Java Messaging Service (JMS)

Kaj Karhu, Teemu Laakkonen, Jaakko Nygren, Kalervo Oikarinen, Taina Vartiala

4.12.2008

Helsingin yliopisto

Tietojenkäsittelytieteen laitos

Rinnakkaisohjelmointi, projekti 2

Java Messaging Service (JMS)

Johdanto

Java Messaging Service (JMS) on viestinvälitysmalli, joka mahdollistaa erilaisten ja toisistaan irrallisten osapuolten keskinäisen viestinnän. Siinä sovellukset eivät ole tekemisissä suoraan toistensa kanssa, vaan viestien välitys tapahtuu palvelimen jonon kautta, mikä tekee JMS:stä joustavan, mutta toisaalta palvelimen häiriöille alttiin ja jossain määrin tehottoman viestijärjestelmän.

Tämä dokumentti on syksyn 2008 Rinnakkaisohjelmointikurssin ryhmän "Mikä tahansa" ryhmätyö, joka käsittelee Java Messaging Serviceä. Aloitamme käymällä läpi dokumentissa käyttämäämme terminologiaa, jonka jälkeen esittelemme JMS:n lyhyesti ja pohdimme, minkälaisia kytköksiä aiheellamme on rinnakkaisohjelmointiin. Keskeiset piirteet -kappaleessa käsittelemme JMS:n viestimallia ja arkkitehtonisia piirteitä, sekä toisaalta sen rajoituksia ja toteutusympäristölleen asettamia vaatimuksia. Lisäksi tutustumme hieman tarkemmin Point to point - ja julkaise-tilaa-järjestelmiin.
Lopuksi annamme vielä kaksi Java Messaging Servicen käyttöesimerkkiä. Hei maailma! -esimerkissä pyrimme avaamaan viestimallin käyttölogiikkaa yksinkertaisen koodiesimerkin avulla. Seti@Home taas on laajemman mittakaavan esimerkki JMS:n tarjoamista monipuolisista mahdollisuuksista.

Terminologia

Alla on kuvattu tässä dokumentissa käytettävä termistö sekä lyhenteet. Termistöstä löytyvät myös suomenkielisten sanojen alkuperäiset englanninkieliset vastikkeet, ja olemme ottaneet vapauden tässä dokumentissa antaa niille suomenkieliset vastineet.

Termi	**Selite**
Asiakas (client)	Viestirajapinnan käyttäjät.
JMS-sovellus	Sovellus koostuu määritetyistä viesteistä sekä joukosta asiakkaita, jotka käyttävät sovellusta.
Viesti	Pienin yksikkö, jonka avulla asiakkaat

	kommunikoivat viestijärjestelmän kanssa
Point to Point (P2P)	JMS:n määrittelemä viestimalli, joka perustuu viestijonoihin.
Julkaise-tilaa-järjestelmä (Publish-subscribe Model)	JMS:n määrittelemä viestimalli, jossa voi olla useita viestin vastaanottajia.
JMS-tarjoaja (JMS Provider)	JMS-tarjoaja tarjoaa toteutuksen viestijärjestelmälle, jonka kanssa JMS-sovellukset kommunikoivat.
Julkaisija (publisher)	JMS Pub-Sub -mallissa oleva Publisher-osapuoli
Tilaaja (subscriber)	JMS Pub-Sub -mallissa oleva Subscriber-osapuoli
Pub-Sub	ks. Julkaise-tilaa-järjestelmä.
Viestityyppien säilö	Message type repository
Siirtoprotokolla	Wire-protocol
Turvallisuus	security
Hallinta	Administration
Virhetapausten viestitys	Error / Advisory notification
Kuorman tasaus / virheensietokyky	Load Balancing / Fault tolerance
Objekti	Tässä yhteydessä objektia käytetään luokan instanssien / ilmentymien synonyyminä.
Kuluttaa (consume)	Tarkoittaa, että asiakas vastaanottaa viestin. Huom suuntana on asiakas -> järjestelmä.
Tuottaa (produce)	Käytetään yleisenä terminä viestin lähettämiselle.
Välittää (deliver)	Viestijonojärjestelmän viestin lähetys siinä tapauksessa, että kyseessä on pub/sub-järjestelmä. Huom. suuntana on järjestelmä -> asiakas.
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface

Taulukko 1: Terminologia

Mikä Java Messaging Service (JMS) on?

