# Nippu-ohjelmistotuotantoprojekti
# Toteutusdokumentti

Michael Forsström, Mikael Jokela, Ville V Nurmi, Ville A Nuutinen, Chen Zhao

Helsinki 31.8.2003

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

# Versiohistoria

| Versio | Päiväys | Koostaja | Kommentti |
|---|---|---|---|
| 0.9 | 29.8.2003 | Mikael Jokela | |
| 1.0 | 31.8.2003 | Mikael Jokela | lopullinen versio |

# Sisältö

# 1 Johdanto

Tämä dokumentti sisältää **teknisen mallin Nippu-tulostuspalvelinohjelmistosta** sekä toteutusvaiheessa tehdyt tarkennukset ja tekniset ratkaisut. Ohjelmointitekniset seikat – kuten tietorakenteet[1] ja liittymät – kuvataan tässä dokumentissa myöhempää ylläpitoa ajatellen riittävällä tarkkuudella.

**Toteutukseen liittyvät rajoitukset** esitellään luvussa 2. Toteutuksen kuvaus jakautuu **yleiskuvaukseen** (luku 3) ja **komponenttien toteutuksen kuvaukseen** (luku 4). Yleiskuvaus sisältää tietokuvauksen, arkkitehtuurikuvauksen, liittymäkuvauksen ja toiminnekuvauksen. Komponenttien toteutuksen kuvauksessa puolestaan käsitellään erikseen kaikki keskeiset ohjelmistokomponentit ja esitellään niiden tehtävät, liittymät sekä toiminta pseudokoodin avulla. Lopuksi lukuun 5 on koottu myöhemmän ylläpidon ja kehityksen kannalta oleellisia huomautuksia.

Tämä dokumentti perustuu suoraan suunnitteludokumentin, jonka sisältö on päivitetty lopullista toteutusta vastaavaksi. Myöhempää ylläpitoa ja kehitystä varten tähän dokumenttiin on kirjattu vaihtoehtoisia toteutustapoja sekä tässä vaiheessa toteuttamatta jääneitä ideoita

Nipun keskeisin tarkoitus on **saavuttaa tehokkuutta suorittamalla samankaltaisia palvelupyyntöjä yhdessä**. Käytännössä ohjelmisto pyrkii esimerkiksi ryhmittelemään lähekkäin sijaitsevien tulostajien työt siten, että ne tulostuvat yhdessä nipussa jollain läheisellä, vapaalla tulostimella. Nippu on osa Helsingin yliopiston tietojenkäsittelytieteen laitoksen ohjelmistotuotantoprojekti-kurssia. Nippua on tarkoitus käyttää Helsingin yliopistossa koostavan ajoitusmenettelyn ja sen kannattavuuden tutkimisessa.

Liitteen 5 sanastossa selitetään tässä dokumentissa esiintyviä käsitteitä.

1 Tässä dokumentissa tietorakenne-käsite tarkoittaa yleisesti tiettyä tiedon esitystapaa eikä niinkään mitään tarkasti määriteltyä korkean profiilin tietorakennetta. Näin esimerkiksi Javan luokkia voidaan kutsua tietorakenteiksi. Tällainen termin käyttötapa on valittu dokumentointiteknisistä syistä.

# 2 Toteutusrajoitukset

## 2.1 Ohjelmointikieli ja toteutusympäristö

Nipun **ohjelmointikieli on Java** (Java 2 standard edition versio 1.4.1). Ulkoiset rajapinnat ovat Javan mukaisia. Toteutusympäristö on Linux-käyttöjärjestelmä.

## 2.2 Ohjelmointitavat ja -standardit

Ohjelmoinnissa sisäiset nimeämiset ja kommentointi on tehty englannin kielellä ja Sunin Java-koodaussuosituksen[2] mukaan. Tavanomaisten kommenttien lisäksi on tehty Javadoc-kommentointi. Valmiista ohjelmakoodista automaattisesti luotu Javadoc-dokumentti on liitteessä 6. Ohjelmointitapoja ja -standardeja käsitellään tarkemmin ryhmän sisäisessä ohjelmointiohjeessa, joka on liitteessä 1.

## 2.3 Tarvittavat ohjelmat ja kirjastot

Nippu vaatii **Terttu-ohjelmistosta** juuri sen version, joka toimitetaan samassa paketissa Nipun kanssa. Tertun lähdekoodiin on nimittäin jouduttu tekemään muutoksia, joista kerrotaan tarkemmin liittymäkuvauksen luvun 3.3.3 lopussa. Tertun kokoonpano täytyy olla sellainen, että fuusiointiparametri on päällä ja ajoittajan avaamat TCP-portit ovat 7700 (ulkoisille asiakkaille) ja 7701 (sisäisille asiakkaille). Nipun mukana toimitetaan valmis Tertun asetustiedosto, joka sisältää edellä mainitut ja muut tarvittavat asetukset.

Nipun mukana toimitetaan myös ulkoisia ohjelmakomponentteja, jotka ovat välttämättömiä Nipun toiminnalle. Esimerkiksi ajonaikaisten viestien kirjaamiseen käytetään Log4j-ohjelmaa[3] ja käyttöliittymä käyttää Jbuilder-ohjelmiston luokkakirjastoja.

Nipun ajamiseen tarvitaan **Java-virtuaalikone**, joka on yhteensopiva Java 2 standard edition 1.4.1:n kanssa. Java-virtuaalikonetta ei toimiteta Nipun mukana.

2 Code Conventions for the Java Programming Language, katso verkko-osoite http://java.sun.com/docs/codeconv/.

3 Tämän dokumentin oikeinkirjoituksessa noudatetaan johdonmukaisesti suomen kielen mukaisia käytänteitä, koska dokumentin kieli on suomi. Tämä tarkoittaa muun muassa sitä, että kaikkien nimien, esimerkiksi Log4j ja Jbuilder, kirjoitusasu ei vastaa alkuperäistä kirjoitusasua. Opetusministeriön alaisen Kotimaisten kielten tutkimuskeskuksen kielitoimiston suomen kielen lautakunnan suosituksen mukaan näin voidaan menetellä.

# 3 Yleiskuvaus

Tässä luvussa kuvataan ohjelmiston yleistä arkkitehtuuria (arkkitehtuurikuvaus). Lisäksi esitellään tarkasti tärkeimmät tietorakenteet (tietokuvaus) ja liittymät (liittymäkuvaus) sekä tarkastellaan ohjelmistokomponenttien yhteistyötä järjestelmän toiminnallisuuden toteutumisessa (toiminnekuvaus).

## 3.1 Tietokuvaus

Nipun keskeisimpiä tietorakenteita ovat tulostustyötä kuvaava **PrintJob-luokka**, tulostustyönippua kuvaava **Bundle-luokka**, käyttäjää kuvaava **NippuUser-luokka**, tulostinta kuvaava **NippuPrinter-luokka**, asetustiedoston käsittelyssä käytettävä **Configuration-Parser-luokka** sekä **asetustiedostot**. Asetustiedostojen määrittely on osa tietokuvausta, mutta ne ovat myös ulkoisia liittymiä, joten niihin perehdytään tarkemmin vasta luvussa 3.3.

Tietokuvauksessa kaikkien kokonaislukujen enimmäisarvo on vakio Integer.MAX_ VALUE, joka on Javan enimmäisarvo kokonaisluvulle. Vakion arvo on 2 147 483 647. Kaikissa merkkijonoissa sallittuja merkkejä ovat vain numerot (0–9) ja pienet kirjaimet (a–z). Skandinaavisia kirjainmerkkejä (å, ä, ö) ei sallita. Kaikkien merkkijonojen vähimmäispituus on kaksi merkkiä. Poikkeus on tulostustyön nimi -merkkijono, jossa sallitaan kaikki muut merkit paitsi pilkku ja %-merkki ja jonka pituus tulee olla vähintään yksi merkki.

### 3.1.1 PrintJob-luokka

PrintJob-luokka on tulostustyön toteutus. Tulostustyö toteutetaan luokan ilmentymänä eli oliona. PrintJob-olio sisältää kaikki tiedot, joita tulostustyöhön halutaan liittää. Tiedot määritellään private-kenttinä.

Luokka sisältää **seuraavat kentät**:

private String jobName
: tulostustyön nimi, enintään 255 merkkiä

private int jobID[4]

tulostustyön tunnus, ei-negatiivinen kokonaisluku

private String user

tulostajan käyttäjätunnus, enintään 8 merkkiä

private int x

tulostajan x-koordinaatti, ei-negatiivinen kokonaisluku

private int y

tulostajan y-koordinaatti, ei-negatiivinen kokonaisluku

private int z

tulostajan z-koordinaatti, ei-negatiivinen kokonaisluku

private int numberOfPages

tulostustyön sivumäärä, aidosti positiivinen kokonaisluku

private String forcedPrinter

mahdollinen pakotettu tulostin, enintään 10 merkkiä

private int senderUI

työn lähettäneen käyttöliittymän tunnus, ei-negatiivinen kokonaisluku

Luokka sisältää kaksi **konstruktoria**:

public PrintJob()

Tämä on peruskonstruktori, joka on tarkoitettu testaukseen.

public PrintJob(String jobName, int jobID, String user, int x, int y, int z,
int numberOfPages, String forcedPrinter, int senderUI)

Konstruktori luo halutunlaisen tulostustyön. Parametrien rajoitteet ovat samat kuin edellä luetelluilla luokan kentillä.

Luokka sisältää seuraavat **get-metodit**, jotka palauttavat vastaavan kentän arvon:

public String getJobName()
public int getJobID()
public String getUser()
public int getX()
public int getY()
public int getZ()

4 JobID-kentästä kerrotaan tarkemmin komponenttien toteutuksen kuvauksessa luvussa 4.1.4.

public int getNumberOfPages()
public String getForcedPrinter()
public int getSenderUI()

Luokka sisältää seuraavat **set-metodit**, joilla voi asettaa kenttien arvot. Metodit ovat tarkoituksella julkisia, koska useiden kenttien muuttaminen olion luomisen jälkeen voi olla tarpeellista. Yhdenmukaisuuden vuoksi kaikki metodit on sitten määritelty julkisiksi. Metodeja tulee kuitenkin käyttää tarkasti harkiten, koska niillä saa helposti aikaan Nipun logiikan vastaisia PrintJob-ilmentymiä. Ohjelmiston sisäinen tietoturva puoltaisi metodien asettamista yksityisiksi, mutta tämä ratkaisu voisi rajoittaa ohjelmiston myöhempää kehitystä.

public boolean setJobName(String name)
: Metodi palauttaa true, jos asettaminen onnistui, muuten false. Parametrina annettava merkkijono *name* voi olla enintään 255 merkkiä pitkä.

public int setJobID(int id)
: Metodi palauttaa asetetun kokonaisluvun, jos asettaminen onnistui, muuten palautetaan -1. Parametrina annettava *id* on ei-negatiivinen kokonaisluku.

public boolean setUser(String user)
: Metodi palauttaa true, jos asettaminen onnistui, muuten false. Parametrina annettava merkkijono *user* voi olla enintään 8 merkkiä pitkä.

public int setX(int x)
: Metodi palauttaa asetetun x:n arvon, jos asettaminen onnistui, muuten -1. Parametrin on oltava ei-negatiivinen kokonaisluku.

public int setY(int y)
: Metodi palauttaa asetetun y:n arvon, jos asettaminen onnistui, muuten -1. Parametrin on oltava ei-negatiivinen kokonaisluku.

public int setZ(int z)
: Metodi palauttaa asetetun z:n arvon, jos asettaminen onnistui, muuten -1. Parametrin on oltava ei-negatiivinen kokonaisluku.

public int setNumberOfPages(int pages)
: Metodi palauttaa asetetun sivumäärän arvon, jos asettaminen onnistui, muuten -1. Parametrin on oltava aidosti positiivinen kokonaisluku.

public boolean setForcedPrinter(String printer)

Metodi palauttaa true, jos asettaminen onnistui, muuten false. Parametri *printer* voi olla enintään 10 merkkiä pitkä.

public int setSenderUI(int ui)

Metodi palauttaa asettamansa ui:n arvon, jos asettaminen onnistui, muuten palautetaan -1. Parametrin on oltava ei-negatiivinen kokonaisluku.

Luokka sisältää lisäksi seuraavat metodit:

public boolean equals(Object other)

Metodin avulla voi vertailla kahta PrintJob-oliota. Metodi vertaa olioiden jokaista kenttää erikseen. Jos kaikki kentät ovat samanlaisia, oliot ovat samanlaiset. Metodi palauttaa true, jos oliot ovat samanlaiset, muuten false. Parametrin *other* tyyppi on oltava PrintJob, muuten palautetaan false.

public String serialize()

Metodi palauttaa merkkijonoesityksen luokan oliosta. Merkkijonossa olion kenttien arvot on erotettu toisistaan %-merkillä. Merkkijono ei ole tarkoitettu käyttäjälle vaan esimerkiksi toiselle oliolle tai ohjelmalle. Metodin palauttama merkkijono on siis muotoa: *jobName%jobID%user%x%y%z% numberOfPages%forcedPrinter%senderUI*.

public static PrintJob deserialize(String serializedPrintJob)

Staattinen metodi saa parametrina merkkijonon, joka vastaa *serialize*-metodin tuottamaa merkkijonoa. Tästä merkkijonosta metodi luo ja palauttaa sitä vastaavan PrintJob-olion.

public String toString()

Metodi palauttaa merkkijonoesityksen luokan oliosta. Merkkijonoesitys on tarkoitettu käyttäjälle esimerkiksi testaukseen.

public boolean hasForcedPrinter()

Metodi palauttaa true, jos PrintJobilla on pakotettu tulostin. Muutoin palautetaan false.

public static void main(String[] args)

pääohjelmametodi testausta varten

### 3.1.2 Bundle-luokka

Bundle-luokka on tulostustyönipun toteutus. Bundle-olio koostuu **yhdestä tai useammasta PrintJob-oliosta**.

Bundle-luokka sisältää seuraavat kentät. Koordinaatit (x, y, z) ovat Bundlen sisältämien PrintJobien koordinaattien keskiarvo.

private int bundleID
: Bundlen tunnus, ei-negatiivinen kokonaisluku

private int x
: Bundlen x-koordinaatti, ei-negatiivinen kokonaisluku

private int y
: Bundlen y-koordinaatti, ei-negatiivinen kokonaisluku

private int z
: Bundlen z-koordinaatti, ei-negatiivinen kokonaisluku

private String[] optimalPrinters
: String-taulukko, joka sisältää tälle nipulle optimaaliset tulostimet. Taulukon koko on aina sama kuin luokan vakio *NUMBEROFOPTIMALPRINTERS* (määrittely alla). Jos taulukon jossain indeksissä ei ole tulostimen nimeä, siihen talletetaan merkkijono " ". Taulukkoon talletettavien tulostinten nimet voivat olla enintään 10 merkkiä pitkiä.

private Vector jobs
: Vector-olio, joka sisältää kaikki tähän nippuun kuuluvat PrintJob-oliot

final static int NUMBEROFOPTIMALPRINTERS
: Tämä on vakio, jonka arvoksi määritellään *optimalPrinters*-taulukon koko. Oletusarvo on 10.

Luokassa on kolme **konstruktoria**:

public Bundle(int bundleID)
: Tämä on peruskonstruktori, jota käytetään vain testauksessa. Parametrina annetaan muodostettavan Bundle-olion *bundleID*. Rajoitukset kerrotaan *bundleID*-kentän kohdalla. Konstruktori asettaa muodostettavalle Bundle-oliolle yhden PrintJob-olion vektoriin *jobs* PrintJob-luokan peruskonstrukto-

rilla *PrintJob*. *OptimalPrinters*-taulukko alustetaan " "-merkkijonoilla.

public Bundle(PrintJob job, int bundleID)

Tämä konstruktori luo Bundle-olion, joka sisältää yhden parametrina saadun PrintJob-olion. Nipun käyttöliittymä käyttää tätä konstruktoria muodostaessaan uuden Bundle-olion. Lisäksi parametrina annetaan muodostettavan Bundle-olion *bundleID*, joka on muodostettava käyttämällä Bundlen staattista metodia *createID* (katso metodi *createID*). Konstruktori asettaa saamansa parametrit (rajoitukset kerrotaan vastaavien kenttien rajoitusten kohdalla). Lisäksi alustetaan *optimalPrinters*-taulukko " "-merkkijonoilla. Bundlen koordinaatit saadaan suoraan parametrina saadun PrintJob-olion koordinaateista.

public Bundle(int bundleID, String[] optimalPrinters, Vector jobs)

Konstruktori luo halutunlaisen Bundle-olion, jonka ominaisuudet saadaan parametreina. Bundlen koordinaatit ovat parametrina saatavan *jobs*-vektorin sisältämien PrintJob-olioiden koordinaattien keskiarvo. Parametrien rajoitukset kerrotaan vastaavien kenttien rajoitusten kohdalla. Parametri *bundleID* luodaan Bundlen staattisella metodilla *createID* (katso metodi *createID*).

