# RMI –

## TOIMIVAA ETÄPROSEDUURIOHJELMINTIA

## 17.12.2007

**Liite 1. analyysi ryhmän toiminnasta**
**Liite 2. XML:t kysymyksistä**

# RMI 4

# RMI

## Johdanto

RMI on Javan malli ja mekanismi Remote procedure calli:sta (RPC:stä), eli proseduurin etäkutsusta. Etäkutsua voi ajatella ohjelmointirajapintana (engl. Application programming interface, API), jolla eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja sekä keskustella keskenään. RMI:ssä on kuitenkin useita tekijöitä, jotka tekevät siitä monipuolisemman kuin perinteinen RPC. Tämä johtuu siitä, että RMI on osa Javan objektikeskeistä lähestymistapaa.

RMI antaa mahdollisuuden suoraviivaiseen ja yksinkertaiseen tapaan toteuttaa hajautettuja operaatioita Java-objekteilla. Objektit voivat olla joko Java-objekteja (Object) tai yksinkertaisia Java-kääreitä. Kääreet tarjoavat ainoan tavan varastoida Javan alkeistyypit (int, string, lang) object-luokan osaksi.

## RMI:n edut

Yksi RMI:n parhaista puolista on olioiden hallinta verkon yli keskitetyllä rakenteella. Koska RMI on keskittynyt Javan ympärille, se tuo mukanaan Javan tehokkuuden, turvallisuuden ja siirrettävyyden hajautettuun laskentaan. RMI:n päälle on rakennettu APEja, joista esimerkkinä olkoon JNI (Java Native Method Interface). JNI on suosittu ohjelmointiympäristö, koska sitä on helpompi käyttää kuin RMI:tä suoraan. Lisäksi se sallii muilla kielillä ohjelmoidun koodin suorittamisen Javan virtuaalikoneessa (JVM). Tällaisia kieliä ovat mm. C, C++ ja Assembly. RMI/JNI ja RMI/JDBC -yhdistelmien avulla on mahdollista kommunikoida olemassa olevien ohjelmien ja palvelimien kanssa. Relaatiotietokantoja voi käyttää JDBC:n avulla, oli relaatiokanta kirjoitettu millä tahansa relaatiotiekantaohjelmalla.

RMI:n päälle rakennettu Java Media Framework API (JMF) mahdollistaa äänen, videon ja muun aikasidonnaisen median liittämiseen Java sovelluksiin ja sovelmiin (applet). Tämä on luokka, jolla voidaan kaapata, toistaa, suoratoistaa (streemata) kuvaa tai ääntä, laajentaa Java 2-alustaa (J2SE) multimediakehittäjille. Se tarjoaa työkaluja alustasta ja paikasta riippumattomien sovellusten kehittämiseen kuten kuvan resoluution (skaalaus) muuttamisen päätelaitteelle sopivaksi.

### Objektikeskeisyys

RMI osaa välittää kokonaisia objekteja parametreina ja paluuarvoina pelkkien ennalta määriteltyjen tyyppien sijaan. Perinteisessä RPC-järjestelmässä asiakasohjelma joutuisi

purkamaan esim. Java-tiivistetauluobjektin useaan alkeelliseen tietotyyppiin, siirtämään nämä ja luomaan tiivistetaulun uudelleen palvelimen puolella.

RMI:n avulla voidaan käsittelymallia muuttaa asiakkaalta palvelimelle ja palvelimelta (server) asiakkaalle (client). Tällaisesta toimii esimerkkinä vaikka uuden Java-luokan luominen tietyn lomakkeen käsittelyä varten. Tällöin asiakas hakee lomakkeen palvelimelta lomakkeen käsittelyn muuttuessa. Asiakkaan ei tarvitse välittää mahdollisista muutoksista, sillä se saa ne automaattisesti noutaessaan lomakkeen palvelimelta. Tarkistukset hoidetaan asiakkaan päässä, jolloin mahdolliset virheet tulevat ilmi heti ja säästytään turhalta tietoliikenteeltä. Ilman tätä jouduttaisiin joko päivittämään jokaisen asiakaskoneen käsittelijä, tai tarkistamaan reaaliaikaisesti palvelimelta vastaako asiakkaan lomakkeentäyttö nykyistä käsittelymallia, mistä seuraisi turhaa tietoliikennettä. Sama ongelma syntyisi jos lomake tarkistetaan kokonaisuudessaan palvelimessa.