Java Messaging Service eli lyhyesti JMS on viestirajapinta, joka on toteutettu Sun Microsystemsin kehittämällä Java-kielellä. JMS-sovelluskehyksen käyttökohteita ovat esimerkiksi yritysten järjestelmäintegraatiot, joissa osajärjestelmien välinen kommunikointi hoidetaan viestien avulla.

JMS voidaan ymmärtää sovelluskehyksenä, sillä siinä on valmiina joukko rajapintoja sekä valmiita osatoteutuksia, jotka sisältävät semantiikan siitä millä tavoin asiakassovellus (client application) kommunikoi viestijärjestelmän kanssa. JMS pyrkii maksimoimaan viestisovellusten siirrettävyyden. Siirrettävyyden yhtenä keskeisenä osana on Java-kieli. JMS:n keskeisimmät kommunikoinnin käsitteet ovat Point to Point (P2P) - sekä Publisher Subscriber (Pub/Sub) - mallit. Nämä mallit käsitellään omissa osioissaan.

Kuinka tämä liittyy rinnakkaisohjelmointiin?

Hajautettu järjestelmä on rinnakkaisohjelma, jossa laskenta on jaettu useille fyysisesti erillisille tietokoneille, joilla ei ole yhteistä jaettua muistia tai prosessoria, vaan joiden välille on rakennettu viestirajapinta. JMS:ä voidaan käyttää toteutettaessa hajautettu järjestelmä Javalla. JMSClient on hajautetun järjestelmän solmu ja JMSProvider toteuttaa viestirajapinnan.

Fyysisesti useaan tietokoneeseen hajautetussa järjestelmässä yhden solmun resurssien jakaminen ja konekäskyjen lomittaminen hallitaan samalla tavalla kuin yhdessä tietokoneessa suoritettavassa rinnakkaisohjelmassa. Tämän lisäksi hajautetussa järjestelmässä tulee kuitenkin hallita solmujen välillä tapahtuva viestien synkronointi. Koska prosesseilla ei ole yhteisiä jaettuja resursseja, lomittaminen perustuu viestien lähettämiseen ja vastaanottamiseen ja solmujen hallintaan niiden diskreettien tilojen avulla. Hajautetussa järjestelmässä jokainen solmu voi olla ainoastaan kolmessa eri tilassa: Suorittamassa omaa laskutehtäväänsä tai lähettämässä tai vastaanottamassa viestiä. Tämän merkittävän eron vuoksi hajautetut järjestelmät eivät ole toteutukseltaan verrannollisia muiden rinnakkaisohjelmien kanssa.

Koska yhteistä nopeaa muistia tai rekistereitä ei ole, vaan solmujen välinen keskustelu tapahtuu hitaammilla viesteillä, toteutuksessa joudutaan punnitsemaan viestirajapinnan kustannuksia suhteessa sen tehokkuuteen. Koska solmujen määrä voi kasvaa hyvin suureksi, tarvittavien viestikanavien määrä kasvaa toteutuksesta riippuen hyvin nopeasti. JMS pyrkii ratkaisemaan ongelman tarjoamalla JMSProvider-viestirajapinnan, jonka kautta kaikki viestit lähetetään ja vastaanotetaan. Tällä tavalla saadaan pudotettua tarvittavien viestikanavien määrää huomattavasti verrattuna siihen, että kaikki solmut olisivat yhteydessä suoraan toisiinsa. Ratkaisu ei ole välttämättä tehokkain kaikkiin tilanteisiin, mutta tarjoaa myös selkeän jaon viestirajapinnan ja solmujen välillä.