Luokka sisältää seuraavat **get-metodit**, jotka palauttavat vastaavan kentän arvon:

public int getBundleID()
public int getX()
public int getY()
public int getZ()
public String[] getOptimalPrinters()
public Vector getJobs()

Luokka sisältää seuraavat kenttiin liittymättömät get-metodit:

public int getNumberOfPages()

Metodi palauttaa Bundlen sisältämien PrintJob-olioiden sivumäärien summan.

public String getForcedPrinter()

Metodi palauttaa kyseisen Bundle-olion pakotetun tulostimen. Jos Bundlen sisältämillä PrintJob-olioilla on eri pakotettuja tulostimia, palautetaan näistä

ensimmäisenä vastaan tuleva[5]. Jos Bundlen yhdelläkään PrintJob-oliolla ei ole pakotettua tulostinta, palautetaan merkkijono "Ei".

Luokka sisältää seuraavat **set-metodit**, joilla voi asettaa kenttien arvoja:

public boolean setOptimalPrinters(String[] printers)

Metodi palauttaa true, jos asettaminen onnistui, muuten false. Parametrina saatava *printers*-taulukko voi sisältää korkeintaan 10 merkin mittaisia tulostinten nimiä. Taulukon koko voi olla enintään luokassa määritellyn *NUMBEROFOPTIMALPRINTERS*-vakion kokoinen. Jos taulukon koko on alle tuon vakion, luodaan uusi *NUMBEROFOPTIMALPRINTERS* kokoinen taulukko, johon kopioidaan *printers*-taulukon arvot ja loput arvot alustetaan " "-merkkijonoilla. Tämä tehdään siksi, että Bundle-olion *optimalPrinters*-taulukko on aina kooltaan *NUMBEROFOPTIMALPRINTERS*.

public boolean setJobs(Vector printjobs)

Metodi palauttaa true, jos asettaminen onnistui, muuten false. Parametrina annettava Vector-olio voi sisältää vain PrintJob-olioita. Metodi päivittää myös Bundlen koordinaattien arvot vastaamaan *printjobs*-vektorin sisältämien PrintJob-olioiden koordinaattien keskiarvoja.

Luokka sisältää lisäksi seuraavat metodit:

public Bundle union(Bundle otherBundle, int componentID,
int componentBundleCounter)

Metodi yhdistää Bundle-olion parametrina saatavaan toiseen Bundle-olioon *otherBundle*. Metodi palauttaa yhdisteenä saatavan Bundle-olion. Jos yhdistäminen ei onnistunut, palautetaan null. Parametri *componentID* on metodia käyttävän komponentin oma tunniste, ja *componentBundleCounter* on metodia käyttävän komponentin laskurin arvo muodostetuista Bundle-olioista. Nämä kaksi parametria tarvitaan muodostettaessa *union*-metodissa yhdisteelle uusi *bundleID* *createID*-metodilla (katso metodi *createID*). Yhdiste-Bundlen optimaalisten tulostinten taulukkoon talletetaan molempien Bundle-olioiden optimaalisten tulostinten leikkaus. Katso myös *Analyzer*-

5 Tällaista Bundle-oliota ei tulisi koskaan olla olemassa, sillä eri pakotetut tulostimet sisältävien PrintJob-olioiden pitäisi olla eri Bundle-ilmentymissä. Tästä ei kuitenkaan huolehdi Bundle-luokka itse vaan Bundle-olioita luovat komponentit, kuten analysoija.

luokan *fusion*-metodi.

public static int createID(int componentID, int componentBundleCounter)

Tämä metodi on keskitetty tapa luoda yksikäsitteisiä tunnuksia uusille Bundle-ilmentymille. Se saa parametreinaan kaksi kokonaislukua. Ensimmäinen luku on sen Nippu-järjestelmän komponentin tunnus, joka on luomassa uutta Bundle-ilmentymää. Toinen luku on saman komponentin sisäinen, luotuja Bundle-ilmentymiä laskeva laskuri. Komponentin tunnuksen oletetaan yksilöivän komponentin kaikista Nippu-järjestelmään ajon aikana liittyvistä komponenteista yksikäsitteisesti.

Näistä kahdesta kokonaisluvusta *createID*-metodi muodostaa ja palauttaa uuden Bundle-ilmentymän tunnukseksi tarkoitetun kokonaisluvun siten, että kahdelle eri Nippu-järjestelmän komponentille ei koskaan palauteta samaa tunnusta. Myöskään saman komponentin sisällä luoduille Bundle-ilmentymille ei anneta samaa tunnusta. Tunnuksen yksikäsitteisyys saadaan aikaan tallettamalla Bundle-ilmentymän tunnuksen ylimpiin 16 bittiin komponentin tunnuksen alimmat 16 bittiä ja Bundle-ilmentymän tunnuksen alimpiin 16 bittiin komponentin sisäisen Bundle-laskurin alimmat 16 bittiä. Metodi takaa palautettujen tunnusten yksikäsitteisyyden, kun Nipun ajonaikaiset komponenttitunnukset eroavat kaikki toisistaan alimmissa 16 bitissään (siis eri komponentteja saa olla korkeintaan 65536 kappaletta) ja kun minkään komponentin sisällä ei olla luotu yli 65536 Bundle-ilmentymää. Jos Bundle-laskuri kasvaa tämän rajan yli, on siltikin epätodennäköistä, että kahdella samanaikaisesti olemassa olevalla Bundle-ilmentymällä tulisi olemaan sama tunnus.

public String serialize()

Metodi palauttaa merkkijonoesityksen luokan oliosta. Merkkijono ei ole tarkoitettu käyttäjälle vaan esimerkiksi toiselle oliolle tai ohjelmalle. Merkkijonossa ovat aluksi kenttien *bundleID, x, y* ja *z* arvot %-merkillä erotettuina. Seuraavaksi erotetaan %-merkillä merkkijono, joka sisältää kaikki taulukon *optimalPrinters* arvot erotettuina pilkuilla toisistaan. Merkkijonoon tulevat myös ne *optimalPrinters*-taulukon indeksit, joissa ei ole tulostinta, siis merkkijonot " ". Lopuksi erotetaan vielä %-merkillä

merkkijono, joka koostuu kaikista *jobs*-vektorin PrintJob-olioiden merkkijonoesityksistä pilkuilla toisistaan erotettuina. Metodin palauttama merkkijono on siis muotoa (ilman rivinvaihtoja):

*bundleID%x%y%z%pr1,pr2,...,prN%*
*jobName1%jobID1%user1%x1%y1%1%numberOfPages1%forcedPrinter1%senderUI1,jobName2%jobID2%user2%x2%y2%z2%numberOfPages2%forcedPrinter2%senderUI2,…*
*,jobNameN%jobIDN%userN%xN%yN%zN%numberOfPagesN%forcedPrinterN%senderUIN*

public static Bundle deserialize(String serializedBundle)

Staattinen metodi saa parametrina merkkijonon, joka vastaa *serialize*-metodin tuottamaa merkkijonoa. Tästä merkkijonosta metodi luo ja palauttaa sitä vastaavan Bundle-olion. Parametrin on oltava aivan samanlainen kuin edellisessä *serialize*-metodissa kuvattiin.

public Bundle[] breakBundle(int componentID, int componentBundleCounter)

Metodi hajottaa nipun useiksi nipuiksi, joista jokainen sisältää vain yhden PrintJob-olion. Niput palautetaan taulukossa. Parametri *componentID* on metodia käyttävän komponentin oma tunniste, ja *componentBundleCounter* on metodia käyttävän komponentin laskurin arvo muodostetuista Bundle-olioista. Nämä kaksi parametria tarvitaan, kun *union*-metodissa jokaiselle uudelle nipulle luodaan uusi *bundleID createID*-metodilla (katso metodi *createID*). Jokaisen uuden Bundle-olion optimaalisten tulostinten taulukko on kopio alkuperäisen Bundlen vastaavasta taulukosta.

public boolean addJob(PrintJob job)

Metodi lisää parametrina annettavan PrintJob-olion *jobs*-vektoriin. Metodi palauttaa true, jos lisäys onnistui, muuten false. Metodi päivittää automaattisesti Bundlen koordinaattien arvot vastaamaan sen PrintJob-olioiden koordinaattien keskiarvoja.

public boolean addOptimalPrinter(String printer)

Metodi lisää parametrina annettavan tulostimen *optimalPrinters*-tauluk-

koon. Metodi palauttaa true, jos lisäys onnistui, muuten false. Jos taulukossa on jo kyseinen tulostin, palautetaan myös true. Jos taulukko on täynnä, palautetaan false. Parametri *printer* voi olla enintään 10 merkkiä pitkä.

public boolean removeOptimalPrinter(String printer)

Metodi poistaa parametrina annettavan tulostimen *optimalPrinters*-taulukosta. Metodi palauttaa true, jos poistaminen onnistui, muuten false.

public boolean hasForcedPrinter()

Metodi palauttaa true, jos vähintään yhdellä Bundlen sisältämällä PrintJobilla on pakotettu tulostin asetettuna. Muutoin palautetaan false.

public String toString()

Metodi palauttaa merkkijonoesityksen luokan oliosta. Merkkijonoesitys on tarkoitettu ihmisen luettavaksi esimerkiksi testausta varten.

public void sort()

Metodi järjestää Bundlen sisältämät PrintJobit siten, että sellaiset PrintJobit, joihin liittyy sama käyttäjätunnus, ovat Vector-taulukon vierekkäisissä indekseissä.

public static void main(String[] args)

pääohjelmametodi testausta varten

### 3.1.3 NippuUser-luokka

Tämä luokka kuvaa asetuksissa määriteltyä **järjestelmän käyttäjää**. Luokan kentät ovat käyttäjätunnus ja sijaintikoordinaatit. Käyttäjätunnus on enintään kahdeksan merkkiä pitkä merkkijono. Sijaintikoordinaatit ovat ei-negatiivisia kokonaislukuja.

Luokassa on yksi **konstruktori**:

public NippuUser(String ID, int X, int Y, int Z)

Luokka sisältää seuraavat **get-metodit**, jotka palauttavat halutun kentän arvon:

```
public int getPosX()
public int getPosY()
public int getPosZ()
public String getUserID()
```

### 3.1.4 NippuPrinter-luokka

Tämä luokka kuvaa **järjestelmän tulostinta**. Luokan kentät ovat tulostimen nimi, nopeus ja sijaintikoordinaatit. Tulostimen nimi on enintään 10 merkkiä pitkä merkkijono. Tulostimen nopeus on aidosti positiivinen kokonaisluku. Sijaintikoordinaatit ovat ei-negatiivisia kokonaislukuja.

Luokassa on yksi **konstruktori**:

```
public NippuPrinter(String ID, int X, int Y, int Z, int speed,
                    Vector connectionsVector)
```

Luokka sisältää seuraavat **get-metodit**, jotka palauttavat halutun kentän arvon:

```
public String getPrinterID()
public int getPrinterX()
public int getPrinterY()
public int getPrinterZ()
public int getPrinterSpeed()
```

### 3.1.5 ConfigurationParser-luokka

Tämästä luokasta luodut oliot sisältävät **järjestelmän yhteisen asetustiedoston**. Levyltä luettu tiedosto annetaan merkkijonoparametrina luokan konstruktorille, joka luo *ConfigurationParser*-olion. Luokan tehtävänä on tulkita asetustiedoston merkkijonoesitys.

Luokka sisältää yhden **konstruktorin:**

public ConfigurationParser(String configuration)

Konstruktori saa parametrina asetustiedoston merkkijonona. Parametri *configuration* on asetustiedoston sisältö merkkijonona – ei tiedoston nimi.

Luokka sisältää seuraavat julkiset **metodit:**

public Vector getUsers()

Metodi palauttaa asetustiedoston sisältämät tiedot käyttäjistä *NippuUser*-olioina Vector-tietorakenteessa (katso *NippuUser*-luokka).

public Vector getPrinters()

Metodi palauttaa asetustiedoston sisältämät tiedot tulostimista *NippuPrinter*-olioina Vector-tietorakenteessa (katso *NippuPrinter*-luokka).

public String getString(String constant)

Metodille annetaan parametrina asetustiedoston jonkin vakion nimi (esimerkiksi *MAX_JOBS*, katso luku 3.3.2). Metodi palauttaa kyseisen vakion arvon merkkijonona. Jos annettua vakiota ei ole asetustiedostossa, palautetaan null.

public int getInt(String constant)

Metodille annetaan parametrina asetustiedoston jonkin vakion nimi (esimerkiksi *MAX_JOBS*, katso luku 3.3.2). Metodi palauttaa kyseisen vakion arvon kokonaislukuna. Määritellyn vakion on oltava tyypiltään int. Jos vakion tyyppi ei ole int tai jos vakiota ei ole asetustiedostossa, palautetaan 0.

public float getFloat(String constant)

Metodille annetaan parametrina asetustiedoston jonkin vakion nimi (esimerkiksi *SAMEUSERMULTIPLIER*, katso luku 3.3.2). Metodi palauttaa kyseisen vakion arvon float-tyyppisenä desimaalilukuna. Määritellyn vakion on oltava tyypiltään float tai int. Jos vakion tyyppi ei ole float tai int tai jos vakiota ei ole asetustiedostossa, palautetaan 0.0f.

public String toString()

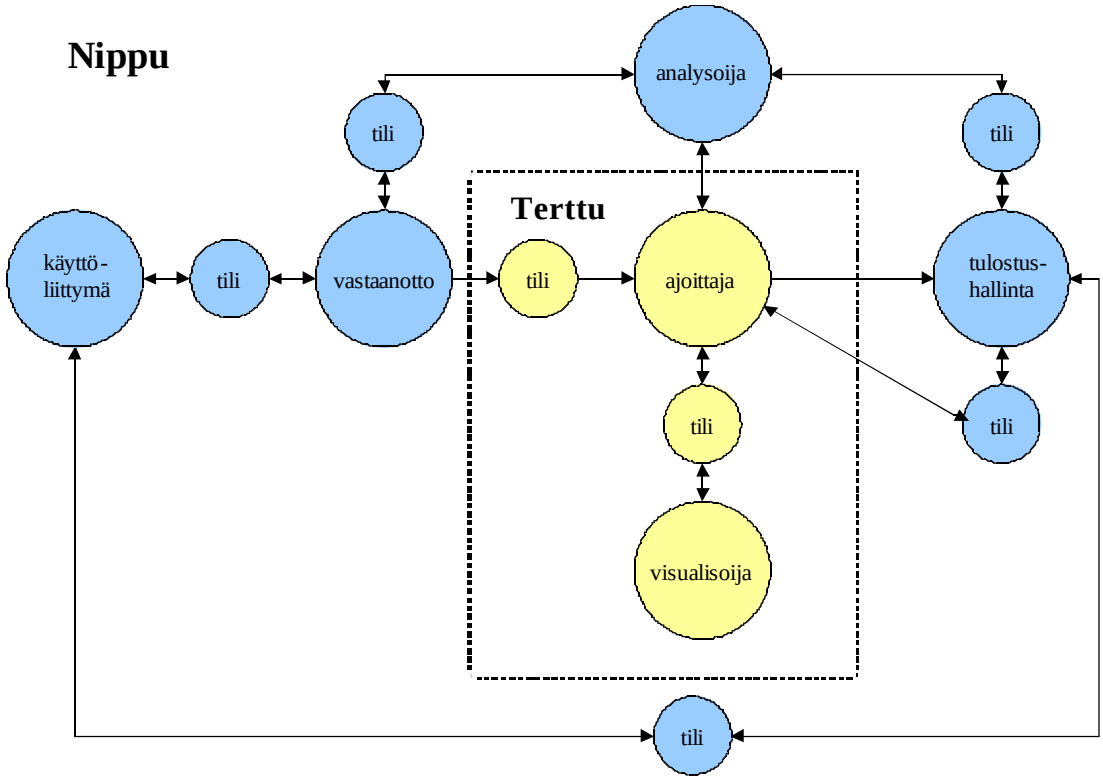
Metodi palauttaa merkkijonoesityksen *ConfigurationParser*-oliosta. Metodi on tarkoitettu luokan testaamiseen.

public static void main(String[] args)

Tämä on testipääohjelma testausta varten.

## 3.2 Arkkitehtuurikuvaus

Nippu käyttää Tertun **ajoittajaa**. Ajoittaja ja Nipun **analysoija** yhdessä muodostavat järjestelmän ytimen. Analysoijan lisäksi Nippuun kuuluvat **käyttöliittymä**, tulostustöiden niputtamista tehostava **vastaanotto**, tulostinten toimintaa emuloiva **tulostushallinta** sekä kaikkia tietoliikenneyhteyksiä hallinnoiva **tietoliikennekomponentti** (lyhennettynä tili). Tertun **visualisoijan** avulla voidaan seurata ajoittajan toimintaa. Komponentit on merkitty kuvaan 1. Ajoittaja käynnistää analysoijan ja tulostushallinnan, joten niitä ei tarvitse käynnistää erikseen.

