja DataInputStream-oliot sekä lukea DataInputStream-olion sisältö. Suorituksen jälkeen FileInputStream- ja DataInputStream-oliot suljetaan ja FileService-olio lähettää tiedoston metodilla sendFile(). Metodi voi aiheuttaa poikkeuksen FileServiceException.

Metodi getFile(DocumentInfo docInfo, File file) hakee tiedoston palvelimelta. Sillä on kaksi parametria, docInfo ja file. Parametri docInfo on dokumentin tiedot ja file käyttäjän työasemalle kirjoitettava tiedosto. Metodissa yritetään luoda FileService- ja FileOutputStream-oliot sekä kirjoittaa DataOutputStream-olion sisältö. Suorituksen jälkeen DataOutputStream-olio suljetaan. Metodi voi aiheuttaa poikkeuksen FileServiceException.

Metodi startApplication(String program, String fileName) käynnistää sovelluksen. Parametri program on käynnistettävän sovelluksen nimi ja parametri fileName avattavan tiedoston nimi. Jos käynnistys ei onnistu, käyttäjä saa virheilmoituksen.

Metodi resolveFileName(DocumentInfo docInfo) muodostaa tiedoston nimen dokumentin koodista ja tyypin tarkenteesta ja palauttaa tiedon käyttäjälle. Metodi resolveUserFileName(DocumentInfo docInfo) muodostaa käyttäjän työasemalla olevan tiedoston nimen ja palauttaa tiedon käyttäjälle. Metodi isReserved(DocumentInfo docInfo) tarkistaa, onko käyttäjä varannut tiedoston. Jos käyttäjä on varannut tiedoston, metodi palauttaa totuusarvon true eli tosi. Kaikissa metodeissa esiintyvä parametri docInfo on dokumentin tiedot.

3.8 Luokat DocumentFrame ja DocumentInfo

Luokka DocumentFrame tuo pakkauksista java.awt, java.awt.event, java.sql sekä javax.swing tarvitsemansa luokat, poikkeukset ja rajapinnat. Luokka DocumentFrame laajentaa abstraktia luokkaa DataFrame ja toteuttaa rajapinnan DbOperations. Se määrittää esimerkkisovelluksen ulkonäön. Luokka DocumentInfo sisältää dokumentin tiedot.

3.9 Rajapinta DocumentOperations

Rajapinta DocumentOperations tuo pakkauksesta java.sql tarvitsemansa poikkeuksen. Rajapinta DocumentOperations on rajapinta, joka määrittää dokumentin käsittelyyn liittyvät operaatiot. Näitä operaatiot ovat katselu (view), muokkaus (edit), varaus (reserve), vapautus (release), tiedoston vienti (import) sekä tuonti (export).

Metodi view(DocumentInfo docInfo) hakee tiedoston ja avaa sen katseltavaksi. Metodi edit(DocumentInfo docInfo) hakee ja varaa tiedoston ja tämän jälkeen avaa sen muokattavaksi. Metodi reserve(DocumentInfo docInfo) hakee ja varaa tiedoston. Metodi release(DocumentInfo docInfo) vapauttaa tiedoston ja vie sen haluttaessa palvelimelle. Metodi importFile(DocumentInfo docInfo) vie tiedoston palvelimelle ja liittää sen dokumenttiriville. Metodi exportFile(DocumentInfo docInfo) hakee tiedoston palvelimelta ja tallentaa sen haluttuun paikkaan. Kaikissa metodeissa esiintyvä parametri docInfo on dokumentin tiedot ja jokainen niistä voi aiheuttaa poikkeukset SQLException (tietokantavirheestä aiheutuva poikkeus) ja FileServiceException (tiedostonsiirtovirheestä aiheutuva poikkeus).