## Helppo luoda ja käyttää

RMI:n avulla on helppoa luoda etäpalvelimia ja asiakassovelluksia, jolloin kaikkien etäkäyttöliittymänä toimii oikea Java käyttöliittymä. RMI-palvelimen luominen on kokonaisuudessaan noin kolme riviä koodia ja on muuten ominaisuuksiltaan samanlainen kuin mikä tahansa Java-luokka. Tämän yksinkertaisuuden johdosta on helppo kirjoittaa täyden mittakaavan hajautettuja objekteja nopeasti, mikä luonnollisesti nopeuttaa testausta, lopullista käyttöönottoa ja ylläpidettävyyttä.

## Rinnakkaislaskenta ja Javan hajautetut laskentaratkaisut

RMI on monisäikeinen, minkä ansiosta palvelin pystyy paremmin käsittelemään asiakkaiden rinnakkaisia pyyntöjä. Kaikki RMI-järjestelmät käyttävät samaa julkista protokollaa, jolloin ne voivat kommunikoida keskenään suoraan ilman protokollan vaihdoksista aiheutuvia viipeitä.

## Hajautettu roskienkeruu

RMI käyttää hajautettua roskienkeruuta poistamaan palvelimen objekteja, joihin mikään verkon asiakas ei enää viittaa. Objektien elinaikaa valvoo Java.rami.dac.leaseValue –luokka. Aikaa mitataan millisekunneissa ja oletusarvo on 10 min. Roskienkeruun takia asiakkaan tulee varustautua siihen, että oletettu etä-yhteys on ”kadonnut”.

## RMI ja palomuuri

Internetissä sijaitsevat asiakkaat ovat yleensä eristettynä palomuurien taakse. RMI tarjoaa palomuurin takana oleville asiakkaille keinot kommunikoida palvelimen kanssa. RMI osaa valita nopeimman mahdollisen tavan yhdistää asiakas ja palvelin. Tämä tapahtuu käyttämällä UnicastRemoteObjectia ensimmäisellä yhdistyskerralla, jolloin valitaan jokin seuraavista kommunikointitavoista:

1. Kommunikoidaan suoraan palvelimen kanssa käyttäen pistokkeita (socket).
   Nopein tapa toimia sisäverkossa. Julkisen tietoverkon yli yhteydenotto vaatii palomuurien konfigurointia.
2. Jos tämä ei onnistu, luodaan URL-osoite palvelimeen ja porttiin, jota RMI-palvelin osaa kuunnella, ja lähetetään HTTP POST -pyyntö palvelimen rungolle (skeleton).
3. Viimeinen vaihtoehto on luoda URL-osoite palvelimeen käyttäen porttia 80 ja CGI-skriptin avulla lähettää RMI-pyyntö palvelimelle.
   Hitain tapa toimia.

Kommunikointitavaksi valitaan se joka näistä ensimmäiseksi onnistuu. Tapa muistetaan ja käytetään myös jatkossa asiakkaan ja palvelimen väliseen kommunikointiin.

Jos mikään edellä mainituista ei onnistu, RMI -kutsu epäonnistuu. Tietoturvan kannalta RMI on tämän piirteen takia parempi kuin esim. RCP. Lisäksi RMI käyttää Javan sisäänrakennettuja turvamekanismeja. Java.security.providers-luokka eristää turvallisuustarkastajan (SecurityManager) käsityksen Javan alustasta. Se määrittelee tarkastajan nimen ja listan siitä mitä turvallisuuspyyntö saa tehdä (JDK 1.6.X).

## Palvelimen tietoturvaratkaisuja

Palvelimelle voidaan asettaa Javan standarditurvallisuusehdot. Tämä varmistaa, että haluttaessa palvelin ei lataa mitään ilmentymiä.

Turvallisuustarkastaja valvoo kaikkia sovelmia, jolloin niiden mahdollista toimintaa voidaan rajoittaa. Javan kääntäjä osaa etsiä mahdollisia yrityksiä hyödyntää esimerkiksi yli- ja alivuotoja ja muita mahdollisia menetelmiä, joilla hyökkäys voidaan tehdä.

Javan käytölle on määritelty selkeät säännöt:

1. Java Applet ei saa kirjoittaa tai lukea tiedostoja.
2. Java Applet ei saa avata verkkoyhteyksiä muualle kuin sille sivulle, josta se on alun perin ladattu.
3. Java Applet ei saa käynnistää uusia prosesseja tai ohjelmia.
4. Java Applet ei saa käyttää natiiveja metodeja.
5. Java Applet ei saa kutsua mitään käyttöjärjestelmän palveluita.
6. Asiakas hakee palvelimelta sopivan palvelun.