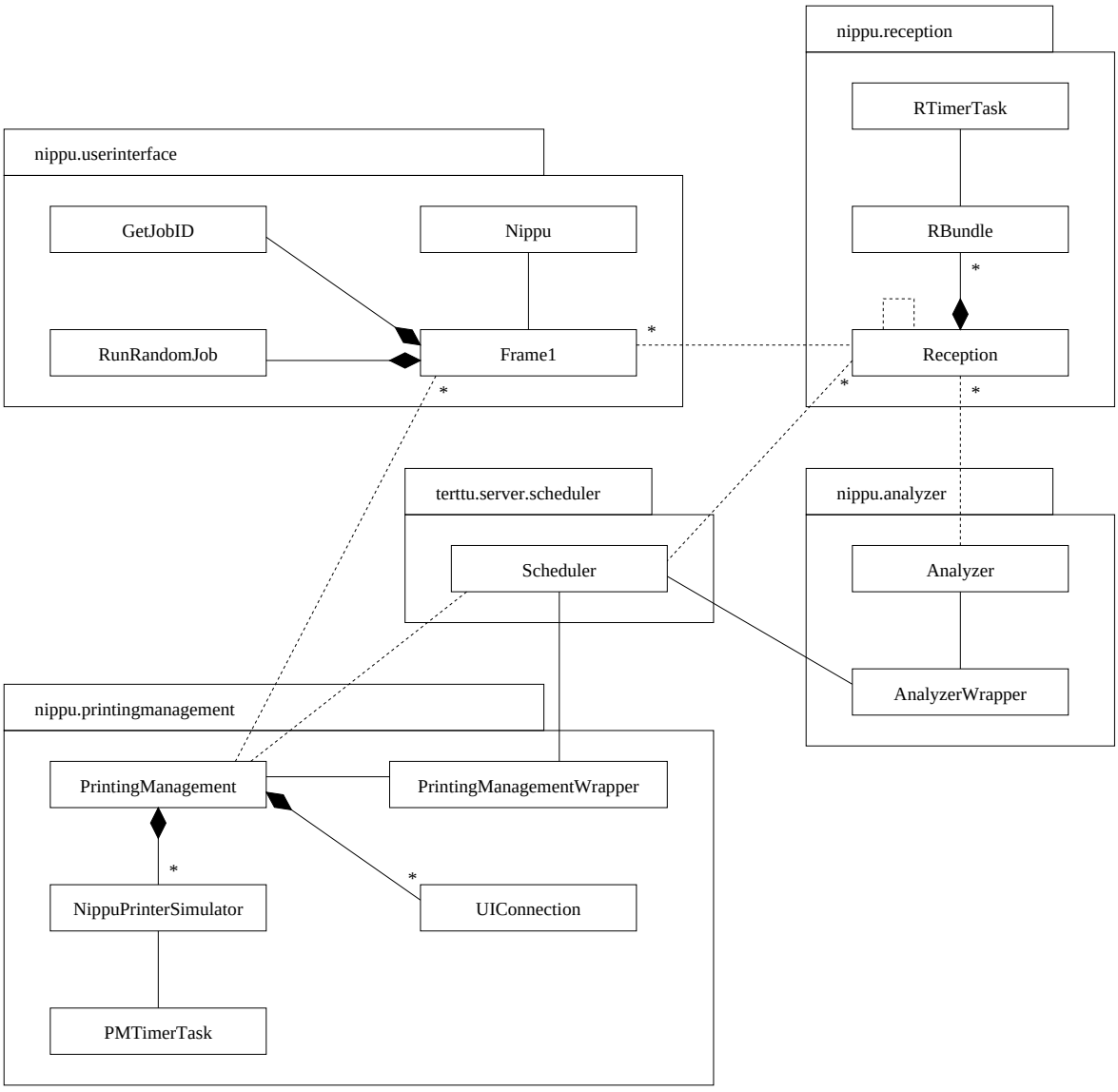



*Kuva 1: Yleiskaavio Nipun arkkitehtuurista ohjelmistokomponenttitasolla. Nuolet kuvaavat tiedon kulkua järjestelmässä. (Pelkästään kuittaustyyppistä tietoa sisältävät tiedon kulkusuunnat on selvyyden vuoksi jätetty merkitsemättä.)*

Tulostustyöt annetaan **käyttöliittymässä**, josta ne toimitetaan tietoliikennekomponentin kautta vastaanottoon. **Vastaanotossa** työt odottavat puskuroituina tietyn ajan muita sellaisia tulostustöitä, jotka voitaisiin mahdollisesti yhdistää samaksi nipuksi. Vastaanotto päättelee **analysoijasta** saatujen tietojen perusteella, mitkä työt tulevat todennäköisesti yhdistymään. Seuraavaksi työt siirtyvät **ajoittajaan**, joka arvottaa niitä analysoijan avulla. Ajoitta-

ja lähettää työt **tulostushallintaan** sopivassa järjestyksessä. Tulostushallinta siirtää saamansa tulostustyöt sopivaan tulostusjonoon ja lähettää ilmoituksen käyttöliittymälle, kun tietty työ on tulostettu. Tulostushallinta välittää analysoijalle tietoja tulostinten sijainneista.

Komponenttien väliset luokkasuhteet ilmenevät **kuvan 2 luokkakaaviosta**, johon on merkitty järjestelmän keskeisimmät luokat.




*Kuva 2: Nipun yleisluokkakaavio. Kuvassa ovat Nipun komponenttien keskeiset omat luokat. Tertun luokista on mainittu vain välttämättömimmät. Kuvaan ei ole merkitty tietoliikenneluokkia eikä yleisiä luokkia, kuten PrintJob ja Bundle. Tavalliset luokkayhteydet on merkitty yhtenäisellä viivalla ja epäsuorat, tietoliikenteen kautta tapahtuvat yhteydet katkoviivalla. Kaikki yhteydet ovat yksi-yhteen, ellei ole toisin merkitty.*

**Nippu käynnistetään** yhdellä komentojonolla, joka käynnistää Tertun, vastaanoton ja yhden käyttöliittymän. Terttu käynnistää analysoijan ja tulostushallinnan. Nipun mukana toimitetaan komentojonot myös yksittäisten komponenttien erikseen käynnistämiseen. Tertun visualisoija voidaan käynnistää erillisellä komennolla.

## 3.3 Liittymäkuvaus

Tässä luvussa kuvataan järjestelmän ulkoiset ja sisäiset liittymät. Ulkoisia liittymiä ovat käyttöliittymä ja asetustiedostot. Sisäisiä liittymiä ovat tietoliikenneyhteydet. Lisäksi sekä ulkoisiin että sisäisiin liittymiin kuuluu luokkarajapintoja.

### 3.3.1 Käyttöliittymä

Tulostustyöt annetaan järjestelmään käyttöliittymän kautta. Käyttöliittymä siirtää työt edelleen vastaanottoon ja näyttää tietoja muun muassa tulostusjonojen tilasta.

Käyttöliittymä on toteutettu yhtenä ohjelmakomponenttina, joka sisältää kaksi erilaista liittymää: **peruskäyttäjän liittymä** ja **pääkäyttäjän liittymä**. Kumpaankin sisältyy **visualisointiliittymä**. Käyttöliittymä on suunniteltu siten, että käyttö on helppoa ja tehokasta. Liitteessä 2 on kuva käyttöliittymästä siinä tilanteessa, kun järjestelmään on otettu yhteys pääkäyttäjänä. Jos järjestelmään olisi otettu yhteys peruskäyttäjänä, puuttuisivat käyttöliittymästä osat "Pääkäyttäjän satunnaistyöt" ja "Pääkäyttäjän toiminnot".

---

**Peruskäyttäjän liittymä**

Käyttöliittymä käynnistetään peruskäyttäjän tilaan antamalla komentorivillä komento "**nippu käyttäjätunnus**". Käyttäjätunnus-parametri on käyttäjän käyttäjätunnus, joka tulee olla määritelty asetustiedostossa. Käyttöliittymä voidaan käynnistää myös antamalla pelkästään komento "**nippu**". Silloin käyttäjätunnus luetaan ympäristömuuttujasta *USER*.

Peruskäyttäjän liittymä koostuu seuraavista osista:

- **Tulosteen nimi** on tekstikenttä, johon kirjoitetaan tulostettavan työn tiedostonimi. Oletuksena kentässä lukee "oletustiedosto.txt". Tätä kenttää ei voi jättää tyhjäksi, eli tulostettavalle työlle on annettava nimi. Nimen pituus tulee olla 1–255 merkkiä, ja se ei voi sisältää pilkkuja eikä %-merkkejä.

- **Sivumäärä** on tekstikenttä, johon tulee syöttää tulostustyön sivumäärä. Oletuksena kentässä on "1". Tätä kenttää ei voi jättää tyhjäksi. Sivumäärän on oltava kokonaisluku väliltä [1, Integer.MAX_VALUE][6].

- **Käyttäjätunnus** on pudotusvalikko, jonka arvona on tulostajan käyttäjätunnus. Peruskäyttäjä ei voi muuttaa tätä kenttää.

- **Pakota tulostimelle** on pudotusvalikko, josta voi valita tulostimen. Jos valitsee vaihtoehdon "Ei", järjestelmä valitsee automaattisesti sopivan tulostimen työlle. Muuten työ menee pakotetulle tulostimelle. Oletusarvona kentässä on "Ei".

- **Tulosta**-painike. Käyttäjä painaa Tulosta-painiketta, kun hän haluaa lähettää tulostustyön eteenpäin.

- **Omat työt** on tekstialue, joka kertoo tietoja tulostajan omista töistä. Se on loki, jossa näkyvät kaikki kyseisestä käyttöliittymästä lähetetyt työt siten, että jokaista työtä kohti on vain yksi rivi. Kyseinen rivi päivittyy aina sen mukaan, mikä kyseisen työn tila milloinkin on. Työllä voi olla yksi neljästä tilasta:

    1. Työ on otettu vastaan. (Tähän tilaan työ siirtyy, kun se lähetetään käyttöliittymästä.)
    2. Työ on jonossa jollain tulostimella. (Tulostushallinta on valinnut tulostimen ja ilmoittanut siitä käyttöliittymälle.)
    3. Työ on tulostumassa tulostimella.

6 Tämän dokumentin useissa kohdissa käytetään arvovälien ilmaisussa matemaattista sulkeisiin perustuvaa merkintätapaa, koska se tekee ilmaisusta lyhyen ja yksikäsitteisen. Esimerkiksi [a, b] tarkoittaa arvoväliä a:sta b:hen siten, että välin päätepisteet a ja b ovat mukana. Normaali sulkumerkki hakasulkeen paikalla tarkoittaisi, että kyseinen päätepiste ei ole mukana.

4. Työ on tulostunut tulostimelle.

Jokaisella rivillä näkyvät seuraavat tiedot: työn tunniste, työn tila, valittu tulostin, tulostajan käyttäjätunnus, työn nimi. Tulostuneet työt voi poistaa näkymästä painamalla tekstikentän alla olevaa painiketta.

---

**Pääkäyttäjän liittymä**

Pääkäyttäjän liittymän voi käynnistää komennolla "**nippu käyttäjätunnus p**". Parametri "p" avaa pääkäyttäjän liittymän. Ohjelman voi käynnistää myös komennolla "**nippu p**", jolloin käyttäjätunnus luetaan ympäristömuuttujasta *USER*.

Pääkäyttäjän liittymä sisältää samanlaisen näkymän yksittäisten töiden lähettämistä varten kuin peruskäyttäjällä. Ainoa ero on, että pääkäyttäjä voi vapaasti vaihtaa käyttäjätunnusta pudotusvalikosta. Näin pääkäyttäjä voi lähettää yksittäisiä tulostustöitä millä tahansa käyttäjätunnuksella.

Pääkäyttäjän liittymä sisältää lisäksi seuraavat osat:

- **Sivumäärä** on kaksi tekstikenttää, joihin annetaan ala- ja yläraja satunnaistöiden sivumäärille. Oletusarvoina kentissä on "10" ja "30". Arvoja muutetaan napsauttamalla hiirellä kenttää ja syöttämällä haluttu arvon kenttään. Käyttöliittymä arpoo sivumäärän annetulta väliltä, kun uusi satunnaistyö lähetetään. Kenttien vieressä on valintaruutu, jonka valitsemalla saa päälle vakiosivumäärän. Kun ruudun valitsee, oikea tekstikenttä tyhjenee ja muuttuu harmaaksi eikä sitä voi enää muuttaa. Kentän vasemmalle puolelle voi sitten antaa haluamansa vakiosivumäärän. Sivumääräkenttä on pakko täyttää. Sallitut arvot ovat kokonaislukuja väliltä [1, Integer.MAX_VALUE].

- **Lähetystiheys** on kaksi tekstikenttää, joihin annetaan lähetystiheyden ala- ja yläraja sekunteina. Oletusarvoina kentissä on "5" ja "15". Arvoja muutetaan napsauttamalla hiirellä kenttää ja syöttämällä haluttu arvo kenttään. Käyttöliittymä arpoo lähetystiheyden annetulta väliltä erikseen jokaisen lähetyskerran jälkeen. Kenttien vieressä on valintaruutu, jonka valitsemalla saa päälle vakiolähetystiheyden. Kun ruudun va-

litsee, kentän oikea puoli tyhjenee ja muuttuu harmaaksi eikä sitä voi enää muuttaa. Kentän vasemmalle puolelle voi sitten antaa haluamansa vakiolähetystiheyden. Lähetystiheys-kenttä on pakko täyttää. Sallitut arvot ovat kokonaislukuja väliltä [0, Integer.MAX_ VALUE].

- **Käyttäjätunnus** on pudotusvalikko, joka sisältää kaikki järjestelmän tuntemat käyttäjätunnukset sekä satunnainen-vaihtoehdon. Pääkäyttäjä voi vapaasti valita käyttäjätunnuksen, jolloin satunnaistyöt lähtevät koko ajan valitun käyttäjän nimissä. Jos pääkäyttäjä valitse satunnainen-vaihtoehdon, järjestelmä arpoo jokaiselle satunnaistyölle käyttäjän erikseen. Näin voidaan simuloida yleistä tulostuspalvelimen kuormitusta. Oletusarvona on valittuna "satunnainen".

- **Pakota tulostimelle** on samanlainen pudotusvalikko kuin peruskäyttäjän liittymässä.

- **Käynnistä**-painikkeella satunnaistöiden lähettämisen voi kytkeä päälle ja vastaavasti pois päältä. Satunnaistöitä lähetetään vasta päälle kytkemisen jälkeen. Kun lähetys on päällä, painikkeen teksti on "pysäytä". Kun satunnaistöitä ei enää haluta tuottaa, on painikkeella kytkettävä satunnaistöiden lähettäminen pois päältä.

  Satunnaistöiden lähetys loppuu ja painike ponnahtaa automaattisesti ylös aina, kun pääkäyttäjä tekee muutoksia edellä mainittuihin pääkäyttäjän liittymän kenttiin. Satunnaistyöt on siis käynnistettävä uudelleen, kun jotain muutoksia on tehty. Näin estetään ei-toivottujen tulostustöiden lähettäminen.

- **Maksimiviive palvelun saannille** näyttää ensin maksimiviiveen nykyisen arvon. Sen jälkeen on tekstikenttä, johon voi antaa halutun maksimiviiveen palvelun saannille sekunteina. Arvo muutetaan painamalla tekstikentän vieressä olevaa painiketta. Muuttaminen on sallittu ainoastaan ensimmäisenä avatussa pääkäyttäjän liittymässä. Kentällä ei ole käyttöliittymässä oletusarvoa, vaan arvo tulee aina vastaanotosta.

- **Ajoitus**-kohdassa on painike, jolla voi pysäyttää ja käynnistää ajoituksen. Jos ajoitus on käynnissä, painike on ylhäällä ja sen teksti on "pysäytä". Jos taas ajoitus on

pysäytetty, painike on alhaalla ja sen teksti on "käynnistä". Kun ajoitus on pysäytetty, järjestelmään voi edelleen normaalisti syöttää lisää tulostuspyyntöjä. Tällöin töitä ei kuitenkaan poistu järjestelmästä, jolloin järjestelmään saadaan asetettua haluttu kuorma. Tulostushallinta saattaa myös pysäyttää ajoituksen, kun se odottaa tulostusjonojen tyhjentymistä. Tällöin painike kertoo, että ajoitus on pysäytetty, mutta käyttäjää ei estetä käynnistämästä sitä. Jos käyttäjä käynnistää ajoituksen tulostushallinnan pysäytettyä sen, tulostushallinta pysäyttää ajoituksen tarvittaessa uudestaan. Painike on käytettävissä ainoastaan ensimmäisenä avatussa pääkäyttäjän liittymässä.

---

**Visualisointiliittymä**

Peruskäyttäjän ja pääkäyttäjän liittymät sisältävät saman tulostimia visualisoivan liittymän. Visualisointiliittymä **esittää järjestelmän tulostimet**, joita voi olla enintään kymmenen. Lisäksi esitetään kaikki tulostinjonot, joista näkyvät jokaiseen työhön liittyvät tiedot.