Hajautettujen järjestelmien toinen merkittävä ero suhteessa muihin rinnakkaisohjelmiin on hajautetun järjestelmän viansietokyky. Kun yhdessä tietokoneessa toimivan rinnakkaisohjelman

laitevika saattaa estää kaikkien prosessien toiminnan ja olla fataali ohjelman suoritukselle, voidaan hajautetusti toteutetussa järjestelmässä usein todeta solmun vioittuminen ja toteuttaa viallisen solmun tehtävät jollakin toisella solmulla, tai ohittaa solmu viestiketjussa. JMS:llä voidaan laskenta toteuttaa jaetusti usealle solmulle, joten vioittuneen solmun virheet havaitaan ja voidaan korvata toisten solmujen laskennan tuloksilla. JMS:n kaltaisessa toteutuksessa kuitenkin koko hajautettu järjestelmä on riippuvainen JMSProviderin toiminnasta. Yhden JMSProviderin järjestelmässä, sen vioittuessa koko järjestelmän viestinvälitys lakkaa toimimasta. Viansietokyky on siten huomattavasti matalammalla tasolla kuin esim. internetprotokollien toteutuksessa, jossa uusi reitti voidaan yhden reitittimen hajotessa tai ruuhkautuessa ottaa vaivatta käyttöön.
(Lähde [4])

Keskeiset piirteet

Tässä kappaleessa käsitellään Java Messaging Servicen keskeisiä piirteitä ja käsitteitä, sekä sekä luodaan katsaus arkkitehtuuriin ja JMS:n määrittämiin viestimalleihin. Tarkastelu on melko pintapuolista, ja tarkempaa tietoa löytyykin JMS-spesifikaatiosta.