Ja samat pelisäännöt koskevat RMI:tä.

## Sovelmat VS RMI

Java Applet on selkeä ja hyvä tapa hoitaa kahden palvelun välistä yhteyttä internetin yli. Kumpikaan ei ole tietoinen siitä, millaisen ohjelman kanssa keskustelevat, yhteistä on vain rajapinta. Selaimen kautta hoidettava yhteys on standardi – tietyin rajoituksin.

Vaikka RMI:n käyttäminen sovelmien kautta tuntuu hyvältä idealta. Käytännössä RMI:n rajoitukset tekevät kuitenkin RMI:n kanssa käytettävien sovelmien epäkäytännölliseksi. Jos tarvetta on, ongelma voidaan kiertää käyttämällä Java servlettejä ja olioiden sarjallistamista (serialization) sovelma – servletti kommunikoinnille.

## Koodikirjaston käyttäminen sovelmissa ja Java RMI:ssä

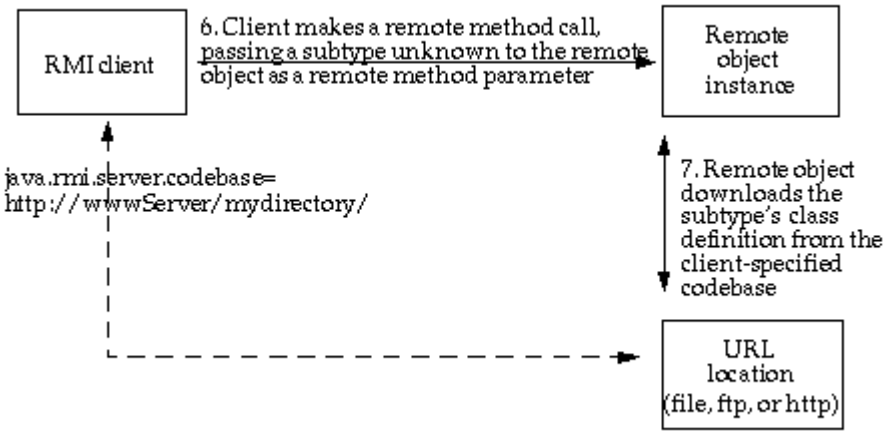
Java RMI -applikaatiot voivat luoda etä-objekteja, jotka hyväksyvät metodikutsuja toisilta virtuaalikoneilta (asiakkailta).

Koodivaraston (codebasen) suorittaman tallennuksen tekee ClassLoader –luokka. Kun Java -ohjelma aloittaa ClassLoader –luokan suorittamisen, tulee sen tietää mihin luokat ladataan. Esimerkiksi CLASSPATH on tietokoneen paikallinen "codebase" (koodikirjasto) - lista paikoista mihin paikalliset luokat ovat siroteltu.

Java RMI codebase on enemmän kuin vain stubin lataamista. Stubiin liittyvien luokkien ja niihin liittyvät luokat asiakkaille java.rmi.server.codebase ominaisuutta voidaan käyttää erityisestä paikasta, mihin jokainen luokka - ei ainoastaan stubit - voidaan ladata.

Kun asiakas tekee metodikutsun etä-oliolle, se voidaan hyväksyä ilman tai argumenttien kanssa. On olemassa kolme tapausta, riippuen datasta ja argumenttien määrästä:

1. Jos kaikki metodin parametrit (ja paluuarvo) ovat primitiivejä (int, long), etä-olio tunnistaa ne metodin parametreiksi, eivätkä ne tarvitse tarkistusta CLASSPATH:in tai muusta codebase:sta.
2. Jos on olemassa ainakin yksi palautettava etä-metodi tai objekti, jonka kuvaus on CLASSPATH luokka-kirjastossa, käsitellään ja palautetaan se.
3. Jos etä-metodi saa luokan ilmentymän (kuten on nähtävissä askel 6, kuva 1), jota etä-objekti ei löydä palvelimen paikallisista luokan määrittelyistä (CLASSPATH). Luokka käsitellään määriteltyjen parametrien alatyyppinä. Asiakkaan lähettämä objektin luokka eli alatyyppi käsitellään määriteltynä parametrina. Alatyyppi voi olla:
    a. Ilmentymän rajapinta kuvataan metodiparametri- (tai paluu)tyyppisenä.
    b. Luokan alaluokka kuvataan metodiparametri- (tai paluu)tyyppisenä.