Tulostimet ja niihin liittyvät jonot esitetään sarakkeittain. Sarakkeet sisältävät seuraavat tiedot:

- tulostimen nimi
- tulostumassa olevan työn tulostajan käyttäjätunnus ja työn nimi
- tulostusjonossa olevien töiden tulostajien käyttäjätunnuksen ja töiden nimet

Hiiren osoitinta siirtämällä saa näkyviin vihjetekstejä, joissa kerrotaan vielä tarkemmat tiedot tulostimesta ja töistä.

Sekä perus- että pääkäyttäjän näkymässä on **tulostusloki**-kenttä, joka on osa visualisointiliittymää. Tulostuslokiin kirjataan kaikki järjestelmän tapahtumat tapahtumajärjestyksessä. Tähän lokiin tulee siis jokaisesta työstä neljä erillistä riviä peruskäyttäjän liittymän kohdassa "omat työt" mainittujen tilamuutosten mukaan. Jokaiselle riville kirjataan työn tunniste, tulostajan käyttäjätunnus, työn nimi, sivumäärä, valittu tulostin ja työn tila tapahtuman jälkeen.

Käyttöliittymät rekisteröityvät tulostushallintaan tietoliikenneyhteydellä, minkä jälkeen tulostushallinta lähettää käyttöliittymille töiden tilasta kertovia viestejä. Visualisointiliittymän tietoja päivitetään tämän perusteella.

Käyttöliittymät rekisteröityvät myös vastaanottoon tietoliikenneyhteydellä, minkä jälkeen vastaanotto lähettää käyttöliittymille palvelunsaannin maksimiviiveestä kertovia viestejä. Pääkäyttäjän liittymän tietoja päivitetään tämän perusteella.

### 3.3.2 Asetustiedostot

Nipussa on yksi yhteinen asetustiedosto, ja tietyillä komponenteilla on komponenttikohtainen asetustiedosto:

- **Yhteinen asetustiedosto – nippu.conf**
  Yhteisen asetustiedoston lukee tulostushallinta. Siinä sijaitsevat muun muassa käyttäjiä ja tulostimia koskevat asetukset. Yhteinen asetustiedosto on luonteeltaan sellainen, että kaikki komponentit voivat käyttää sen sisältämiä tietoja.
- **Käyttöliittymän asetustiedosto – nippu-userinterface.conf**
  Käyttöliittymän asetustiedoston lukee käyttöliittymä. Siinä sijaitsevat tulostushallinnan ja vastaanoton verkkoparametrit.
- **Vastaanoton asetustiedosto – nippu-reception.conf**
  Vastaanoton asetustiedoston lukee vastaanotto. Siinä sijaitsevat ajoittajan ja analysoijan verkkoparametrit sekä nippujen lähetystapa -parametri.
- **Analysoijan asetustiedosto – nippu-analyzer.conf**
  Analysoijan asetustiedoston lukee analysoija. Siinä sijaitsevat tulostushallinnan verkkoparametrit.
- **Tulostushallinnan asetustiedosto – nippu-printingmanagement.conf**
  Tulostushallinnan asetustiedoston lukee tulostushallinta. Siinä sijaitsevat ajoittajan verkkoparametrit.

Asetustiedostojen syntaksit muistuttavat toisiaan, mutta tiedostoissa määritellään erilaisia asioita. Yksi tietokokonaisuus määritellään aina **yhdellä rivillä**. Esimerkiksi jos toimintaympäristö sisältää 10 tulostinta, asetustiedostoon tulee 10 riviä tietoa tulostimista siten, että jokaista tulostinta kohti tulee yksi rivi. Asetustiedostossa voi olla myös tyhjiä rivejä ja #-merkillä alkavia kommenttirivejä. Nämä rivit eivät vaikuta Nipun toimintaan.

Nipun mukana toimitetaan myös muita asetustiedostoja, jotka eivät kuitenkaan ole Nipun liittymiä (esimerkiksi Tertun asetustiedosto). Niiden käyttö selitetään Nipun käyttöohjeessa.

Seuraavaksi kuvataan asetustiedostojen sisällöt. Jokaisesta asetustiedostosta on malli liitteessä 4.

---

**Yhteinen asetustiedosto – nippu.conf**

Järjestelmän **käyttäjiä, tulostimia ja analysoijan vakioita koskevat asetukset** määritellään yhteisessä asetustiedostossa. Jokaisesta tulostimesta tiedostoon merkitään tulostimen nimi ja nopeus sekä sen sijainti kolmiulotteisessa koordinaatistossa (x, y, z). Jokaisesta käyttäjästä tiedostoon kirjataan käyttäjätunnus ja vakiosijaintikoordinaatit (x, y, z). Lisäksi tiedostoon kirjataan, kuinka monta työtä järjestelmään voidaan enintään ottaa. Asetustiedostossa myös määritellään vakioita, jotka vaikuttavat analysoijan toimintaan.

Asetustiedosto sisältää seuraavia asetustyyppejä: maksimimäärä tulostustöille, käyttäjien asetukset, tulostinten asetukset ja analysoijan vakiot. Seuraavaksi kerrotaan, kuinka asetustyypit määritellään. Jokaisen rivin eri kenttien erottamiseen käytetään %-merkkiä. Kaikkien kokonaislukujen enimmäisarvo on Integer.MAX_VALUE, ellei toisin mainita. Joidenkin asetusten arvot ovat reaalilukuja, mutta ne kirjoitetaan desimaalilukuina käyttäen desimaalierottimena pistettä eikä pilkkua, joka olisi suomalaisen standardin mukainen tapa. Pisteen käyttö helpottaa Javalla ohjelmointia.

- **Tulostustöiden enimmäismäärä**

  **MAX_JOBS=xx**

  Tässä xx on aidosti positiivinen kokonaisluku, joka kertoo töiden enimmäismäärän. Oletusarvona tälle on "100". Tiedosto voi sisältää vain yhden MAX_JOBS-rivin. *Tätä asetusparametria ei käytetä tässä ohjelmistoversiossa.* Tosiasiassa töiden enimmäismäärän rajoittaminen vaatisi lisäominaisuuksien toteuttamista, koska työt tulevat käyttöliittymien kautta, ja eri käyttöliittymät eivät voi vaikuttaa toisiinsa.

- **Käyttäjien asetukset**

  **USER_userId=x%y%z**

  Tässä userId on 2–8 merkin mittainen merkkijono, joka kertoo käyttäjän käyttäjätunnuksen. Käyttäjätunnuksessa saa esiintyä vain numeroita (0–9) ja pieniä kirjaimia (a–z). Skandinaavisia kirjainmerkkejä (å, ä, ö) ei sallita. UserId toimii avaimena käyttäjien asetuksia luettaessa. Ei-negatiiviset kokonaisluvut x, y ja z ovat käyttäjän sijaintikoordinaatit. Jokaiseen käyttäjään liittyy täsmälleen yksi tällainen rivi.

- **Tulostinten asetukset**

  **PRINTER_printerId=x%y%z%printer_speed**

  Tässä printerId on 2–10 merkin mittainen merkkijono, joka kertoo tulostimen nimen (esimerkiksi ps8). Tulostimen nimessä saa esiintyä vain numeroita (0–9) ja pieniä kirjaimia (a–z). Skandinaavisia kirjainmerkkejä (å, ä, ö) ei sallita. PrinterId toimii avaimena tulostinten asetuksia luettaessa. Ei-negatiiviset kokonaisluvut x, y ja z ovat tulostimen sijaintikoordinaatit. Tulostimen nopeus annetaan parametrina printer_speed, jonka arvo voi olla kokonaisluku väliltä [1, Integer.MAX_VALUE]. Parametri kertoo, kuinka monta sivua minuutissa tulostin tulostaa. Jokaiseen tulostimeen liittyy täsmälleen yksi tällainen rivi.

- **Analysoijan asetukset**

  Analysoija käyttää seuraavia ajonaikaisia vakiota. Vakioiden vaikutusta selitetään tarkemmin analysoijaa koskevassa luvussa 4.3. Vakioiden arvot esitetään seuraavalla tavalla: VAKIO=arvo. Yleisesti yhtä pienemmät arvot suosivat töiden niputusta ja yhtä suuremmat arvot vähentävät niputusta. Vakiot ja niiden mahdolliset arvot on lueteltu seuraavaksi. Asetustiedostossa voi olla vain yksi rivi jokaista analysoijan vakiota kohti.

  **SAMEUSERMULTIPLIER**

  Tämä on yhdistettyyn kustannukseen vaikuttava kerroin, kun yhdistettävillä nipuilla on yhtenevät tulostajat. Mielekkäät arvot ovat reaalilukuja väliltä [0, 1]. Oletusarvo on "0.3".

  **MAXNUMBEROFPAGES**

  Tämä on sivumäärän alaraja *BIGBUNDLEMULTIPLIER*-kertoimen käytölle. Siis

jos kahden nipun yhdisteen töiden yhteenlaskettu sivumäärä on tätä rajaa suurempi, yhdisteen kustannus skaalautuu *BIGBUNDLEMULTIPLIER*-kertoimen mukaan. Mielekkäät arvot ovat nollaa suurempia kokonaislukuja. Oletusarvo on "100".

**BIGBUNDLEMULTIPLIER**

Tämä on kerroin, jolla yhdistetyn nipun kustannus skaalautuu nipun sivumäärän ylittäessä *MAXNUMBEROFPAGES*-vakion mukaisen rajan. Mielekkäät arvot ovat ei-negatiivisia reaalilukuja. Oletusarvo on "1.1".

**DISTANCELIMIT**

Tämä on raja yhdistettävien nippujen etäisyydelle tulostustyöavaruudessa, jossa etäisyys ei vaikuta yhdisteen kustannukseen. Mitä suuremmaksi etäisyys tästä kasvaa, sitä suuremmalla lineaarisella kertoimella yhdisteen kustannus kasvaa. Jos etäisyys on rajaa pienempi, kustannuskerroin on vastaavasti kustannusta pienentävä. Mielekkäät arvot ovat aidosti positiivisia reaalilukuja. Oletusarvo on "20.0".

**MINDISTANCEMULTIPLIER**

Tämä on kerroin nippujen yhdisteen kustannukselle, kun nippujen etäisyys tulostustyöavaruudessa on nolla. Mielekkäät arvot ovat reaalilukuja puoliavoimelta väliltä [0, 1). Oletusarvo on "0.7".

**SAMEFORCEDPRINTERMULTIPLIER**

Tämä on kerroin kahden nipun yhdisteen kustannukselle silloin, kun yhdistettävät niput sisältävät kumpikin samalle tulostimelle pakotetun työn. Mielekkäät arvot ovat ei-negatiivisia reaalilukuja. Oletusarvo on 1.0.

**COORDINATESTYLE**

Tämä vakio määrää tulostustyöavaruuden akselit. Mahdolliset arvot ovat merkkijonot "submitter", "printer" ja "submitter and printer". Vaihtoehto "submitter" laittaa tulostustöiden koordinaateiksi työn tulostajan fyysisen sijainnin koordinaatit. Vaihtoehto "printer" laittaa tulostustöiden koordinaateiksi työn tulostajaa lähimmän tulostimen fyysisen sijainnin koordinaatit. Vaihtoehto "submitter and printer" laittaa tulostustöiden koordinaateiksi sekä työn tulostajan että häntä lähimmän tulostimen fyysisten sijaintien koordinaatit. Tällöin tulostustyöavaruudessa on kaksinkertainen määrä akseleita verrattuna tulostajien ja tulostinten fyysiseen sijainti-

avaruuteen. Oletusarvo on "submitter".

**FASTPRINTERDISTANCE**

Tämä vakio määrää sen, kuinka kaukaiset tulostimet voivat tulla suositelluiksi tulostimiksi nipulle. Täsmällisemmin sanottuna: tämän vakion arvo on suurin sallittu etäisyys 50-sivuisesta[7] nipusta tulostimelle, jonka nopeus on 50, jotta kyseinen tulostin lisätään kyseisen nipun suositeltujen tulostinten listaan. Tämän vakion arvon perusteella määrätään vastaavat suositeltavuuden etäisyysrajat lineaarisesti myös muilla nopeuksilla toimiville tulostimille ja muun kokoisille nipuille (katso luku 4.3.4). Mielekkäät arvot ovat ei-negatiivisia reaalilukuja. Oletusarvo on "50.0".

---

**Käyttöliittymän asetustiedosto – nippu-userinterface.conf**

Käyttöliittymän asetustiedostossa määritellään **tulostushallinnan ja vastaanoton verkkoparametrit**: DNS-nimet (tai IP-verkko-osoitteet) sekä TCP-porttinumerot. DNS-nimet voidaan kirjoittaa joko lyhyessä muodossa tai täydellisessä FQDN-muodossa. DNS-nimissä sallittuja merkkejä ovat kirjaimet (a–z), numerot (0–9), tavuviiva ja piste. Tämä tiedosto tarvitaan, jotta käyttöliittymä voi ottaa tietoliikenneyhteyden tulostushallintaan ja vastaanottoon. Tiedoston tulee sisältää täsmälleen kaksi tietokokonaisuutta:

- **Vastaanoton sijainti**

  **RECEPTION_LOCATION=aaa:xx**

  Tässä aaa on vastaanoton verkkoaseman DNS-nimi. Sen sijasta voidaan kirjoittaa myös vastaanoton IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255]. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa vastaanoton TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]

- **Tulostushallinnan sijainti**

  **PRINTINGMANAGEMENT_LOCATION=aaa:xx**

  Tässä aaa on tulostushallinnan verkkoaseman DNS-nimi. Sen sijasta voidaan kir-

7 Luku 50 on valittu tämän vakion määrittelyn pohjaksi, sillä sen oletetaan olevan mielekäs, ihmisen helposti hahmotettavissa oleva luku niin tulostimen nopeudessa, nippujen koossa kuin myös tulostinten ja käyttäjien välisissä etäisyyksissä.

joittaa myös tulostushallinnan IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255]. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa tulostushallinnan TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]

---

**Vastaanoton asetustiedosto – nippu-reception.conf**

Vastaanoton asetustiedostossa määritellään **ajoittajan ja analysoijan verkkoparametrit**: DNS-nimet (tai IP-verkko-osoitteet) sekä TCP-porttinumerot. DNS-nimet voidaan kirjoittaa joko lyhyessä muodossa tai täydellisessä FQDN-muodossa. DNS-nimissä sallittuja merkkejä ovat kirjaimet (a–z), numerot (0–9), tavuviiva ja piste. Lisäksi tiedostossa määritellään nippujen lähettämistapa ajoittajaan: esiniputettuna tai alkeistöiksi purettuna Tämä tiedosto tarvitaan, jotta vastaanotto voi ottaa tietoliikenneyhteyden ajoittajaan ja analysoijaan.

Ajoittaja ja analysoija sijaitsevat aina samassa verkkoasemassa, joten niiden verkkoaseman DNS-nimi ja IP-verkko-osoite ovat aina samat. TCP-porttinumerot kuitenkin eroavat. DNS-nimi tai verkko-osoite tulee määriteltyä kahteen kertaan, jotta tähän tiedostoon ei tarvitse määritellä monenlaisia rivisyntakseja. Toisaalta joskus tulevaisuudessa ohjelmistoa saatetaan kehittää siten, että ajoittaja ja analysoija sijaitsevat eri verkkoasemissa.

Tiedoston tulee sisältää täsmälleen kolme tietokokonaisuutta:

- **Analysoijan sijainti**

  **ANALYZER_LOCATION=aaa:xx**

  Tässä aaa on analysoijan verkkoaseman DNS-nimi. Sen sijasta voidaan kirjoittaa myös analysoijan IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255]. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa analysoijan TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]

- **Ajoittajan sijainti**

  **SCHEDULER_LOCATION=aaa:xx**

  Tässä aaa on ajoittajan verkkoaseman DNS-nimi. Sen sijasta voidaan kirjoittaa myös ajoittajan IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255]. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa ajoittajan TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]. Tämän porttinumeron on tarkoitus olla sama kuin Tertun asetuksissa määritelty *tcpServer.External-ClientPort*.

- **Nippujen lähettämistapa ajoittajaan**

  **PRINTJOBSENDING=aaa**

  Tässä aaa on joko "bundle" tai "single". Bundle tarkoittaa töiden lähetystä vastaanoton joka tapauksessa tekemissä odottavissa nipuissa eli esiniputettuna. Käytännössä tämä johtaa suurempaan nippuuntumisasteeseen. Single-vaihtoehto purkaa odottavat niput, ja työt lähetetään ajoittajaan alkeistöinä. Tämä antaa ajoittajalle enemmän valtuuksia niputtaa tai olla niputtamatta.

---

**Analysoijan asetustiedosto – nippu-analyzer.conf**

Analysoijan asetustiedostossa määritellään **tulostushallinnan verkkoparametrit**: DNS-nimi (tai IP-verkko-osoite) sekä TCP-porttinumero. DNS-nimi voidaan kirjoittaa joko lyhyessä muodossa tai täydellisessä FQDN-muodossa. DNS-nimessä sallittuja merkkejä ovat kirjaimet (a–z), numerot (0–9), tavuviiva ja piste. Tämä tiedosto tarvitaan, jotta analysoija voi ottaa tietoliikenneyhteyden tulostushallintaan. Tiedoston tulee sisältää täsmälleen yksi tietokokonaisuus:

- **Tulostushallinnan sijainti**

  **PRINTINGMANAGEMENT_LOCATION=aaa:xx**

  Tässä aaa on tulostushallinnan verkkoaseman DNS-nimi. Sen sijasta voidaan kirjoittaa myös tulostushallinnan IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255].