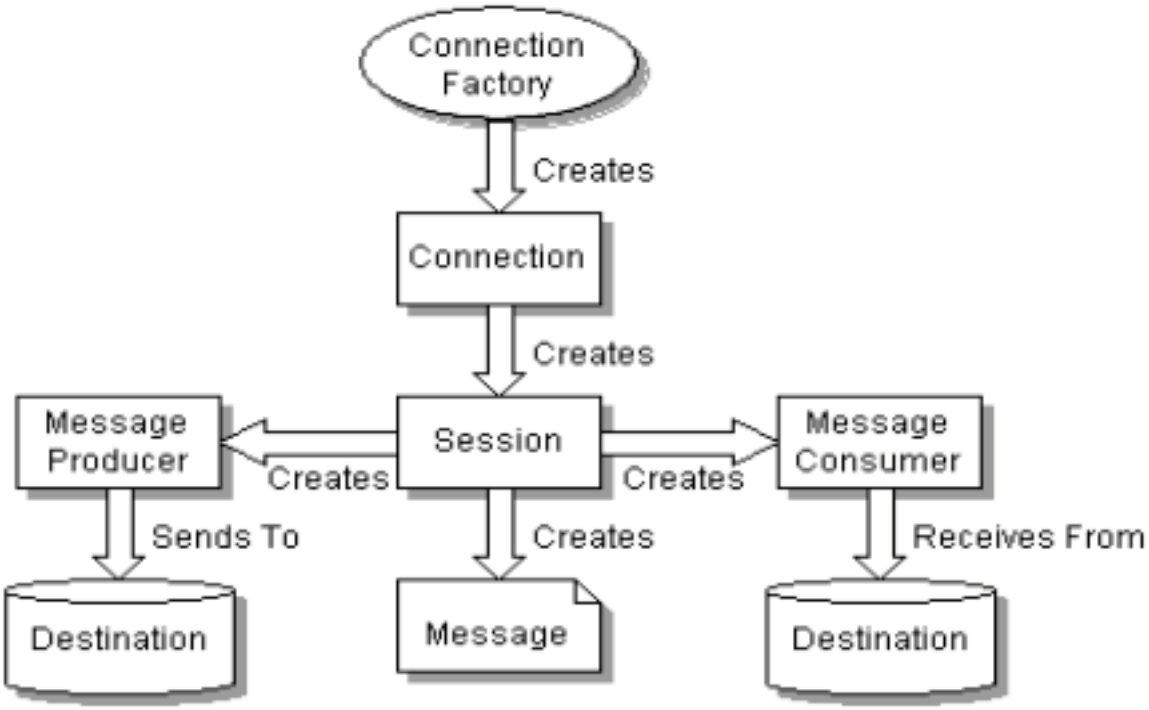
Arkkitehtuuri

JMS on joukko ennaltamääritettyjä rajapintoja, joiden avulla abstrahoidaan viestijärjestelmä asiakassovellusten kannalta ja täten vähennetään asiakassovelluksen riippuvuutta tietystä viestijärjestelmästä. JMS:n yleiset rajapinnat ovat seuraavat (JMS-mallit sisältävät omat variantit näistä rajapinnoista, mutta niiden semantiikka on näiden rajapintojen kaltainen): ConnectionFactory, Connection, Destination, Session, MessageProducer sekä MessageConsumer (ks. kuva 1).

JMS:ssä on kaksi erilaista mallia kommunikointiin: *point to point* - ja *julkaise-tilaa-järjestelmä*. JMS tarjoaa rajapinnat molempien mallien kanssa kommunikointiin. JMS:n keskeisimmät käsitteelliset komponentit ovat: JMS-asiakkaat (JMS clients), ei-JMS-asiakkaat (non-JMS clients), viestit (Messages), JMS-tarjoaja (JMS Provider) ja hallitut objektit (Administered objects).

JMS-asiakkaat ovat java-ohjelmointikielellä toteutettuja sovelluksia, jotka toimivat itsenäisesti ja muiden kuin viestien osalta mahdollisesti riippumattomina JMS-tarjoajasta. JMS:n kannalta asiakkaat vain vastaanottavat ja lähettävät viestejä. Ei-JMS-sovellukset ovat sellaisia sovelluksia, jotka hyödyntävät viestijärjestelmän natiivia rajapintaa kommunikointiin JMS:n

sijaan. Viestit ovat JMS:n kommunikoinnin peruskomponentti. Jokainen JMS-sovellus määrittelee itse viestit, joiden avulla viestijärjestelmä kommunikoi asiakkaan kanssa. JMS-tarjoaja on täydellinen viestijärjestelmän toteutus. JMS itsessään on pelkkä spesifikaatio ja kehys viestirajapinnasta. Tarjoaja on sellainen järjestelmä, joka toteuttaa itsenäisen sovelluksen vaatimat piirteet sekä tarjoaa toteutuksen JMS:lle. Hallitut objektit ovat sellaisia objekteja jotka on ennalta määritetty viestijärjestelmän toteutuksen puitteissa. JMS:ssä on olemassa kahdenlaisia JMS:n ennaltamäärittämiä komponentteja: Yhteystehdas (connection factory), jonka tehtävänä on luoda yhteys asiakkaan ja tarjoajan välille sekä kohde (destination), joka määrittää päätepisteen, jonne asiakassovellus lähettää viestinsä. Nämä hallitut objektit sijoitetaan JNDI (*Java Naming and Directory Interface*) -avaruuteen, josta asiakkaat voivat hakea ne.



Kuva 1: JMS:n keskeiset rajapinnat ja niiden roolit (lähde [1]).