Kuva 1: Java RMI asiakas suorittamassa etä-metodikutsua välittäen tuntematonta alatyyppiä metodina.

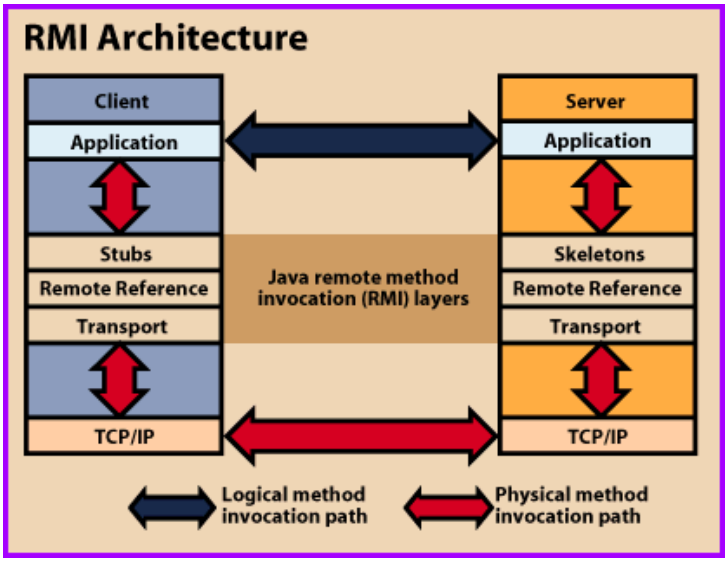
Toisin kuin sovelmilla, RMI:llä on mahdollista ajaa olioita toisella VM:llä.

## RMI:n toteuttaminen

RMI on ollut osa Javaa alkaen JDK 1.1, joten se löytyy jokaisessa sen jälkeisessä Javan virtuaalikoneessa.

RMI-sovellukset koostuvat pääosin kahdesta erillisestä ohjelmasta: palvelinoliosta ja asiakasoliosta (server/client). Metodia kutsuva osapuoli on asiakas, ja kutsuttava osapuoli on palvelin. Tällaisia sovelluksia kutsutaan joskus hajautetuiksi oliosovelluksiksi (distributed object application).

Rmi:ssä on kolme tasoa: 1. stub/skeleton layer, 2. remote/reference layer ja 3. transport layer:




Kuljetuskerros luo yhteydet JVM:ien välille. Kaikki yhteydet ovat suoratoistopohjaisia ja käyttävät TCP/IP:tä. Vaikka kaksi eri JVM:ää ovat suorituksessa samassa tietokoneessa, ne muodostavat yhteyden toisiinsa TCP/IP:n kautta.

### Etä-palveluiden nimeäminen ja luominen

Palvelinohjelma luo palvelun luomalla ensin paikallisen olion, joka toteuttaa halutun palvelun. Seuraavaksi se siirretään RMI:hin. Kun olio on siirretty, RMI luo palvelun joka odottaa asiakkaan kutsua. Lähetyksen jälkeen se rekisteröidään RMI:iin rekisteriin julkisella nimellä.

Asiakkaan näkökulmasta RMI-rekisteri tavoitetaan staattisen Naming-luokan kautta. Se sisältää metodin tarkista() (lookup), jolla hyväksytään URLin (Uniform Resource Locator), joka määrittelee palvelimen nimen ja halutun palvelun.

Esim: Teemme todella yksinkertaisen palvelun. Laskuri-palvelussa asiakas pyytää palvelinta laskemaan annetun numeron yhteen itsensä kanssa.

Ensin on suunniteltava yhteys:

```
rmi://localhoast (host_name)
      [:<xxxx] (name_service_port)
        /<laskuri> (Service_name)
```

Host_name RMI://<loclhoast:n täytyy olla joko sisäverkossa LANin (localhost) tai internetissä DNS:n tunnistama. Name_service_port_ -portti tarvitaan ainoastaan, jos se poikkeaa oletusarvosta 1099. Lipuksi tarvitaan halutun palvelun nimi.