Analysoija ja tulostushallinta ovat aina samassa verkkoasemassa, joten tässä tulisi aina lukea localhost tai 127.0.0.1. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa tulostushallinnan TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]. Tässä ohjelmistoversiossa tulostushallinnan porttinumero on vakio: 7704.

Analysoijan vakiot voitaisiin määritellä tässä tiedostossa eikä yhteisissä asetuksissa. Nykyinen ratkaisu johtuu siitä, että analysoijan oma asetustiedosto otettiin käyttöön vasta toteutuksen loppuvaiheessa. Tässä tilanteessa haluttiin selvitä vähimmillä muutoksilla.

---

**Tulostushallinnan asetustiedosto – nippu-printingmanagement.conf**

Tulostushallinnan asetustiedostossa määritellään **ajoittajan (Tertun) sisäisten asiakkaiden verkkoparametrit**: DNS-nimi (tai IP-verkko-osoite) sekä TCP-porttinumero. DNS-nimi voidaan kirjoittaa joko lyhyessä muodossa tai täydellisessä FQDN-muodossa. DNS-nimessä sallittuja merkkejä ovat kirjaimet (a–z), numerot (0–9), tavuviiva ja piste. Tämä tiedosto tarvitaan, jotta tulostushallinta voi ottaa tietoliikenneyhteyden ajoittajaan. Tiedoston tulee sisältää täsmälleen yksi tietokokonaisuus:

- **Ajoittajan sijainti**

  **SCHEDULER_LOCATION=aaa:xxx**

  Tässä aaa on ajoittajan verkkoaseman DNS-nimi. Sen sijasta voidaan kirjoittaa myös ajoittajan IP-verkko-osoite. IP-osoite kirjoitetaan neljällä toisistaan pisteillä erotetulla kokonaisluvulla, joiden arvot ovat välillä [0, 255]. Ajoittaja ja tulostushallinta ovat aina samassa verkkoasemassa, joten tässä tulisi aina lukea localhost tai 127.0.0.1. DNS-nimen tai IP-verkko-osoitteen jälkeen tulee kaksoispiste, jota seuraa tulostushallinnan TCP-porttinumero (tässä xx). Porttinumero on kokonaisluku väliltä [1, 65535]. Porttinumero on Tertun sisäisten asiakkaiden käyttämän TCP-portin numero, joka on sama kuin Tertun asetuksissa määritelty *TcpServer.InternalClientPort*. Oletusarvo on 7701.

---

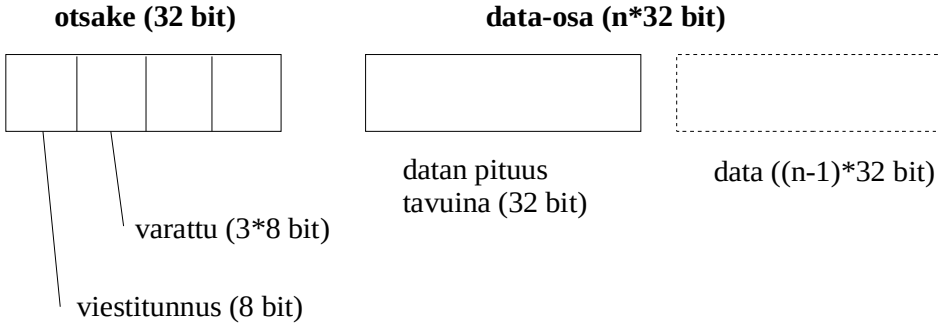
### 3.3.3 Tietoliikenne

Suurin osa sisäisistä liittymistä on toteutettu **tietoliikenteen** avulla. Tietoliikennerajapinta on lähes kaikkien ohjelmistokomponenttien välillä. Tietoliikenneliittymät ovat joustavia, koska komponentit voivat sijaita joko samassa tai eri verkkoasemassa, ja komponentteja voidaan muokata tai korvata täysin toisistaan riippumatta, kunhan ne toteuttavat määritellyn tietoliikenneprotokollan.

Kaikki tietoliikenne tapahtuu **IP-verkossa TCP-yhteyksillä**. Tietoliikenne on toteutettu Tertun mallin mukaan siinä toivossa, että ylläpito olisi helpompaa. Nipussa käytetään hieman muokattuja versioita **Tertun ulkoisten ja sisäisten asiakkaiden tietoliikenneprotokollista** ja niiden toteutukseen liittyvistä luokista.

Tässä luvussa määritellään **tietoliikenneprotokolla**, jota kaikki Nipun komponentit käyttävät. Komponenttien toteutuksen kuvauksessa luvussa 4.5 kerrotaan tarkemmin, miten tietoliikenne on toteutettu Javalla, ja määritellään tietoliikenteeseen liittyvät yhteiskäyttöiset luokat. Luvussa 4 kerrotaan myös jokaisen komponentin kohdalla, miten kyseinen komponentti käyttää tässä luvussa esiteltävää tietoliikenneprotokollaa.

Nipun tietoliikenneprotokollan käyttämien **viestien yleisrakenne** kuvataan kuvassa 3. Viesti jakaantuu kahteen osaan: **otsakkeeseen ja data-osaan**. Otsakkeen koko on aina 32 bittiä. Data-osa sisältää aina vähintään **datan pituus -kentän**, jonka koko on 32 bittiä. **Data-kenttä** jää pois, jos datan pituudeksi on määritelty nolla.




*Kuva 3: Nipun tietoliikenneprotokollan viestin yleisrakenne on täsmälleen sama kuin Tertussa. Kuvaan on merkitty kaikkien osien ja kenttien koot. Tässä n on mielivaltainen nollaa suurempi kokonaisluku.*

Nipun tietoliikenneprotokollaan liittyvät **viestityypit** on merkitty kuvan 4 taulukkoon. Osa viesteistä on kysymys-vastaus-tyyppistä keskustelua, jossa pyyntöä seuraa vastaus, osa on rekisteröitymisen jälkeen ilman erillistä pyyntöä lähetettäviä viestejä. Pyyntö muuttaa ajoittajan ajotilaa (85) ei saa automaattisesti vastausta (C1). Vastaus lähetetään yhteisesti kaikille kuuntelijoille vasta, kun ajotilan muutos on toteutunut. Viestejä 91, E1, 94, E4, 95 ja E5 ei käytetä tässä ohjelmistoversiossa. Ne ovat kuitenkin mukana, jotta analysoijan tietoliikennerajapinta vastaisi täsmälleen sen metodirajapintaa. Viestiä EF käyttää tässä ohjelmistoversiossa vain analysoija vastauksissaan vastaanotolle. Teoriassa myös ajoittaja voisi lähettää EF-viestin, mutta siihen ei ole varauduttu eikä sellaista ole käytännössä havaittu.

| **Viestitunnus** | **viestin kuvaus** | **käyttö** |
|---|---|---|
| 01 | pyyntö työn lisäämiseksi | kä–>va, va–>aj, va–>va |
| 21 | vastaus työn lisäyspyyntöön (01) | kä<–va, va<–aj, va<–va |
| 22 | vastaus työn lisäyspyyntöön (01) | va<–aj |
| 10 | pyyntö palvelunsaannin maksimiviiveen muuttamiseksi | kä–>va |
| 30 | vastaus palvelunsaannin maksimiviiveen muuttamispyyntöön (10) | kä<–va |
| 81 | pyyntö rekisteröityä kuuntelijaksi | kä–>tu, an–>tu, kä–>va |
| A1 | vastaus kuuntelijaksi rekisteröitymispyyntöön (81) | kä<–tu, an<–tu, kä<–va |
| 82 | pyyntö poistaa kuuntelijaksi rekisteröityminen | kä–>tu, an–>tu, kä–>va |
| A3 | vastaus kuuntelijaksi rekisteröitymisen poisto-pyyntöön (82) | kä<–tu, an<–tu, kä<–va |
| B5 | tieto uuden työn siirtymisestä tulostusjonoon | tu–>kä, tu–>an |
| B6 | tieto työn tulostuksen valmistumisesta | tu–>kä, tu–>an |
| 85 | pyyntö muuttaa ajoittajan ajotilaa | kä–>tu, tu–>aj |
| C1 | tieto ajoittajan ajotilan muuttumisesta | tu–>kä, tu–>an, aj–>tu |
| D1 | asetustiedoston sisältö | tu–>kä, tu–>an |

| Viestitunnus | viestin kuvaus | käyttö |
|---|---|---|
| 91 | pyyntö saada nipun koordinaatit | va–>an |
| E1 | vastaus pyyntöön saada nipun koordinaatit (91) | va<–an |
| 92 | pyyntö saada nipun kustannus | va–>an |
| E2 | vastaus pyyntöön saada nipun kustannus (92) | va<–an |
| 93 | pyyntö saada nippujen yhdisteen kustannus | va–>an |
| E3 | vastaus pyyntöön saada nipun yhdisteen kustannus (93) | va<–an |
| 94 | pyyntö yhdistää kaksi nippua | va–>an |
| E4 | vastaus pyyntöön yhdistää kaksi nippua (94) | va<–an |
| 95 | pyyntö muuttaa nippu merkkijonoksi | va–>an |
| E5 | vastaus pyyntöön muuttaa nippu merkkijonoksi (95) | va<–an |
| EF | vastaus virheelliseen pyyntöviestiin | va<–an |

*Kuva 4: Nipun tietoliikenneprotokollaan liittyvät viestityypit. Viesteihin liittyy tyyppikohtainen viestitunnus. Osa viesteistä on identtisiä Tertun protokollien kanssa, jolloin viestitunnus on sama kuin Tertussa. Käyttö-sarake kertoo, minkä komponenttien välillä viesti on käytössä tässä ohjelmistoversiossa (kä=käyttöliittymä, va=vastaanotto, aj=ajoittaja, an=analysoija, tu=tulostushallinta). Käyttöesimerkeissä on mainittu ensin se komponentti, jonka aloitteesta viestinvaihto tapahtuu. Nuolet osoittavat kyseisen viestin kulkusuuntaa. Pienennetyllä kirjasimella merkittyjä viestityyppejä ei käytetä tässä ohjelmistoversiossa.*

Kuvan 4 taulukon viestitunnukset ovat kahdeksanbittisiä heksalukuja väliltä [0, 255]. Tämä arvoalue ei mahdu Javan byte-tietotyyppiin. Viestitunnuksen **x** oikeellinen muuttaminen Javan byte-tietotyypiksi suoritetaan ilmaisulla **(byte)x**. Jos tässä x:n arvo on yli 127, tulee ilmaisun arvoksi negatiivinen luku **x-256**. Luvun bittiesitys pysyy kuitenkin oikeana, joten kun se esimerkiksi täydennetään enemmän merkitsevästä päästä nollabiteillä int-tietotyypiksi, saadaan taas alkuperäinen arvo x.

Palvelupyynnöt **koodataan tavuiksi** ja lähetetään tavupakettina viestin data-osassa. Tietotyypit esitetään XDR-muodossa[8] eli kokonaisluvut esitetään ns. big endian -muodossa[9] ja tavutietoon lisätään tavuja siten, että pituudeksi saadaan alkuperäistä pituutta lähin neljällä jaollinen määrä tavuja. Jos hyötydata ei täytä koko viestiä, data-kentän loppuun lisätään tarvittava määrä täytettä. Viestit eivät sisällä järjestysnumeroita tai aikaleimoja, koska TCP-yhteysprotokolla takaa viestien perille pääsyn sekä järjestyksen säilymisen. Nipun tietoliikenneprotokollan **viestien data-osien muodot** kuvataan kuvan 5 taulukossa.

8 XDR on Internet-standardin RFC1832 mukainen tapa esittää tietotyyppejä.
9 Big endian tarkoittaa sitä, että monitavuisen tietotyypin tavut järjestetään merkitsevimmästä vähiten merkitsevään. Big endianin vastakohta on little endian.

Kokonaisluvut esitetään 32 bitillä etumerkillisinä Javan int-tietotyypin mukaisesti. Reaaliluvut ovat 32-bittisiä IEEE754-standardin mukaisia liukulukuja. Tertun viesteissä käytetään myös Javan tietotyyppiä long (pitkä kokonaisluku), joka on 64-bittinen. Long-tietotyypin esitystavalla ei ole merkitystä, koska Nippu ei lähetä niitä eikä lue Tertun lähettämiä long-arvoja.

Viesteissä esiintyvissä merkkijonoissa käytetään ISO-8859-15-koodausta[10]. Ennen merkkijonojen tavukoodausta tulee tavallinen, neljän tavun mittainen, etumerkillinen kokonaisluku, joka ilmoittaa kyseisen merkkijonon tavukoodauksen tavujen lukumäärän. Merkkijonoiksi koodatut niput tarkoittavat Bundle-luokan *serialize*-metodin tuottamia merkkijonoja. Merkkijonoiksi koodatut tulostustyöt tarkoittavat PrintJob-luokan *serialize*-metodin tuottamia merkkijonoja.

| **Viesti-tunnus** | **datan pituus (tavua)** | **data** |
|---|---|---|
| 01 | n | työn prioriteetti (reaaliluku) + nippu merkkijonoksi koodattuna |
| 21 | 4 | paluuarvo (kokonaisluku) |
| 22 | 4 | paluuarvo (kokonaisluku) |
| 10 | 4 | pyydetty palvelunsaannin maksimiviive (kokonaisluku) |
| 30 | 4 | uusi palvelunsaannin maksimiviive (kokonaisluku) |
| 81 | 4 | rekisteröityjän tyyppi (kokonaisluku; Nipulle lähetettäessä: 1 = pääkäyttäjän käyttöliittymä, 0 = muu; Tertulle lähetettäessä: bitti 1 asetettu = kuunteleva, bitti 2 asetettu = hallitseva) |
| A1 | 8 | kuuntelijan tunniste (kokonaisluku) + kuuntelijan tyyppi (kokonaisluku, 1 = ensimmäinen pääkäyttäjän käyttöliittymä, 0 = muu) |
| 82 | 4 | pyynnön lähettäjän tunniste (kokonaisluku) |

10 ISO-8859-15 eli Latin 9 on Suomessa yleisesti käytetty merkistökoodausstandardi. ISO-8859-15 on muutamia merkkejä lukuun ottamatta yhtenevä vanhemman ISO-8859-1-merkistön kanssa. Huomattavin ero merkistöissä on, että ISO-8859-15 sisältää euro-merkin. Valitettavasti ei voida olla varmoja, että Java-alusta aina tukisi ISO-8859-15-koodausta. Tämä riippuu Java-toteutuksesta. Jos jossain tapauksessa ISO-8859-15:ttä ei tueta, Nippu käyttää ISO-8859-1:tä. Tästä todennäköisesti ei aiheudu ongelmia, koska merkistöt ovat lähes yhtenevät.

| Viesti-tunnus | datan pituus (tavua) | data |
|---|---|---|
| A3 | 4 | paluuarvo (kokonaisluku) |
| B5 | n | tulostimen nimi + työ merkkijonoksi koodattuna |
| B6 | n | työ merkkijonoksi koodattuna |
| 85 | 4 | pyydetty ajotila (kokonaisluku, 3 = pysäytä, 4 = käynnistä) |
| C1 | 16 tai 4 | Tertun lähettämänä: viestin lähetyksen aikaleima (pitkä kokonaisluku), ajoittajan entinen tila (kokonaisluku), ajoittajan uusi tila (kokonaisluku); Nipun lähettämänä: ajoittajan uusi tila (kokonaisluku) |
| D1 | n | asetustiedosto merkkijonona siinä muodossa kuin se on levyltä luettu (kommentit ja tyhjät rivit poistettu, syntaksi tarkistettu) |
| 91 | n | nippu merkkijonoksi koodattuna |
| E1 | n + 2 | nipun tunnus (kokonaisluku) + koordinaattien lukumäärä (= n, kokonaisluku) + 1. koordinaatti (reaaliluku) + .. + n:s koordinaatti (reaaliluku) |
| 92 | n | nippu merkkijonoksi koodattuna |
| E2 | 8 | nipun tunnus (kokonaisluku) + nipun kustannus (reaaliluku) |
| 93 | n | kaksi nippua peräkkäin merkkijonoiksi koodattuna |
| E3 | 12 | ensimmäisen nipun tunnus (kokonaisluku) + toisen nipun tunnus (kokonaisluku) + yhdisteen kustannus (reaaliluku) |
| 94 | n | kaksi nippua peräkkäin merkkijonoiksi koodattuna |
| E4 | n | ensimmäisen nipun tunnus (kokonaisluku) + toisen nipun tunnus (kokonaisluku) + yhdistenippu merkkijonoksi koodattuna |
| 95 | n | nippu merkkijonoksi koodattuna |
| E5 | n | nipun tunnus (kokonaisluku) + ihmisen tulkittavissa oleva nipun merkkijonokuvaus |
| EF | 4 | valinnainen syy virheeseen (kokonaisluku) |

*Kuva 5: Nipun tietoliikenneprotokollan viestityyppien data-osien muodot. Plus-merkki (+) tarkoittaa tietotyyppien tavuesitysten liittämistä peräkkäin. Pienennetyllä kirjasimella merkittyjä viestityyppejä ei käytetä tässä ohjelmisto-versiossa.*

Nipun tietoliikenneprotokolla muistuttaa läheisesti Tertun käyttämiä tietoliikenneprotokollia, joten joihinkin ratkaisuihin liittyviä lisätietoja kannattaa etsiä Tertun määrittelydokumentin luvusta 4.2, suunnitteludokumentin luvusta 4 sekä manuaalin luvusta 5. Tertun manuaalissa sivuilla 23–28 on myös C-kielinen protokollan käyttöesimerkki.