JMS-viestimalli

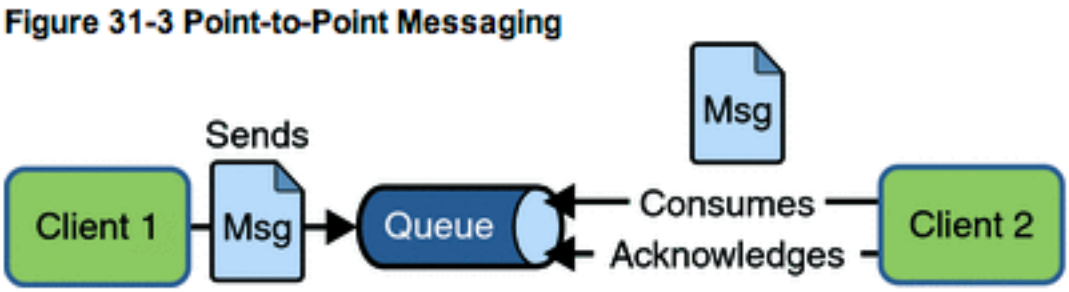
JMS:ssä viesteillä on seuraavat osat: otsake (header), ominaisuudet (properties) sekä runko (body). Otsaketiedot pitävät sisällään tietoja joita käytetään viestien tunnistamiseen ja reitittämiseen. Ominaisuudet sisältävät esimerkiksi sovelluskohtaisia lisäominaisuuksia tai lisäotsaketietoja. JMS määrittää muutamia erilaisia standardeja runkotyyppejä, jotka kattavat suurimman osan käyttötapauksista. JMS:ssä on seuraavat viestirungot: StreamMessage, MapMessage, TextMessage, ObjectMessage sekä BytesMessage.

Point to Point -järjestelmä

Point to Point (P2P) -järjestelmä on rakennettu viestijonojen käsitteen päälle [1]. Tässä järjestelmässä viestit sijoitetaan jonoihin, joita asiakkaat purkavat. Tällöin asiakkaan purkama viesti poistuu viestijonosta. Jokaiselle asiakkaalle on järjestelmässä olemassa oma spesifi jononsa. Yleisesti yksittäinen asiakas lähettää viestinsä yhteen jonoon (ks. kuva 2).
PTP-mallin keskeinen tehtävä on määrittää kuinka asiakassovellus käyttäytyy jonojen kanssa. Mallissa määritetään millä tavoin asiakas lähettää, vastaanottaa sekä löytää viestit jonoista.

Mallin keskeiset rajapinnat (ks. myös kuva 1):

- QueueConnectionFactory - hoitaa yhteyden
- QueueConnection - Aktiivinen yhteys jonoon
- Queue - Jono
- QueueSession - Tarjoaa keskeisiä tehdasmetodeja eri objektien toteutukseen
- QueueSender - Viestienlähetysmekanismi
- QueueReceiver - Viestin vastaanottomekanismi



Kuva 2: Point to point käsite (Lähde [2])

Julkaise-tilaa-järjestelmä (Publish-Subscribe)

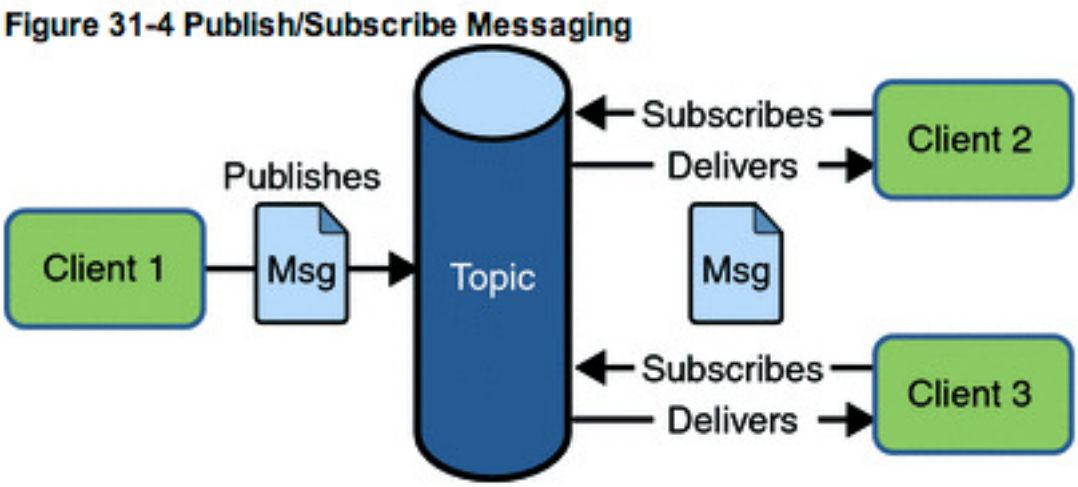
Julkaise-tilaa (publish/Subscribe) -viestijärjestelmässä yksi lähettäjä voi lähettää viestejä usealle vastaanottajalle. Sitä voi verrata anonyymiin ilmoitustauluun, jolta kiinnostuneet käyvät lukemassa viestejä. Järjestelmässä lähettäjä lähettää viestin jostain aiheesta (Topic) palvelun tarjoajalle, joka jakaa sen kaikille kyseisen aiheen tilaajille. JMS itsessään ei säilytä viestejä niiden jakamisen jälkeen. Julkaisija luo ensin aiheen, jota muut asiakasohjelmat voivat sen jälkeen alkaa tilata. Tilaajat saavat vai ne viestit, jotka on lähetetty tilauksen jälkeen.