Toimiva RMI systeemi on koostettu monista osista:

1. Rajapinnan toteuttavan luokan luonti
2. Etäpalvelun luonti
3. Stub ja Skeleton
4. Palvelimen host etä-palvelut
5. RMI nimeämispalvelu (Naming), joka sallii asiakkaiden löytää etäpalvelut
6. Luokka (HTTP tai FTP palvelin)
7. Asiakasohjelma, joka tarvitsee etäpalveluita

### Rajapinnan toteuttavan luokan luonti

```
rmi://<localhoast>
      [:<1720] /<laskuri>

public interface Laskuri
          extends Java.rmi.Remote {
    public long add(long a)
        throws Java.rmi.RemoteException;
}
```

### Etäpalvelun luonti

```
ImpLaskuri      // ilmentymän (implements) luominen

public class ImpLaskuri
    extends Java.rmi.server.UnicastRemoteObject
    implements Laskuri {

    //RemoteException exception

   public ImplLaskuri()
        throws Java.rmi.RemoteException {
        super();
        }

    public long lisaa(long a)
        throws Java.rmi.RemoteException {
        return a + a;
        }
}
```

## Skeletonin ja Sutbin luonti

```
// rmic -käännös
rmic ImlLaskuri

  -vcompat  (default)
            Create stubs/skeletons compatible
                 with both JDK 1.1 and Java 2
                     stub protocol versions
```

## Palvelimen luonti

```
import Java.rmi.Naming;

public class Laskuri {

   public Laskuri() {
     try {
        Laskuri L = new ImpLaskuri();
        Naming.rebind("rmi://localhoast/laskuri", l);
        } catch (Exception e) {
                System.out.println("Pieleen man: " + e);
                }
            }

        public static void main(String args[]) {
                new Laskuri();
        }
}
```

### Asiakkaan luonti

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

public class LaskuriAs {

    public static void main(String[] args) {
        try {
            Laskuri L = (laskuri)
                          Naming.lookup(
                              "rmi://localhost/Laskuri)