Tähän ohjelmistoversioon ei sisälly sellaista sisäistä liittymää, jolla **tulostettavan tiedoston sisältö** siirrettäisiin käyttöliittymästä tulostushallintaan. Liittymä ei ole välttämätön, koska tämä ohjelmistoversio vain emuloi tulostusta. PrintJob-tietorakenteeseen sisältyvät ainoastaan työn sivumäärä ja nimi, joka vastaa tulostettavan tiedoston nimeä. Ohjelmistoa voidaan myöhemmin kehittää esimerkiksi siten, että käyttöliittymä siirtää tiedoston erillisenä komponenttina toteutettavaan tietovarastoon, josta tulostushallinta lukee tiedoston. Ei ole nimittäin tarpeellista syöttää käyttäjän tiedoston sisältöä ajoittajaan.

Tertun tietoliikenneprotokollan **viestien enimmäiskoko** on oletuksena 128 tavua, joka ei riitä Nipussa käytettäville viesteille. Tertun asetuksissa on mahdollista muuttaa tätä arvoa, mutta havaittiin, että Terttu ei toimi muutettujen asetusten mukaan. Nipun mukana toimitettavan Tertun lähdekoodia onkin muokattu siten, että se toimii myös asetustiedostossa määritellyllä suuremmalla pakettikoolla. On mahdollista, että tämäkään arvo ei jossain tilanteessa riitä, jolloin täytyy kasvattaa Tertun asetuksissa määriteltävää *tcpserver.MaxMessageSize*-parametria.

### 3.3.4 Luokkarajapinnat

Sekä sisäisiin että ulkoisiin liittymiin kuuluu Javalle tyypillisiä luokkarajapintoja. **Sisäiset** luokkarajapintaliittymät ovat Tertun ajoittajan liittymät analysoijaan ja tulostushallintaan. **Ulkoisia** luokkarajapintaliittymiä puolestaan ovat liittymät Log4j-komponenttiin sekä Jbuilder-ohjelman luokkatiedostoihin. Log4j:ta käyttävät kaikki Nipun komponentit. Käyttöliittymä käyttää Jbuilderin luokkatiedostoja.

**Log4j** on lokimerkinnöistä huolehtiva komponentti, jota käytetään Nipun ajonaikaisten viestien kirjaamiseen. Log4j on osa Apache Jakarta -projektia[11]. Log4j:n käyttö tapahtuu kahden luokan kautta. Luokat ovat *Logger* ja *PropertyConfigurator*, ja ne sijaitsevat pak-

11 Tarkka kuvaus Log4j:sta ja sen käytöstä on projektin verkkosivuilla osoitteessa http://jakarta.apache.org/log4j/docs/index.html. Myös Nipun osana toimiva Terttu käyttää Log4j-komponenttia.

kauksessa *org.apache.log4j*. Luokkien määrittelyt Nipun kannalta oleellisilta osin ovat seuraavat:

public class PropertyConfigurator
    public static void configure(String configFilename)
        Metodi lukee ja ottaa käyttöön parametrina annettavasta tiedostosta Log4j-komponentin asetukset.

public class Logger
    public static Logger getLogger(String name)
        Metodi palauttaa annettua nimeä vastaavan olion, jolla lokiin voi kirjata merkintöjä.
    public void debug(Object message)
        Metodi kirjaa lokiin DEBUG-tasoisen viestin.
    public void info(Object message)
        Metodi kirjaa lokiin INFO-tasoisen viestin.
    public void warn(Object message)
        Metodi kirjaa lokiin WARN-tasoisen viestin.
    public void error(Object message)
        Metodi kirjaa lokiin ERROR-tasoisen viestin.
    public void fatal(Object message)
        Metodi kirjaa lokiin FATAL-tasoisen viestin.

Log4j:n asetustiedostossa voidaan määritellä, miten lokiviestit muotoillaan ja minne ne lähetetään, sekä myös minkätasoisia viestejä ylipäänsä otetaan huomioon. Nipun mukana toimitettavassa kokoonpanossa viestit kirjautuvat sekä näytölle että lokitiedostoon.

## 3.4 Toiminnekuvaus

Nipun keskeisin toiminnallisuus on samankaltaisten tulostustöiden yhdistäminen nipuiksi yhteistä suoritusta varten. **Ajoittaja** tekee päätöksen töiden niputtamisesta **analysoijalta** saamiensa tietojen perusteella. Analysoija puolestaan suorittaa niputuksen ajoittajan pyynnöstä. Jos nippu tai alkeistyö on mahdollista tulostaa usealla tulostimella, vasta **tulostushallinta** lopullisesti päättää, mitä tulostinta käytetään. **Tulostustyön käsittely** tapahtuu

luvussa 3.2 esitettyjen ohjelmistokomponenttien avulla pääpiirteissään seuraavasti:

1. Kun käyttäjä on painanut **käyttöliittymän** painiketta "tulosta" tai käynnistänyt satunnaistulosteet, käyttöliittymä luo käyttäjän antamista tiedoista PrintJob-olion. Sen jälkeen luodaan tyhjä Bundle-olio, johon ensin luotu PrintJob-olio liitetään. Bundle-olio (eli yksitöinen nippu) lähetetään tietoliikenneyhteydellä vastaanottoon.
2. **Vastaanotossa** Bundle-olio odottaa mahdollisia muita samaan nippuun todennäköisesti sopivia töitä tietyn ajan. Bundle-olioiden nippuuntumistodennäköisyyttä arvioidaan kysymällä analysoijalta tarvittavia tietoja tietoliikenneyhteydellä. Vastaanotto ei tee itse työlle muuta kuin puskuroi sen sopivaksi ajaksi ja lähettää tietoliikenneyhteydellä ajoittajaan. Vastaanotto kuitenkin lähettää töitä ajoittajaan valmiiksi niputettuna, jos vastaanoton asetuksissa on esiniputus-parametri päällä.
3. **Ajoittaja** päättää töiden eli Bundle-olioiden yhdistämisestä eli niputuksesta suuremmiksi Bundle-olioiksi. Ajoittaja kysyy **analysoijalta** kunkin työn suorituskustannusta yksittäin tai yhdistettynä muihin töihin. Jos ajoittaja päättää yhdistää töitä, se välittää yhdistämispyynnön analysoijalle. Analysoija suorittaa töiden yhdistämisen, ja työt tulevat takaisin ajoittajaan niputettuna. Ajoittaja saattaa jatkaa yhdistelyä isommiksi nipuiksi. Kun suorituskustannusta ei enää voida niputtamalla pienentää, ajoittaja lähettää työn eli Bundle-olion tulostushallintaan tulostusta varten.
4. **Tulostushallinta** päättää lopullisen tulostimen. Päätös tehdään nipun mukana seuraavan optimaalisten tulostinten listan sekä tulostajien ja tulostimien koordinaattien perusteella. Työ sijoitetaan jonkin vapaan tulostimen jonoon, ja tästä ilmoitetaan käyttöliittymille.
5. Kun työ on tulostettu, tulostushallinta ilmoittaa siitä käyttöliittymille.
6. **Käyttöliittymän** tulostuslokiin ja omat työt -näkymään jää tieto työn valmistumisesta ja valitun tulostimen nimi.

**Käyttöliittymän osat** liittyvät eri ohjelmistokomponentteihin seuraavasti:

- Perus- ja pääkäyttäjän liittymien tulostinvalintamahdollisuudet perustuvat tulostushallinnan tietoliikenteellä välittämän asetustiedoston sisältämään listaan mahdollisista tulostimista. Tulostushallinta lukee asetustiedoston levyltä.
- Peruskäyttäjän liittymän "omat työt" -kentän tiedot perustuvat tulostushallinnan tietoliikenteellä lähettämiin tietoihin kyseisestä käyttöliittymästä lähetettyjen töiden etenemisestä.

- Pääkäyttäjän liittymän käyttäjätunnus-valinta perustuu tulostushallinnan tietoliikenteellä välittämän asetustiedoston sisältämään listaan järjestelmän käyttäjätunnuksista.
- Pääkäyttäjän liittymässä säädettävä maksimiviive palvelun saannille on vastaanoton ominaisuus. Käyttäjän asettama arvo päivittyy vastaanottoon tietoliikenneyhteydellä.
- Pääkäyttäjän ajoituksenpysäytyspainike toimii tulostushallinnan kautta, koska tulostushallinnalla on oltava ensisijainen oikeus pysäyttää ja käynnistää ajoittaja. Tieto painikkeen käytöstä välittyy tulostushallintaan tietoliikenneyhteydellä, ja tulostushallinta tiedottaa ajoittajan käynnistämisestä ja pysäyttämisestä käyttöliittymille tietoliikenneyhteydellä.
- Visualisoinnin jononäkymä ja tulostusloki ovat tulostushallinnan tilatietoja. Jononäkymä sisältää hetkellisen tilanteen. Tulostuslokiin jäävät tiedot töiden sijoittumisesta jonoihin ja tulostumisesta. Tulostushallinta lähettää tiedot käyttöliittymille tietoliikenneyhteydellä.
- Peruskäyttäjän tulosta -painike ja pääkäyttäjän satunnaistulostepainike käynnistävät tulostustyön ja nipun tietorakenteen luonnin ja lähetyksen käyttöliittymästä vastaanottoon tietoliikenneyhteydellä.

# 4 Komponenttien toteutuksen kuvaus

Tässä luvussa käsitellään yksityiskohtaisesti Nipun ohjelmistokomponentit: käyttöliittymä, vastaanotto, analysoija ja tulostushallinta. Lisäksi käsitellään tietoliikenteen Java-toteutusta tietoliikennekomponentti-luvussa. Jokaiseen ohjelmistokomponenttiin liittyvät seuraavat aliluvut:

1. tehtävät ja toiminta,
2. liittymät,
3. tietorakenteet ja
4. toimintalogiikka.

Toimintalogiikkaa kuvataan muun muassa pseudokoodin avulla. Näiden alilukujen jälkeen seuraa tarvittaessa muita, sopivasti otsikoituja alilukuja.

## 4.1 Käyttöliittymä

### 4.1.1 Käyttöliittymän tehtävät ja toiminta

Peruskäyttäjä lähettää tavalliset tulostustyöt käyttöliittymän kautta. Pääkäyttäjä puolestaan voi sekä lähettää tulostustöitä että suorittaa hallinta- ja testaustehtäviä käyttöliittymän avulla. Perus- ja pääkäyttäjä voivat kumpikin tarkastella tulostinten tiloja.

**Peruskäyttäjän liittymässä** annetaan käyttäjätunnus, työn nimi ja sivumäärä. Käyttöliittymä ei käynnisty vaan tulostaa varoitustekstin, jos käyttäjä yrittää käynnistää käyttöliittymän ilman oikeellista käyttäjätunnusta tai jos vastaanottoon ei saada yhteyttä. Virheilmoitus annetaan myös, jos käyttäjä kirjoittaa tulosteen nimen tai sivumäärän väärin (esimerkiksi negatiivinen sivumäärä, työn nimi yli 255 merkkiä tai sisältää kiellettyjä merkkejä). Omien tulosteiden näkymässä luetellaan vain kyseisestä käyttöliittymästä lähetettyjen töiden nimet sekä niiden tilat.

**Pääkäyttäjän liittymä** tarkistaa, onko muita pääkäyttäjän liittymiä käynnissä. Pääkäyttäjän hallintatoiminnot (palvelunsaannin maksimiviiveen säätö ja ajoituksen pysäyttäminen) toimivat vain ensimmäisenä käynnistetyssä pääkäyttäjän liittymän ilmentymässä –

muissa tapauksissa toimintojen käyttö estetään. Näin varmistetaan, että vain yksi pääkäyttäjä kerrallaan voi suorittaa hallinnollisia tehtäviä.

**Visualisointiliittymä** tulee sekä perus- että pääkäyttäjän liittymän ympärille. Siinä näytetään tulostinkohtaiset tiedot, kuten tulostimen nimi, tulostumassa olevan työn nimi ja tulostusjonon tila. Nämä tiedot päivitetään välittömästi, kun tulostushallinnasta saapuu uutta tietoa. Visualisointiliittymä sisältää myös tulostuslokin, johon kirjataan kaikkien käyttäjien tulosteiden nimet ja tilat.

### 4.1.2 Käyttöliittymän liittymät

Käyttöliittymä lukee käynnistyessään **asetustiedostosta** tulostushallinnan ja vastaanoton verkkoparametrit. Asetustiedoston tulee noudattaa liittymäkuvauksen luvussa 3.3.2 määriteltyä syntaksia.

Käyttöliittymä saa tiedot järjestelmän käyttäjistä ja tulostimista tulostushallinnalta seuraavia **tietoliikenneyhteysluokkia** käyttäen: *UserInterfacePrintingManagementProtocol* ja *UserInterfacePrintingManagementServic*e. Käyttöliittymä lähettää tulostustyöt vastaanotolle kutsumalla tietoliikenneyhteysluokkia *UserInterfaceReceptionProtocol* ja *UserInterfaceReceptionService.* Tietoliikenneyhteysluokat sijaitsevat pakkauksessa *nippu.communications.*

Kun käyttöliittymä luodaan, ensin käyttöliittymä rekisteröi itsenä tulostushallinnan kuuntelijaksi *UserInterfacePrintingManagementProtocol.sendListenerRegistration*-metodia käyttäen. Käyttöliittymä saa vastauksena yksilöivän tunnisteen *UserInterfacePrintingManagementService.registrationAccepted*-metodia käyttäen. Käyttöliittymä saa käyttäjiä ja tulostimia koskevat asetustiedot tulostushallinnasta *UserInterfacePrintingManagementService.configuration*-metodilla. Käyttöliittymä rekisteröi itsensä myös vastaanoton kuuntelijaksi *UserInterfaceReceptionProtocol.sendListenerRegistration*-metodilla.

Kun käyttäjä lähettää tulostustöitä (peruskäyttäjän tulostustöitä tai pääkäyttäjän satunnaistöitä), käyttöliittymä lähettää tulostustyöt Bundle-olioina vastaanotolle *UserInterfaceReceptionProtocol.sendNewBundle*-metodia käyttäen.

Käyttöliittymä päivittää visualisointinäkymää tulostushallinnan lähettämien viestien

mukaan *UserInterfacePrintingManagementService.printJobQueued*- ja *printJobPrinted*-metodeja käyttäen

Pääkäyttäjä voi pysäyttää tai käynnistää ajoituksen käyttöliittymän kautta. Tällöin käyttöliittymä kutsuu *UserInterfacePrintingManagementProtocol.sendSchedulerStateChangeQuery*-metodia. Pääkäyttäjä voi myös vaihtaa palvelunsaannin maksimiviivettä. Tällöin käyttöliittymä kutsuu *UserInterfaceReceptionProtocol.sendMaxServiceDelayChangeQuery*-metodia. Kun käyttöliittymä suljetaan, se irrottautuu tulostushallinnasta *UserInterfacePrintingManagementProtocol.sendListenerUnregistration*-metodilla ja vastaanotosta *UserInterfaceReceptionProtocol.sendListenerUnregistration*-metodilla.

### 4.1.3 Käyttöliittymän tietorakenteet

Käyttöliittymä ei käytä mitään erityisiä tietorakenteita tietokuvauksessa mainittujen luokkien lisäksi.

### 4.1.4 Käyttöliittymän toimintalogiikka

Käyttöliittymän Swing-komponenttikaavio on liitteessä 2. Käyttöliittymä sisältää seuraavat luokat:

---

**Nippu**

Sovelluksen pääluokka. Nippu rekisteröityy vastaanottoon. Nippu rekisteröityy myös tulostushallintaan ja saa komponentin tunnisteen, käyttäjälistan ja tulostinlistan tulostushallinnalta käyttäen *UserInterfacePrintingManagementProtocol*- sekä *UserInterfacePrintingManagementService*-luokkia. Käyttäjälista perustuu *NippuUser*-taulukkoon ja tulostinlista *NippuPrinter*-taulukkoon. Nippu luo *Frame1*-ilmentymän.