Mallin keskeiset rajapinnat:

- TopicConnectionFactory - Hoitaa yhteyden

- TopicConnection - Aktiivinen yhteys palvelun tarjoajaan
- TopicSession - Tarjoaa metodeja objektien toteutukseen
- TopicPublisher – Asiakas, joka lähettää viestejä johonkin aiheeseen
- TopicSubschiber - Asiakas, joka ottaa vastaan viestejä jostakin aiheesta
- Durable TopicSubscriber - Asiakas, joka ottaa vastaan kaikki viestit jostakin aiheesta

Tavallisen ja jatkuvan (durable) tilauksen ero on, että tavallisessa asiakas ei saa niitä viestejä, jotka on lähettetty tämän ollessa passiivinen, kun taas jatkuva tilaaja saa kaikki aiheeseen lähetetyt viestit.



Kuva 3: Publish/Subscribe Messaging (Lähde [2])

Rajaukset

JMS:ssä on seuraavat rajaukset, joihin JMS kehys ei ota lainkaan kantaa eikä tarjoa valmiita toteutuksia:

- Kuorman tasaus / virheensietokyky
- Virhetapausten viestitys asiakkaille / palvelimelle
- Hallinta - JMS ei tarjoa valmista API:a JMS-järjestelmien hallintaliittymän toteuttamiseen
- Turvallisuus - JMS ei ota kantaa viestin kryptaukseen tai kommunikaatioväylän salaukseen
- Siirtoprotokolla - JMS ei määrittele kiinteää protokollaa, jonka avulla JMS-viestejä siirretään (täten käy esim. HTTP tai UDP)
- Viestityyppien säilö - JMS ei määrittele mitään säilöä viestityyppien määrittelyjen tallettamiseen eikä myöskään kieltä viestien määrittämiseen.

Vaatimukset

JMS asettaa myös tiettyjä vaatimuksia toteutusympäristölle. Java-kielestä johtuen JMS vaatii ympäristöltä, että siinä on Java-suoritusympäristö käytettävissä. Yleensä tehtäessä JMS:llä liittyy siihen myös joukko muita J2EE-teknologioita (esim. JDBC, JNDI, Java Beans, J2EE platform jne.). Nämä lisäteknologiat asettavat järjestelmälle myös omat vaatimuksensa, jotka järjestelmän tulee täyttää.

Käyttöohje

Tärkeimmät rajapinnan osat

Käytössä tärkeimmät rajapinnan osat määrätyvät valitun viestimallin mukaan. Keskeisimmät rajapinnat löytyvät mallia käsittelevästä kappaleesta sekä arkkitehtuurikappaleesta, jossa käsitellään JMS:n yleisiä arkkitehtonisia piirteitä.