3.10 Luokka FileExtensionFilter

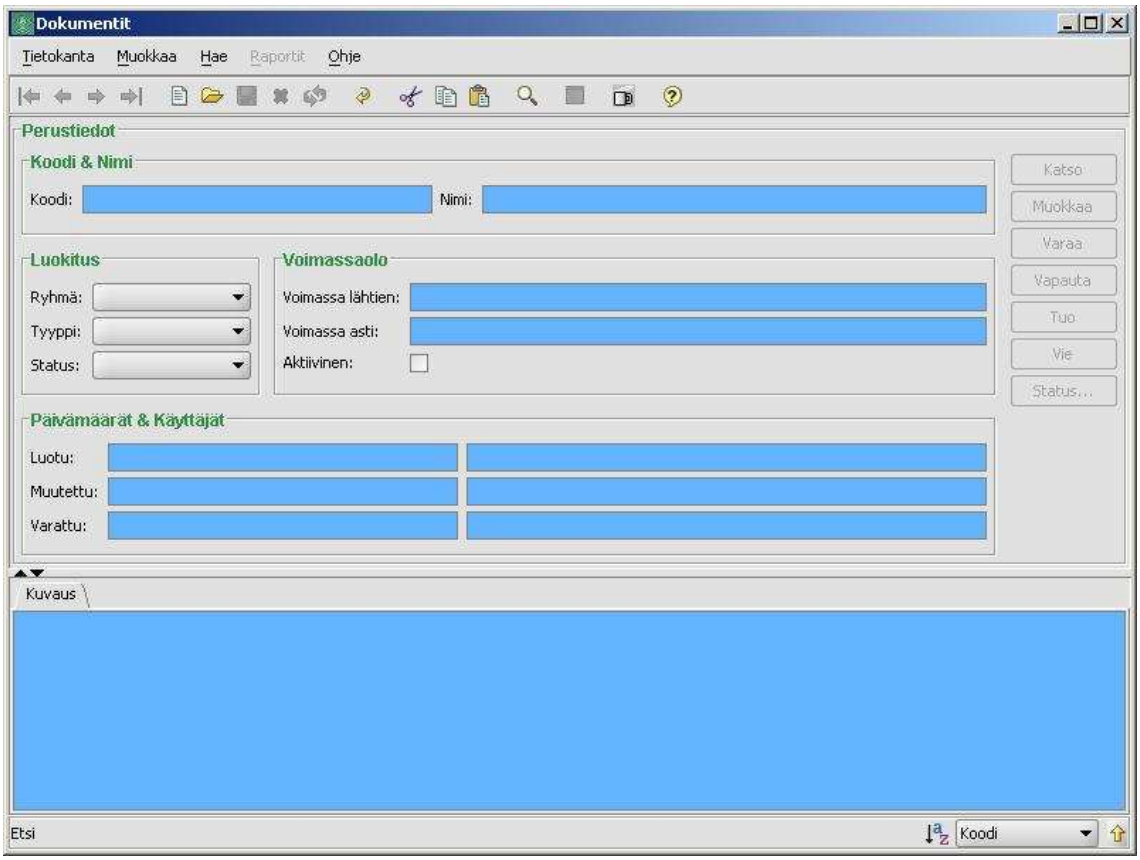
Luokka FileExtensionFilter tuo pakkauksista java.io, java.util seka javax.swing tarvitsemansa luokat ja rajapinnat. Luokka FileExtensionFilter laajentaa rajapintaa FileFilter. Tämä rajapinta suodattaa pois kaikki ne tiedostot, joiden päätettä se ei tunnista.

Konstruktoreita voi olla viidenlaisia. Parametriton FileExtensionFilter hyväksyy kaikenlaiset tiedostot. FileExtensionFilter(String extension) luo suodattimen, joka hyväksyy tiettyjä päätteitä. FileExtensionFilter(String extension, String description) luo suodattimen, joka hyväksyy kaikki annetut tiedostotyypit. FileExtensionFilter(String[] filters) luo suodattimen annetusta string-taulukosta. FileExtensionFilter(String[] filters, String description) luo suodattimen annetusta string-taulukosta ja kuvauksesta. Kolmen viimeksi mainitun konstruktorin kohdalla päätteen edellä olevaa pistettä ei tarvita, eikä sitä huomioida.