- **Kentät**
  - static Vector userList

    *NippuUser*-olioiden taulukko
  - static Vector printerList

    *NippuPrinter*-olioiden taulukko

- static boolean superUser

  pääkäyttäjän lippu
- static String currentUser

  käyttöliittymän käynnistäjän käyttäjätunnus

- **Metodit**
  - Nippu()

    Konstruktori. Konstruktorissa luodaan *Frame1*-ilmentymä ja rekisteröidytään tulostushallintaan ja vastaanottoon.
  - main(String[] args)

    Päämetodi. Metodille voi antaa kaksi parametria: ensimmäinen on käyttäjätunnus ja toinen on pääkäyttäjän parametri "p".
  - static NippuUser createNippuUser(String userID, String userPos)

    Metodi luo yhden *NippuUser*-ilmentymän.
  - static NippuPrinter createNippuPrinter(String printerID, String printerPos)

    Metodi luo yhden *NippuPrinter*-ilmentymän.

---

**Frame1**

Luokassa luodaan kaikki Java Swing -komponentit.

- **Kentät**
  - kaikki Swing-komponentteja ilmentävät oliot
  - boolean showSuperUser

    pääkäyttäjän lippu, jonka arvo tulee *Nippu*-luokasta
  - Vector userList

    *NippuUser*-olioiden taulukko, jonka arvo tulee asetustiedostosta
  - Vector printerList

    *NippuPrinter*-olioiden taulukko, jonka arvo tulee asetustiedostosta
  - String userID

    käyttöliittymän käynnistäjän käyttäjätunnus, jonka arvo tulee *Nippu*-luokasta
  - int jobCounter

    juokseva numero, joka erottaa saman käyttöliittymän tulostustyöt

- int componentID

  käyttöliittymän tunniste, jonka arvo tulee tulostushallinnasta käyttöliittymän rekisteröityessä

- **Metodit**
  - public Frame1(String currentUser, boolean superUser, Vector nippuUserList, Vector nippuPrinterList)

    konstruktori
  - private void jbInit() throws Exception

    Swing-komponenttien alustus, pääkäyttäjän lippu otetaan huomioon
  - erilaisia AWT:n tapahtumametodeja

    Nämä luokat hoitavat Swing-komponenttien erilaisia toimintoja. Tarkemmat tapahtumametodit ovat seuraavat:
  - void printButton_actionPerformed(ActionEvent e)

    Luodaan PrintJob- ja Bundle-ilmentymiä, jotka lähetetään vastaanotolle tietoliikenteen metodilla.
  - void sRandomJobButton_actionPerformed(ActionEvent e)

    Käynnistetään *RunRandomJob*-säie pääkäyttäjän satunnaistöille tai pysäytetään se sen mukaan, onko satunnaistöiden luonti käynnissä vai ei.
  - void sStopSchedulerButton_actionPerformed(ActionEvent e)

    Lähetetään tulostushallinnalle ajoittajan ajotilan mukaan joko ajoittajan käynnistys- tai pysäytyspyyntö

---

**RunRandomJob**

Tämä luokka luo pääkäyttäjän satunnaistulostustyöt (PrintJob- ja Bundle-tietorakenteet) ja lähettää ne vastaanotolle tietoliikennekomponentin metodilla.

- **Kentät**
  - public Boolean runme

    Kun *runme* on TRUE, aloitetaan *RunRandomJob*-säie. Muuten säie pysäytetään.

- **Metodit**
  - RunRandomJob(Vector allUsers, int sizeMin, int sizeMax,
    int frequencyMin, int frequencyMax, String userID,
    String printerID, int listenerID)

    Tässä konstruktorissa saadaan kaikki satunnaistöiden määritykset *Frame1*-luokalta.
  - void run()

    Säikeen päämetodi. Tämä metodi suoritetaan, kun Boolean-parametri *runme* on TRUE. Ensin arvotaan satunnaiskäyttäjätunnuksia, sivumääriä ja lähetystiheyksiä. Sen jälkeen lähetetään tulostustöitä Bundle-olioina vastaanotolle.
  - void stopJee()

    Asetetaan FALSE-arvo *runme*-parametrille, jolloin käytännössä pysäytetään *RunRandomJob*-säie.
  - Seuraavilla apumetodeilla tuotetaan sivumäärän, lähetystiheyden ja käyttäjätunnuksen satunnaisarvoja:

    int checkPageSize(int minPageSize, int maxPageSize)
    int checkFrequency(int minFrequency, int maxFrequency)
    NippuUser checkUser(String userID)

---

**GetJobID**

Tällä luokalla annetaan tulostustöille yksilöivät tunnisteet, joiden avulla erotetaan saman käyttöliittymän tulostustyöt. Tunnisteena käytetään juoksevaa kokonais-lukunumerointia.

- **Kentät**
  - static volatile int jobID
- **Metodit**
  - static synchronized int grantJobID()

    JobID:tä kasvatetaan aina yhdellä.

---

Seuraavaksi esitellään käyttöliittymän toimintalogiikkaa pseudokoodin avulla.

**Käyttöliittymän käynnistys**

```
if (käynnistetty oikein ja vastaanotto on käynnissä)
      switch (käynnistysparametri)
             case null: luodaan peruskäyttäjän liittymä ja
                      visualisointiliittymä
             case p: if (olemme ensimmäinen pääkäyttäjä)
                    luodaan pääkäyttäjän liittymä ja
                      visualisointiliittymä
             else
                    luodaan pääkäyttäjän liittymä, jossa funktio-
                      painikkeiden käyttö on estetty, ja
                      visualisointiliittymä
else
      switch (virheet)
             case väärä käyttäjätunnus: virheilmoitus väärästä
                      käyttäjätunnuksesta
             case vastaanotto ei käynnissä: virheilmoitus puuttuvasta
                      vastaanotosta
```

**Peruskäyttäjän liittymän luominen ja käyttö**

```
luo peruskäyttäjän liittymä oletusarvojen mukaan

when (tulosta-painiketta on painettu)
      if (tulosteen nimi on oikea ja sivumäärä ei ole negatiivinen)
             lähetä käyttäjätunnus, tulosteen nimi, sivumäärä ja
               pakotettu tulostimen nimi tietoliikennekomponentille
      else
             if (tulosteen nimessä on kiellettyjä merkkejä)
                    lähetä virheilmoitus tulosteen nimestä
             if (sivumäärä negatiivinen)
                    lähetä virheilmoitus sivumäärästä

      päivitä käyttäjän omien tulosteiden näkymä

      while (uudet tiedot ovat tulleet tulostushallinnasta)
             if (pakotettu tulostin ei ole saatavilla)
                    ilmoitetaan käyttäjälle
             päivitetään käyttäjän omien tulosteiden näkymä
```

**Pääkäyttäjän liittymän luominen ja käyttö**

```
luo peruskäyttäjän liittymä ja sen logiikka

luo pääkäyttäjän liittymä oletusarvojen mukaan

while (käyttöliittymä on käynnissä)
      while (käynnistä satunnaistyöt –painiketta on painettu)

             if (painike on nimeltään "Käynnistä satunnaistyöt")
                    arvotaan satunnaiskäyttäjätunnuksia, sivumääriä
                      sekä lähetystiheyksiä ja jälleen lähetetään
                      tulostustöitä vastaanotolle tilin kautta sekä
```

```
                    päivitetään painike "lopeta satunnaistyöt"
                    –nimiseksi

            if (painike on nimeltään "Lopeta satunnaistyöt")
                    lopeta satunnaistyöt ja päivitä painike "käynnistä
                      satunnaistyöt" –nimiseksi

      while (pysäytä ajoittaja –painiketta on painettu)
             if (painike on nimeltään "pysäytä ajoittaja")
                    lähetä asiaan kuuluvat tiedot tilille ja päivitä
                      painike "käynnistä ajoittaja"–nimiseksi

             if (painike on nimeltään "käynnistä ajoittaja")
                    lähetä asiaan kuuluvat tiedot tilille ja päivitä
                      painike "pysäytä ajoittaja"–nimiseksi

      while (uudet tiedot ovat tulleet tulostushallinnasta)
             päivitetään visualisointinäkymä
```

## 4.2 Vastaanotto

### 4.2.1 Vastaanoton tehtävät ja toiminta

Vastaanotto on olio, joka käynnistetään itsenäiseksi prosessiksi. Vastaanoton tehtävä on **tehostaa samankaltaisten tulostustöiden yhdistelyä**. Se pitää kirjaa odottavista tulostustyönipuista. Uuden tulostustyön saapuessa vastaanotto tarkistaa suoraan analysoijalta, tulisiko tämä työ todennäköisesti yhdistymään vastaanotossa jo olevien, odottavien nippujen töihin. Jos näin näyttää käyvän, tulostustyö siirretään sopivaan odottavaan nippuun. Muussa tapauksessa se muodostaa uuden odottavan nippunsa. Niput odottavat puolittuvan summasarjan mukaisella tavalla (katso vaatimusdokumentin luku 3.1.2).

Tulostustyöt **lähetetään ajoittajalle**, kun odotusaika on kulunut. Jos ajoittaja ei ole käynnissä tai lähetys muusta syystä epäonnistuu, työt jäävät vastaanottoon, joka yrittää lähetystä myöhemmin uudelleen. On todettu, että kun ajoittajalla ei ole tulostustöitä ja sille lähetetään uusia töitä, ensimmäinen työ päätyy aina suoritukseen, ennen kuin seuraavia ehditään lähettää. Tämän vuoksi ajoittaja ei koskaan yhdistä ensimmäistä työtä muihin töihin. Koska tämä on ristiriidassa Nipun vaatimusmäärittelyn kanssa, on vastaanoton toiminta säädettävissä sellaiseksi, että se lähettää työt ajoittajalle valmiiksi niputettuina. Näin voidaan varmistaa samankaltaisten töiden niputus. Jos töitä ei niputeta valmiiksi vastaanotossa, ajoittaja suorittaa töiden varsinaisen niputtamisen riippumatta vastaanoton tekemästä niputuksesta.

### 4.2.2 Vastaanoton liittymät

Vastaanoton pääkomponentti on ***Reception*-luokka**, jonka kautta tietoliikennekomponentti voi ottaa siihen yhteyden. Luokka määritellään seuraavasti:

**Reception-luokka**

- **Määrittely**
  public class Reception
- **Peruskonstruktori**
  public Reception()
- **Julkiset metodit**
  public boolean addJob (Object newJob)
    Metodi lisää uuden tulostustyön. Työ lisätään joko olemassa olevaan nippuun tai uudeksi omaksi nipukseen. Metodi palauttaa true, jos lisäys onnistui, ja false, jos lisäys epäonnistui.
  public int getNumberOfJobs()
    Metodi palauttaa vastaanotossa olevien töiden lukumäärän.
  public int getNumberOfBundles()
    Metodi palauttaa vastaanotossa olevien nippujen lukumäärän.
  public String toString()
    Metodi palauttaa lyhyen, ihmisen luettavaksi tarkoitetun kuvauksen vastaanoton tilasta.

Vastaanotto käyttää **Nipun tietoliikennerajapintaa** (katso luku 4.5) yhteydenpitoon käyttöliittymien, analysoijan ja ajoittajan kanssa. Vastaanottoon liittyvä tietoliikennekomponentti avaa TCP-portin 7702, johon käyttöliittymät voivat ottaa yhteyden. Käyttöliittymät voivat lähettää työnlisäyspyynnön viestillä 01 (katso tietokuvauksen luku 3.3.3), jonka vastaanotto kuittaa vastausviestillä 21. Vastaanotto lähettää analysoijalle kustannuskyselyjä viesteillä 92 ja 93. Analysoija vastaa viesteillä E2, E3 tai EF. Vastaanotto lähettää niput eteenpäin ajoittajalle viestillä 01 ja odottaa vastauksena viestiä 21 tai 22. Käyttöliittymien ja vastaanoton välisessä viestinnässä käytetään vielä rekisteröitymisviestejä 81, 82, A1 ja A3 sekä palvelunsaannin maksimiviiveen hallintaviestejä 10 ja 30.

Vastaanotolla on myös tietoliikenneyhteys itseensä. Tällä yhteydellä vastaanotto siirtää

ajoittajan hylkäämät työt takaisin odottavien töiden puskuriin, jotta työt voisivat edelleen muodostaa odottavia nippuja muiden töiden kanssa. Yhteys käyttää samoja tietoliikenneluokkia kuin käyttöliittymä ollessaan yhteydessä vastaanottoon. Vastaanotto vain lähettää itselleen viestejä. Omia vastausviestejään vastaanotto ei kuitenkaan käsittele.

Vastaanotto lukee oman komponenttikohtaisen asetustiedostonsa, joka sisältää analysoijan ja ajoittajan tietoliikenneparametrit sekä töiden lähetystapa -parametrin. Asetustiedoston syntaksin tulee olla tietokuvauksen mukainen.

### 4.2.3 Vastaanoton tietorakenteet

Niput toteutetaan *Reception*-luokassa ***RBundle*-tietorakenteena**. *RBundle* on olio, joka sisältää Bundle-tietorakenteen ja ajastimen. Vastaanotto säilyttää *RBundle*-olioita dynaamisessa Vector-tietorakenteessa.

Vastaanotolla on **hajautusmenetelmällä** toteutettu kuvaus rekisteröityneiden käyttöliittymien tunnuksilta protokollaolioihin. Kukin protokollaolio on yhteys kyseiseen käyttöliittymään. Kuvaus on toteutettu *java.util.HashMap*-oliona, koska siinä on vakioaikaiset lisäys- ja haku-operaatiot. Näin vastaanotto voi palvella tehokkaasti useita käyttöliittymiä samaan aikaan.

### 4.2.4 Vastaanoton toimintalogiikka

Vastaanotto koostuu esitellyistä *Reception*- ja *RBundle*-luokista sekä *RTimerTask*-luokasta, joka laajentaa Javan luokkaa *java.util.TimerTask. RTimerTask*-luokka vastaa sitä toimintoa, joka tulee suorittaa tiettyyn *RBundle*-olioon liittyvän ajastimen ajan kuluttua loppuun. *RTimerTask*-luokan konstruktorille annetaan *RBundle*-olio, johon ajastintehtävä liittyy, sekä viite *Reception*-olioon, jota voidaan pyytää lähettämään kyseinen *RBundle*-olio eteenpäin ajoittajalle.

Kun uusi työ saapuu vastaanottoon, **analysoijalle lähetetään kysely** uuden työn kustannuksesta, jokaisen odottavan nipun kustannuksesta, sekä uuden työn ja kunkin odottavan nipun yhdisteen kustannuksesta. Kun kaikkiin kyselyihin on saatu vastaus, tehdään päätös uuden työn liittämisestä odottaviin nippuihin. Se tehdään samalla logiikalla kuin Tertun ajoittajassa: uusi työ yhdistetään siihen odottavaan nippuun, jolla yhdisteen kustannus on

mahdollisimman paljon pienempi kuin uuden työn ja odottavan nipun yhteenlaskettu kustannus yksittäin tulostettuina. Jos kaikissa tapauksissa yhdistäminen lisäisi kustannusta, luodaan uudesta työstä oma odottava nippu. Nipun ajastimen odotusaika asetetaan uudestaan aina, kun siihen lisätään uusi työ. Odotusaika on puolet siitä, mitä se oli edellisen odotuskerran alussa.

Vastaanotto tarvitsee **oman komponenttitunnuksen** voidakseen luoda uusia nippuja Bundle-luokan metodilla *breakBundle*. Varsinaisia komponenttitunnuksia jakaa tulostushallinta käyttöliittymälle ja analysoijalle niiden rekisteröityessä tulostushallintaan. Vastaanotto ei rekisteröidy, joten sille on varattu oma komponenttitunnus, joka on nolla. Tämä valinta perustuu oletukseen, että tulostushallinta aloittaa tunnusten numeroinnin yhdestä. Lisäksi nippujen tunnusten yksikäsitteisyyden takaamiseksi täytyy olettaa, että yhteen ajoittajaan on yhteydessä vain yksi vastaanotto. Nippujen tunnusten yksikäsitteisyys ei kuitenkaan ole välttämätön vaatimus Nippu-järjestelmän toiminnan kannalta. Nippujen päästyä vastaanottoon asti on niiden tunnuksilla merkitys vain vastaanotolle itselleen. Ajoittaja, tulostushallinta ja lopuksi käyttöliittymä eivät enää käytä nipun tunnusta.