1. Käyttöesimerkki - Hei Maailma!

LahetetaanHeiMaailma.java - Yksinkertainen 'Hei, maailma!' sovellus,
jossa demonstroidaan viestin lähettämistä nimettyyn viestijonoon JMS:n avulla.

```
import javax.jms.*
import java.util.*

// lähettää viestin JMS-serverin kanavalle 'jono'
public class LahetetaanHeiMaailma {
  public static void main (String[] args) {
    try {
      ConnectionFactory omaConnectionFactory;
      Queue omaQueue;

      // haetaan yhteystehdas ja kohde
      omaConnectionFactory = new com.sun.messaging.ConnectionFactory();

      // luodaan yhteys
      Connection omaConnection = omaConnectionFactory.createConnection();

      // luodaan sessio
      Session omaSession = omaConnection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);
```

```
      // valitaan kanava
      omaQueue = new.com.sun.messaging.Queue(“jono”);

      // alustetaan viestien lähettäjä
      MessageProducer omaMessageProducer=omaSession.createProducer(omaQueue);

      // luodaan viesti
      TextMessage omaTextMessage = omaSession.createTextMessage();
      omaTextMessage.setText(“Hei maailma!”);

      //lähetetään viesti
      omaMessageProducer.send(omaTextMessage);

      // suljetaan sessio
      omaSession.close();

      // suljetaan yhteys
      omaConnection.close();
    } catch (Exception jmse) {
    System.out.println(jmse.toString());
  }
}
```

LuetaanHeiMaailma.java - Yksinkertainen sovellus, jolla demonstroidaan edellisen sovelluksen lähettämän viestin lukemista viestijonosta JMS:n avulla.

```
import javax.jms.*
import java.util.*

// lukee viestin JMS-serverin kanavalta 'jono'
public class LuetaanHeiMaailma {
  public static void main (String[] args) {
    try {
      ConnectionFactory omaConnectionFactory;
      Queue omaQueue;

      // haetaan yhteystehdas ja kohde
      omaConnectionFactory = new com.sun.messaging.ConnectionFactory();

      // luodaan yhteys
```

```
      Connection omaConnection = omaConnectionFactory.createConnection();

      // luodaan sessio
      Session omaSession = omaConnection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

      // valitaan kanava
      omaQueue = new.com.sun.messaging.Queue(“jono”);

      // alustetaan viestien lukija
      MessageConsumer omaMessageConsumer=omaSession.createConsumer(omaQueue);

      // avataan yhteys
      omaConnection.start();

      // luetaan viesti
      Message message = omaMessageConsumer.receive();

      // tulostetaan viesti
      if (message instanceof TextMessage) {
        TextMessage textMessage = (TextMessage) message;
        System.out.println(textMessage.getText());
      }

      // suljetaan sessio
      omaSession.close();

      // suljetaan yhteys
      omaConnection.close();
    } catch (Exception jmse) {
    System.out.println(jmse.toString());
  }
}
```

2. Käyttöesimerkki - Seti@Home

SETI@home (Search for Extra-Terrestrial Intelligence at home) on tieteellinen hajautetun laskennan projekti, jossa etsitään Maapallon ulkopuolista älyllistä elämää. Projektiin osallistuvat vapaaehtoiset lataavat tietokoneelleen ohjelman, joka analysoi radioteleskoopin keräämää dataa etsien säännöllisiä kapeakaistaisia radiosignaaleja. Teleskoopilta postitse lähetetyt