            System.out.println( l.add(3) );
        }
        catch (MalformedURLException urlmurhe) {
            System.out.println();
            System.out.println(
              "MalformedURLException");
            System.out.println(urlmurhe);
        }
        catch (RemoteException remot) {
            System.out.println();
            System.out.println(
                         "RemoteException");
            System.out.println(remot);
        }
        catch (NotBoundException eisidosta) {
            System.out.println();
            System.out.println(
                        "NotBoundException");
            System.out.println(eisidosta);
        }
        catch (Java.lang.ArithmeticException matuvirhe) {
            System.out.println();
            System.out.println(
             "Java.lang.ArithmeticException");
            System.out.println(matuvirhe);
        }
    }
}
```

Remote Object rajapinnan metodeilla voi asiakas kutsua palvelimelta palvelua.

RMI:ssä käynnistysjärjestys on tärkeä:

1. Rajapinnan toteuttavan luokan luonti
2. Etäpalveluiden toteutus
3. Stub ja Skeleton tiedostot
4. Palvelin
5. RMI Naming-palvelu
6. Luokka (HTTP tai FTP palvelin)

7. Asiakasohjelma

1. Javac-kääntäjä kääntää lähdekooditiedostot, jotka sisältävät palvelin-, ja asiakasluokkien sekä etärajapintojen toteutukset.
2. Rmic-kääntäjä luo stubit ja skeletonit etäolioille. Stub toimii proxyna etäoliolle ja on vastuussa etäolioiden metodikutsujen välittämisestä palvelimelle.

## Palvelin- ja asiakasoliot (API-luokat)

### Palvelimen rajapintoja

Java.lang.Object
  └ **Java.rmi.server.RemoteObject**

All Implemented Interfaces:
Remote, Serializable
Direct Known Subclasses:
RemoteServer, RemoteStub

public class UnicastRemoteObject
extends RemoteServer
**********
Stub and Skeleton Compiler
**********
RMI Socket Factories
**********

Java.lang.Object
  └ Java.rmi.server.RemoteObject
    └ Java.rmi.server.RemoteServer
      └ **Java.rmi.server.UnicastRemoteObject**

All Implemented Interfaces:
Remote, Serializable
**********

Java.lang.Object
  └ Java.rmi.server.RemoteObject
    └ **Java.rmi.server.RemoteServer**

All Implemented Interfaces:
Remote, Serializable

Direct Known Subclasses:
Activatable, UnicastRemoteObject

public abstract class RemoteServer extends RemoteObject

### Asiakkaan rajapinnat

Java.lang.Object
  └ **Java.rmi.Naming**

public final class Naming extends Object
**********

Java.lang.Object

└ Java.lang.Throwable
  └ Java.lang.Exception
    └ Java.io.IOException
      └ **Java.rmi.RemoteException**

## Lähdeluettelu


http://www.cs.tut.fi/lintula/manual/Java/tutorial/rmi/overview.html

http://www.cs.tut.fi/~hajap/luennot/12JavaRMI.pdf

http://www.tml.tkk.fi/Studies/Tik-110.300/1999/Essays/corba.html#luku2

http://Java.sun.com/Javase/technologies/core/basic/rmi/whitepaper/index.jsp

http://Java.sun.com/developer/onlineTraining/rmi/RMI.html#IntroRMI

http://www.Javacoffeebreak.com/articles/Javarmi/Javarmi.html

http://keskustelu.plaza.fi/muropaketti/bbs/t377347

http://Java.sun.com/Javase/6/docs/technotes/guides/rmi/codebase.html

http://fin.afterdawn.com/sanasto/termit/



| Tekijät: | Opiskelijanumerot: |
|---|---|
| Paula Mäenpää | 012906529 |
| Janne Sundell | 012686108 |
| Jarmo Mäkäläinen | 012685691 |


## Osallistuminen:

| | |
|---|---|
| Paula Mäenpää | 100% |
| Janne Sundell | 100% |
| Jarmo Mäkäläinen | 100% |

## Käytetyt työtunnit:

| | |
|---|---|
| Paula Mäenpää | n. 34 h |
| Janne Sundell | n. 28 h |
| Jarmo Mäkäläinen | n. 36 h |

## ANALYYSI PROJEKTIN TULOKSISTA

Lähtökohta oli vaikea. Aihe oli tuntematon ja tehtävän laajuus epäselvä. Osaamiseen nähden vaikeusaste oli liiallinen. Samoin myös tehtäväksi anto jäi

roikkumaan ilmaan. Iso pettymys oli, että emme saaneet ulkopuolista ohjausta, sillä opintopiirikin tarvitsisi tukea, pieniä vihjeitä, missä lähtee vikasuuntaan. Kun kolme sokeaa hiirtä taluttaa toisiaan, niin valitettavasti se ei palvele oppimista. Olisimme toivoneet ulkopuolista tukea laskuharjoitusten pitäjältä, mutta eipä sitä ollut.

Kaiken kaikkiaan projekti ei saavuttanut sille asettuja tavoitteita. Erityisesti ohjelmoinnin osuus on erittäin puutteellinen ja sitä mitä on näkyvillä kiittäminen jGurua. Olimme asettaneet tavoitteemme liian korkealle, suhteessa osaamiseemme. Tässä opintopiirissä kukaan ei ollut eikä ole erityisen (java-) koodaushenkinen.

Oli kiva tutustua RMI:hin. Ainakin tietoa on opittu hakemaan, yhdistelemään eri lähteitä ja etsimään olennaisuuksia verkon dis-informaatiosta.

Yksi olennainen asia oli huomata kuinka paljon jo näinkin pieni ryhmä vaatii byrokratiaa asioiden ja tavoitteiden saavuttamiseksi sekä turhan, päällekkäisen työn välttämiseksi.

Suurimpia pettymyksiä oli tiedon jäsentämisen työläys. Useat lähteet tarjosivat saman tiedon, niin että se näytti aluksi erilaiselta.

Opintopiirin toiminnasta:
Teimme osin päällekkäistä työtä, joka johti osin tarpeettomaan ajankulutukseen. Email viestitys aiheutti paljon päällekkäisyyttä. Seuraavalla kerralla täytyy ottaa käyttöön dokumentin hallintaohjelma, ilmaisista verkkojärjestelmistä hyvä on Google -document.

Työtapana tapasimme noin kaksi – kolme kertaa viikossa laskuharjoitusten ulkopuolella. Projektiin on siis käytetty varmasti ohjeaikoja enemmän tunteja. Siksi tavoitteisiin pääsemättömyys harmittaa enemmän.

Mitä opimme?

Paula Mäenpää:
Sen että yli yhden ryhmissä tarvitaan projektipäällikkö, jotta ei tulisi tehtyä ylimääräistä työtä. Nyt on pieni käsitys etä – proseduuririnnakkaisuudesta.

Janne Sundell
Java-ohjelmointia sekä tekemään asioita yhdessä.

Jarmo Mäkäläinen
Kuinka vähän sitä tietääkään. Nöyränä on sinun istuttava Sunin Java sivujen ääressä, ja seurattava tiedon päättymätöntä virtaa. Muilta opin jatkuvan jahkailemisen sijasta tekemään sitä mistä oli sovittu, sekä kärsivällisyyttä. Tunnit eivät aina ole laadun mitta. Takalisto kyllä kestää, mutta kestääkö pää? Harkitsemalla asioita voisi saada samassa ajassa enemmän valmiiksi, joten vastuuta ei tarvitsisi vyöryttää aikapulan takia.

Rinnakkaisuuden eri muotoja, loputtomien väittelyiden tuloksena hyväksymään sen, että etäproseduuri-ohjelmointikin on samanaikaista ja / tai rinnakkaista. Aluksi se ei sopinut

järkeen, mutta kyllä muiden ankaran painostuksen alla ja RCP: speksejä selailemalla, usko etäproseduurien rinnakkaisuuteen oli hyväksyttävä.

**Kysymys 1:**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
- <  task id="TKTS10011" type="singleselection" language="fi" author="Paula Mäenpää,
Jarmo Mäkäläinen, Janne Sundell">

  <course>Rinnakkaisohjelmointi</course>
  <topic>Kertauskysymys</topic>
 <question>Missä seuraavista tilanteista voi tapahtua nälkiintyminen?</question>

    <selection iscorrect="True">
   <answer>Kun algoritmi on reilu.</answer>
   <explanation>Oikein. Busy-wait-semaforilla ei voida taata, että prosessi pääsee kriittiseen
 vaiheeseen – edes kahden prosessin tapauksessa.</explanation>
    </selection>

    <selection iscorrect="False">
    <answer>□¬(p2 ∧ ◊p4) &amp;#9633;&amp;#172;(p2 &amp;#8743; &amp;#9674;p4)</answer>
    <explanation>Väärin. Jokaista vaihetta, jossa p2 on tosi, seuraa vaihe, jossa p4 on
    tosi.</explanation>
   </selection>

    <selection iscorrect="False">
    <answer>Readers-and-writers-ongelmassa </answer>
    <explanation>Väärin. Nälkiintymistä ei esiinny, koska jos blocked-tilassa on kirjoittajia, uusi
    lukija joutuu odottamaan, kunnes ainakin ensimmäinen kirjoittaja lopettaa suorituksensa. Jos taas
    blocked-tilassa on lukijoita, vapautetaan ne kaikki ennen seuraava kirjoittamista.</explanation>
    </selection>

    <selection iscorrect="False">
    <answer>Vahvassa semaforissa</answer>
    <explanation>Väärin. Vahvassa semaforissa S.L, eli blocked-tilassa olevien semaforien määrä
    korvataan jonolla. Tällöin ei voi esiintyä nälkiintymistä. Jono puretaan siinä järjestyksessä, jossa se
    täytettiin (FIFO).</explanation>
    </selection>
 </task>
```