Metodi addExtension(String extension) lisää pisteen (".") tiedostonnimeen ja metodi getExtension(File f) palauttaa tiedostonpäätteen. Metodi setDescription(String description) asettaa selkokielisen esityksen suodattimesta ja metodi getDescription() palauttaa sen. Metodi setExtensionListInDescription(boolean b) päättää tuleeko päätelistan olla luettavissa selkokielisessä kuvauksessa ja metodi isExtensionListInDescription() palautuu, jos näin on. Kaksi viimeksi mainittua metodia ovat tarpeellisia vain, jos kuvaus tehtiin konstruktorissa tai metodilla setDescription().

3.11 Asiakassovellus

Asiakassovellus on yksityiselle toiminimelle tehty taloushallintojärjestelmä, johon tiedostopalvelu on lisätty. Kuvassa 6 on esitelty sovelluksen ulkonäkö. Kun ikkuna avataan, tiedostopalveluun liittyviä painikkeita ei voi käyttää. Vasta kun käyttäjä on avannut jo olemassa olevan tiedoston tai luonut uuden, painikkeet ovat käytettävissä. Pakollista tietoa ovat kentät Koodi ja Nimi sekä alasvetovalikot Ryhmä ja Tyyppi.



Kuva 6. Asiakassovelluksen ulkonäkö

Katso: Käyttäjä voi katsella luomaansa (tai toisen luomaa) tiedostoa (tyypiltään tiedosto voi olla tekstitiedosto, Excel-taulukko, JPEG-/GIF-kuva...).

Muokkaa: Käyttäjä voi muokata omaa tiedostoaan tai jonkun toisen tiedostoa. Tiedoston sulkemisen yhteydessä sovellus kysyy "Haluatko päivittää tiedoston?" Vastausvaihtoehtoina on kyllä, ei ja peruuta.

Varaa: Käyttäjä voi saada ilmoituksen: "Olet jo varannut tämän tiedoston. Haluatko varata sen uudelleen?" Vastausvaihtoehtoina on kyllä ja ei. Jos käyttäjä yrittää varata tiedoston ennen kuin tiedostopalvelu on päällä, hän saa virheilmoituksen (kuva 7).

Kuva 7. Virheilmoitus siitä, ettei tiedostopalvelu ole päällä.

Vapauta: "Haluatko päivittää tiedoston?" Vastausvaihtoehtoina kyllä, ei ja peruuta. Jos tiedostoon on tehty tallentamattomia muutoksia, sovellus varmistaa ”Haluatko tallentaa tiedot?” Käyttäjä voi vastata kyllä tai ei.

Tuo: Käyttäjä voi hakea tietyntyyppisiä tiedostoja. Tyyppi määritellään Luokitus-osiossa alasvetovalikosta Tyyppi. Jos tiedoston tyypiksi on määritelty Notepad, vain ".txt"-päätteiset tiedostot ovat käytössä. Tuo-ikkunassa (kuva 8) valittu tyyppi näkyy kohdassa Files of Type.

Kuva 8. Asiakassovelluksen Tuo-ikkuna

Vie: Käyttäjä voi viedä omia tiedostojasi haluamaansa paikkaan. Tämän ikkunan ulkonäkö on samanlainen kuin Tuo-ikkunalla, mutta se toimii toiseen suuntaan.

Status: Status on käyttäjän määriteltävissä. Esimerkkisovelluksessa status voi olla "Tarkistettu", "Hylätty" tai "Keskeneräinen".

4 YHTEENVETO

Edellä on kuvattu tiedostopalvelun tekeminen RMI-tekniikalla ja esitelty yksinkertainen esimerkkisovellus. Tiedostopalvelun toteutus esimerkkisovellukseen ei ollut kovinkaan vaikea tehtävä, koska järjestelmä oli jo ”valmis”. Jos en olisi tehnyt taloushallintojärjestelmää jo aiemmin, en olisi tähän projektiin ryhtynyt. Opinnäytetyöhön varattu kuusi kuukautta olisi todennäköisesti riittänyt, mutta oma aikatauluni oli tiukempi. Se jätti työn tekemiseen aikaa vain pari kuukautta. Olin kuitenkin tutustunut materiaaliin jo parin kuukauden ajan ennen opinnäytetyön aloittamista.