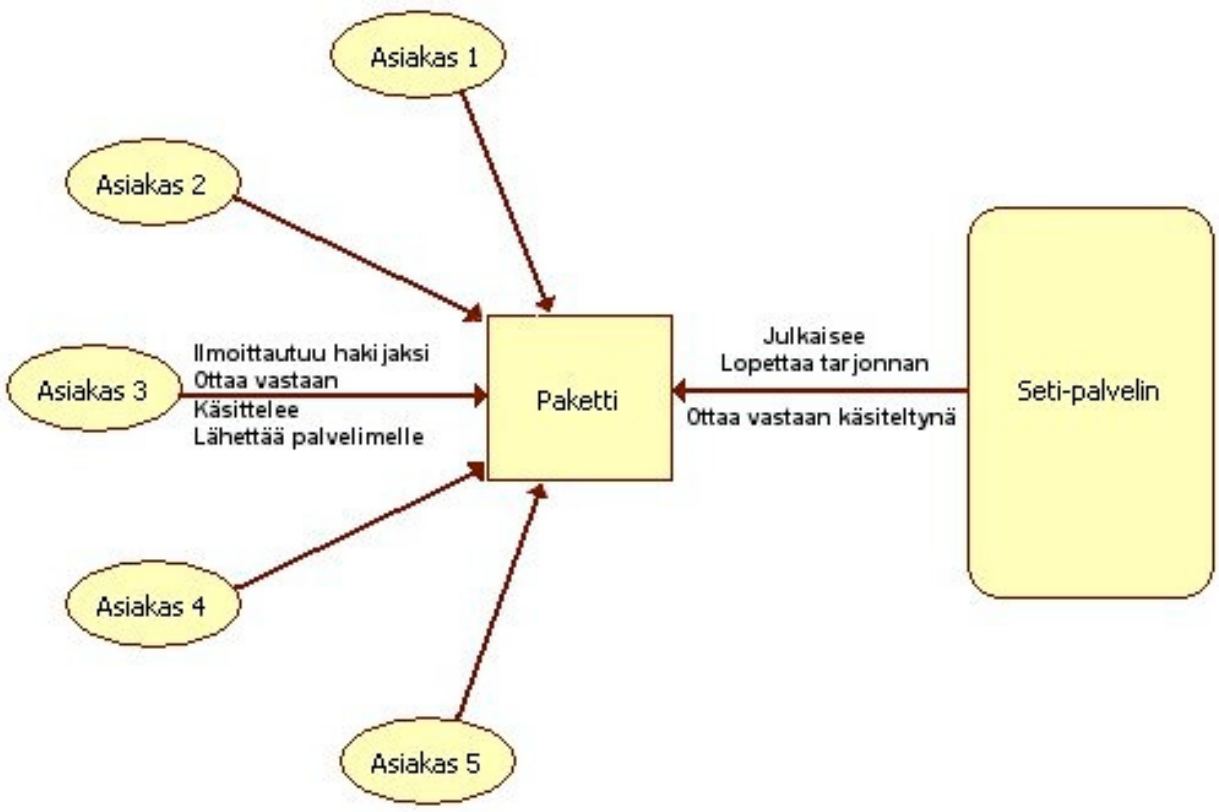
datanauhat pilkotaan Berkeleyssä kestoltaan ja taajuusalueeltaan pienemmiksi työpaketeiksi, joita lähetetään internetin välityksellä vapaaehtoisille analysoitavaksi. Alla esimerkit kuinka Seti-palvelin ja -asiakas voitaisiin toteuttaa JMS:n avulla.

Seti-palvelin

Palvelin julkaisee seuraavan työpaketin. Viisi ensimmäistä aktiivista paketin hakijaksi ilmoittautunutta asiakasta saavat luvan pakettiin. Nämä viisi asiakasta hakevat paketin itselleen. Palvelin lopettaa kyseisen paketin tarjoamisen ja julkaisee uuden paketin, jonka saavat taas viisi ensimmäistä asiakasta. Sama paketti annetaan viidelle eri asiakkaalle, jotta virheelliset ja peukaloidut tulokset pystytään eliminoimaan.

Seti-asiakas

Asiakas ilmoittautuu paketin hakijaksi ja hakee paketin, kun palvelin on myöntänyt sille luvan siihen. Asiakas suorittaa paketin datan analysoinnin ja lähettää käsitellyn paketin suoraan seti-serverille p2p-yhteydellä. Tämän jälkeen asiakas ilmoittautuu taas uuden paketin hakijaksi.



Lähdeluettelo

[1] - JMS Spesifikaatio vuodelta 2002 (http://java.sun.com/products/jms/)

[2] - http://java.sun.com/javaee/5/docs/tutorial/doc/bncdx.html

[3] – Sun Microsystemsin viestijärjestelmä (ilmeisesti ilmainen): https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_SMI-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=JavaSystem-MsgQ3-2005Q1-OTH-G-F@CDS-CDS_SMI

[4] - M. Ben Ari: Principles of Concurrent and Distributed Programming (Addison-Wesley, 2006 p.16)