**Kysymys 2:**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
- <  task id="TKTS10011" type="singleselection" language="fi" author="Paula Mäenpää,
Jarmo Mäkäläinen, Janne Sundell">


    <course>Rinnakkaisohjelmointi</course>
    <topic>Kertauskysymys</topic>
   <question>Mikä seuraavista pätee hajautettuihin järjestelmiin?</question>

        <selection iscorrect="True">
       <answer>Kaksi erillistä prosessia (solmua) kommunikoi asymmetrisillä
viesteillä.</answer>
        <explanation>Oikein.</explanation>
        </selection>

        <selection iscorrect="False">
        <answer>Lähettäjä estää (suspended wait) toiminnan, jos vastaanottaja ei voi
vastaanottaa viestiä.</answer>
        <explanation>Väärin. Vastaanottaja tekee eston.</explanation>
       </selection>

        <selection iscorrect="False">
        <answer>Asynkroninen kommunikaatio vaatii puskurin vastaanotetuille
    viesteille.</answer>
        <explanation>Väärin.  Asynkroninen kommunikaatio vaatii puskurin lähetetyille viesteille,
    ei vastaanotetuille (esim. sähköposti).
        </explanation>
        </selection>

        <selection iscorrect="True">
        <answer>Viestien vastaanottojärjestystä ei ole määritelty.</answer>
        <explanation>Oikein.</explanation>
        </selection>
    </task>
```