Vaikeimmaksi osuudeksi opinnäytetyön tekemisessä osoittautui tärkeän ja tarpeellisen tiedon erottelu kaiken saatavilla olevan materiaalin joukosta. Lisäksi joillekin englanninkielisille termeille oli erittäin vaikea keksiä hyvää suomenkielistä vastinetta. Tällaisissa tapauksissa olen useimmiten kirjoittanut englanninkielisen termin sulkeissa suomenkielisen perään.

Tiedostopalvelu oli kovin yksinkertainen ja luultavasti parantelen sitä vielä joskus tulevaisuudessa. Tällä hetkellä esimerkiksi virheilmoitukset ovat vain englanninkielisiä. Niihin on tarkoitus lisätä kielituki myös suomenkielelle, sillä koko alkuperäinen taloushallintojärjestelmä on käytettävissä sekä englannin- että suomenkielellä. Alun perin taloushallintojärjestelmä oli tarkoitus ottaa käyttöön, mutta silloisen asiakkaani suunnitelmat muuttuivat. Olisi hienoa, jos järjestelmä joskus otettaisiin käyttöön. Haluaisin tietää kuinka paljon sinne on jäänyt virheitä, joita en itse ole huomannut. Ja tietysti haluaisin myös saada ulkopuoliselta käyttäjältä palautetta tiedostopalvelun toimivuudesta.

Javalla on ollut oma, sisäänrakennettu ORB (Object Request Broker) nimeltään RMI (Remote Method Invocation) JDK:n versiosta 1.1 lähtien (www.omg.org 1997, [verkkodokumentti].) Ohjelmoijilla on jo kokemusta siitä miten teknologia toimii ja joillakin yrityksillä on jo ehkä käytössä RMI:tä käyttäviä järjestelmiä. RMI:tä kehitetään koko ajan ja tulevaisuudessa nähdään toimiiko RMI-IIOP.

LÄHDELUETTELO

Verkkolähteet:

Baclawski, Kenneth [verkkodokumentti], 1998, [viitattu lokakuussa 2006]: ”Java RMI tutorial” Saatavissa http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

Curtis, David [verkkodokumentti], 1997, [viitattu lokakuussa 2006]: ”Java, RMI and CORBA” Saatavissa http://www.omg.org/news/whitepapers/wpjava.htm

Gopalan Suresh Raj [verkkodokumentti], ei päiväystä, [viitattu lokakuussa 2006]: ”A detailed comparison of CORBA, DCOM and Java/RMI” Saatavissa http://my.execpc.com/~gopalan/misc/compare/html

Hagge, Damian [verkkodokumentti], 12.3.2002, [viitattu lokakuussa 2006]: “RMI-IIOP in the enterprise - An introduction to running RMI over IIOP” Saatavissa http://www-128.ibm.com/developerworks/java/library/j-rmi-iiop/

Reilly, David [verkkodokumentti], ei päiväystä, [viitattu lokakuussa 2006]: “Introduction to Java RMI” Saatavissa http://www.javacoffeebreak.com/articles/javarmi/javarmi.html

Ei tekijää [verkkodokumentti], ei päiväystä, [viitattu lokakuussa 2006]: ”Java RMI specification” Saatavissa http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html

Ei tekijää [verkkodokumentti], 2003, [viitattu lokakuussa 2006] ” API Specification” Saatavissa C:\j2sdk142\docs\api\index.html

Ei tekijää [verkkodokumentti], ei päiväystä, [viitattu lokakuussa 2006]: ”Java RMI & CORBA - A comparison of two competing technologies” Saatavissa http://www.javacoffeebreak.com/articles/rmi_corba/index.html

Ei tekijää [verkkodokumentti], ei päiväystä, [viitattu lokakuussa 2006] Saatavissa http://myy.helia.fi/~atk84d/rmi/rmi.html