Vastaanoton toimintaa esitetään alla pseudokoodin avulla:

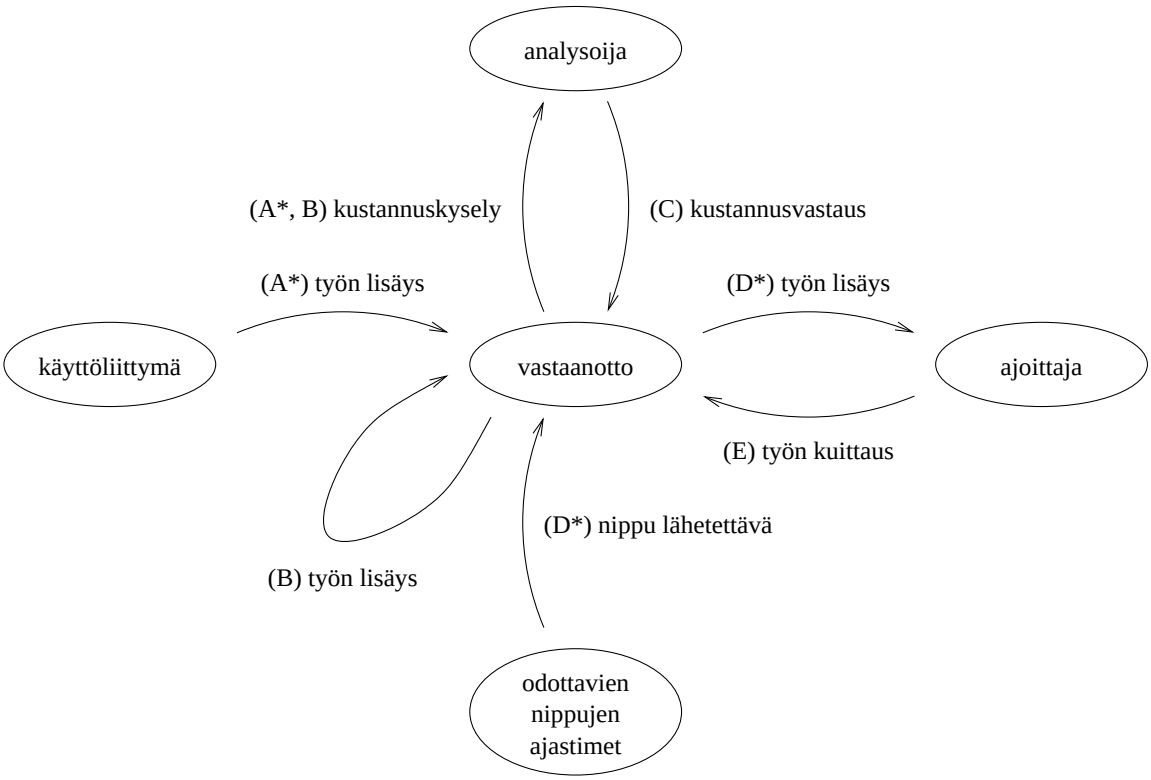
**Uuden työn lisäys:**

```
lähetä analysoijalle kysely uuden työn kustannuksesta

foreach (odottava nippu)
      lähetä analysoijalle kysely odottavan nipun kustannuksesta
      lähetä analysoijalle kysely odottavan nipun ja uuden työn
        yhdisteen kustannuksesta

odota vastauksia

if (työ ei sovi yhteenkään odottavaan nippuun)
      tee työstä oma nippunsa
      aseta uuden nipun ajastimelle oletusodotusaika
      talleta uusi nippu Vectoriin
else
      lisää työ edullisimpaan nippuun
      muuta nipun ajastimen odotusaikaa puolittuvan summasarjan
        mukaisesti
```

**Ajastimen ajan loputtua:**

```
if (esiniputus käytössä)
      lähetä nippu ajoittajalle
      odota ajoittajan vastausta
      if (ajoittaja hylkäsi nipun)
            lähetä nippu itselle takaisin
else
      foreach (työ nipussa)
            lähetä työ ajoittajalle
            odota ajoittajan vastausta
            if (ajoittaja hylkäsi työn)
                  lähetä työ itselle takaisin

poista nippu Vectorista
```

Vastaanotto on tietoliikenneyhteydessä useisiin komponentteihin. Koska jokainen tietoliikenneyhteys tapahtuu omassa säikeessään, on vastaanotossa useita rinnakkaisia säikeitä. Kuva 6 esittää vastaanoton ajonaikaisia säikeitä. Kuvan 7 kaaviossa ovat säikeiden nimet sellaisina kuin ne ilmenevät kunkin säikeen metodia *java.lang.Thread.getName* kutsuttaessa.




*Kuva 6: Eräitä vastaanoton ajonaikaisten säikeiden välittämiä viestejä. Kuvassa näkyvät vastaanotto ja sen säikeisiin liittyvät muut tahot. Nuolet kuvaavat eri tahojen välistä viestintää. Nuolten selitteet kuvaavat viestien luonteita. Selitteet*

*ilmaisevat myös sitä, missä säikeessä viestit tapahtuvat. Säie on merkitty kirjaintunnuksella sulkuihin. Kirjaintunnuksen yhteydessä oleva tähti tarkoittaa, että saman tyyppistä säiettä voi olla useita kappaleita. Kuvasta on jätetty pois viestejä, jotka tapahtuvat jo kuvaan merkityissä säikeissä. Kuvasta puuttuu vastaanoton käynnistävä säie, sillä sen toiminta lakkaa välittömästi käynnistyksen jälkeen.*

| Tunnus | säikeen nimi |
|---|---|
| A | *Communications-portlistener #XX* |
| B | *Communications-portlistener #1* |
| C | *reAnThread* |
| D | *Thread-XX* |
| E | *reScThread* |

*Kuva 7: Vastaanoton ajonaikaisten säikeiden nimet. Tunnukset viittaavat kuvaan 6. Nimissä XX tarkoittaa kokonaislukuarvoa, joka määräytyy ajon aikana kullekin säikeelle erikseen.*

## 4.3 Analysoija

### 4.3.1 Analysoijan tehtävät ja toiminta

Analysoija on olio, jonka ajoittaja luo käynnistyessään. Analysoija tarjoaa tulostustöiden väliseen vertailuun ja yhdistämiseen liittyviä palveluja. Niitä on pääasiallisesti kolme: annettu tulostustyö voidaan **kuvata N-ulotteiseen tulostustyöavaruuteen**, sen **kustannus voidaan arvioida** yksin suoritettuna tai toisen työn kanssa yhdistettynä ja **kaksi tulostustyötä voidaan yhdistää** yhdeksi uudeksi työksi. Tässä tulostustyö voi tarkoittaa yhtä alkeistulostustyötä tai useista sellaisista muodostuvaa töiden nippua. Lisäksi analysoija tarjoaa tulostustyöolioiden puolesta niiden *toString*-palvelua, joka kuuluu yleiseen Java-ohjelmointityyliin. Tämä neljäs palvelu on mielekäs vain sellaisille tahoille, jotka käsittelevät tulostustöitä tavujonoina eikä olioina. Analysoija on Tertun arkkitehtuurin mukainen komponentti, ja se toteuttaa Tertun sille asettaman rajapinnan.

Ensisijaisesti analysoijan palveluja tarvitsee ajoittaja. Ajoittaja toimii niin, että kun se havaitsee tulostustyöavaruudessa tiheän ryppään töitä, se siirtää ryppään tiheimmästä kohdasta asetuksissa määrätyllä säteellä kaikki työt suoritukseen. Ennen suoritukseen siirtämistä säteen sisällä olevat työt niputetaan yhdeksi tai useammaksi nipuksi. Niputtamisen etenemisen määrää se, missä suhteessa erillisten töiden kustannukset ovat pareittain

yhdisteltyjen töiden kustannuksiin. Töitä niputetaan, jos se vähentää töiden yhteensä aiheuttamaa kustannusta.

Myös vastaanotto käyttää analysoijaa pyrkiessään kokeilemaan ajoittajan yhdistämispäätöksiä. Vastaanotto viivyttää töitä valikoivasti tavoitteenaan lähettää työt ajoittajalle sellaisissa rykelmissä, että rykelmässä olevat työt tulevat todennäköisesti yhdistymään keskenään ajoittajassa.

Kun ajoittaja päättää töiden yhdistämisestä, se käyttää tulostustöiden kustannuksia sellaisella tavalla, että analysoijan tulee kustannuksia laskiessaan antaa pieniä kustannuksia sellaisten töiden yhdisteille, jotka halutaan tulostuvan samalla tulostimella.

### 4.3.2 Analysoijan liittymät

Analysoijan palveluja voi käyttää kahden liittymän kautta: **Julkiset metodit** ovat Tertun arkkitehtuurin mukaisia. Lisäksi analysoijaan voi ottaa **tietoliikenneyhteyden**, jonka kautta palveluja voi käyttää. Pääasiallisesti julkisia metodeja kutsuu ajoittaja ja tietoliikenneyhteyttä käyttää vastaanotto. Analysoijalla on myös **tietoliikennerajapinta tulostushallinnan kanssa**. Tämän rajapinnan kautta analysoija saa järjestelmän asetustiedot, muun muassa tulostinten nimet ja koordinaatit.

**Julkisten metodien liittymä** koostuu seuraavista viidestä metodista. Kukin niistä saa parametrikseen yksi tai kaksi *Object*-ilmentymää. Näiden ilmentymien oletetaan olevan *byte[]*-ilmentymiä, jotka sisältävät jonkin Bundle-ilmentymän merkkijonoksi koodattuna. Toisin sanoen, kun parametrina on *Object data,* niin Java-ilmaisu *nippu.Bundle.deserialize(new String((byte[])data))* palauttaa sen Bundle-ilmentymän, joka *data*-parametrissa välittyy. Vastaavasti *fusion*-metodi palauttaa *Object*-viitteen *byte[]*-ilmentymään, joka sisältää palautettavan nipun merkkijonokoodauksen.

public float[] getCoordinates(Object data)

> Metodi palauttaa annetun nipun koordinaatit tulostustyöavaruudessa. Nämä koordinaatit on tarkoitettu ajoittajan sisäiseen käyttöön, eivätkä ne välttämättä ole missään suhteessa Bundle-luokan kenttiin x, y ja z (katso tietokuvauksen luku 3.1.2). Palautusarvona olevan taulukon koko on ajonaikainen vakio, joka määräytyy asetustiedoston perusteella (katso liittymäkuvauksen luku 3.3.2).

public float cost(Object data)

Metodi palauttaa annetun nipun kustannuksen yksinään suoritettuna. Palautusarvo ei mittaa mitään reaalimaailman suuretta. Arvolla on merkitys vain ajoittajalle, kun sitä verrataan metodista *combinedCost* saatuihin palautusarvoihin.

public float combinedCost(Object data1, Object data2)

Metodi palauttaa annettujen nippujen yhdisteen kustannuksen. Palautusarvo ei kuitenkaan ole välttämättä missään suhteessa lukuun *cost(fusion(data1, data2))*. Palautusarvo ei mittaa mitään reaalimaailman suuretta. Arvolla on merkitys vain ajoittajalle, kun arvoa verrataan metodista *cost* saatuihin palautusarvoihin.

public Object fusion(Object data1, Object data2)

Metodi palauttaa annetuista nipuista yhdistämällä saatavan nipun. Tämä sisältää Bundle-luokan *union*-metodin toiminnallisuuden (katso tietokuvauksen luku 3.1.2) sekä yhdiste-Bundlen *optimalPrinters*-kentän päivittämisen asetustiedostossa määritettyjen vakioiden mukaisesti (katso liittymäkuvauksen luku 3.3.2).

public String toString(Object data)

Metodi palauttaa ihmisten luettavaksi tarkoitetun merkkijonon, joka kuvaa annettua nippua. Merkkijono on sama, jonka annetun nipun *toString*-metodi palauttaa. Tätä analysoijan metodia tarvitsee vain ajoittaja, joka ei itse osaa muuntaa *data*-parametria Bundle-olioksi.

Analysoijaan liittyvä tietoliikennekomponentti avaa TCP-portin 7703, johon vastaanotto voi ottaa yhteyden. Nipun tietoliikenneprotokollan mukaiset paketit on kuvattu liittymäkuvauksen luvussa 3.3.3. Analysoija käyttää näistä paketteja 92 ja 93 (vastaanoton pyyntöviestit), E2 ja E3 (vastaukset vastaanotolle) sekä EF. Myös paketit 91, 94, 95, E1, E4 ja E5 ovat käytettävissä, mutta käytännössä niitä ei tarvita vastaanoton yksinkertaisen luonteen takia. Paketteihin sisältyvät niput on sarjallistettu metodilla *Bundle. serialize*, ja ne voidaan koota olioiksi metodilla *Bundle.deserialize*. Paketteihin sisältyvät nippujen tunnukset ovat metodin *Bundle.getBundleID* antamia kokonaislukuja.

Analysoijalla on Tertun ajoittajaa varten erityinen **alustamisesta huolehtiva kääreluokka** *nippu.analyzer.AnalyzerWrapper*[12]. Ajoittaja luo analysoijan kutsumalla sen parametritonta konstruktoria. Ajoittaja olettaa, että tämän jälkeen analysoija on täysin alustettu. *Analyzer-*

12 Tämä kääreluokka otettiin toteutusvaiheessa käyttöön siitä syystä, että analysoijan alustuksessa oli ongelmia, kun alustus suoritettiin yhdellä konstruktorin kutsulla. Toteutuksen loppuvaiheessa analysoijan alustus saatiin toimimaan ilman kääreluokkaakin, mutta kääreluokka jäi silti käyttöön. Samanlainen kääreluokka on myös tulostushallinnalla.

*Wrapper*-luokan konstruktori luo *nippu.analyzer.Analyzer*-olion ja kutsuu sen metodeja *initializeCommunicationsClient* ja *initializeCommunicationsServer*, jotka muodostavat analysoijan yhteyden tulostushallintaan ja avaavat analysoijan palvelimen vastaanottoa varten. *AnalyzerWrapper* toteuttaa lisäksi *Analyzerin* kanssa saman rajapinnan mutta siirtää sen metodikutsut muuttamattomina eteenpäin oikealle analysoijalle. Analysoijan käärеluokassa ei ole tässä ohjelmistoversiossa lisätoiminnallisuutta, mutta siitä voi olla hyötyä ohjelmiston myöhemmässä kehityksessä.

Analysoija lukee oman komponenttikohtaisen asetustiedostonsa, joka sisältää tulostushallinnan tietoliikenneparametrit. Tiedoston syntaksin tulee olla tietokuvauksen mukainen.

### 4.3.3 Analysoijan tietorakenteet

Analysoija ei käytä erityisiä tietorakenteita tietokuvauksessa mainittujen luokkien lisäksi, vaan sen toiminta perustuu funktioihin ja ajonaikaisiin vakioihin. Vakioiden arvot luetaan asetuksista, kun analysoija luodaan. Vakioiden merkitykset ja mahdolliset arvot on kuvattu yhteisen asetustiedoston määrittelyn yhteydessä liittymäkuvauksen luvussa 3.3.2.

### 4.3.4 Analysoijan toimintalogiikka

Palvelupyyntöavaruuden ja kustannusten välistä suhdetta tarkastellaan yksityiskohtaisesti liitteen 3 osassa A.

**Tulostustöiden kuvaus tulostustyöavaruuteen** tehdään niin, että samankaltaisille töille annetaan lähekkäiset koordinaatit. Tämän on tarkoitus auttaa ajoittajaa löytämään kaikkien tulostustöiden joukosta samankaltaisten töiden keskittymät. Töiden koordinaatit ovat sellaisia, että erityisesti sellaiset työt, joiden ei haluta vetävän toisiaan mukanaan tullessaan siirretyksi ajoittajasta tulostushallintaan, ovat avaruudessa kaukana toisistaan.

**Tulostustyöavaruuden akselien** lukumäärä on joko kolme tai kuusi. Se kiinnitetään vasta, kun Nippu-ohjelmisto on käynnissä ja ohjelmistoon liittyvät asetukset on luettu. Akseleiden mielletään kuvaavan tulostustyön fyysistä sijaintia. Koska tulostustyö ei varsinaisesti ole fyysinen esine, tämä sijainti on asetusmuuttujan *COORDINATESTYLE* mukaan jokin seuraavista kolmesta vaihtoehdosta:

a) tulostajan fyysinen sijainti (kolme akselia; submitterX, submitterY, submitterZ) (*COORDINATESTYLE*=submitter)
b) tulostajan kannalta suotuisimman tulostimen sijainti (kolme akselia; printerX, printerY, printerZ) (*COORDINATESTYLE*=printer)
c) kumpikin edellisistä (kuusi akselia; submitterX, submitterY, submitterZ, printerX, printerY, printerZ) (*COORDINATESTYLE*=submitter and printer).

Vaihtoehdoissa (b) ja (c) suotuisin tulostin tarkoittaa oletusarvoisesti tulostajaa lähintä tulostinta. Oletusarvon korvaa tulostustyölle mahdollisesti pakotettu tulostin.

Tulostajan **käyttäjätunnusta ei oteta huomioon** tulostustyöavaruudessa, koska luonnollinen pyrkimys olisi saada eri tulostajien identtiset työt yhtä kauas toisistaan tulostajasta riippumatta. Jos tulostajia on T kappaletta, tarvitaan avaruuteen tällöin T-1 akselia. Jos akseleita on vähemmän, niin mitä ilmeisemmin on löydettävissä sellaiset kaksi paria tulostajia (A, B) ja (C, D), että jos A ja B tulostavat keskenään identtiset työt ja C ja D tulostavat keskenään identtiset työt, niin A:n ja B:n töiden yhdistämistä suositaan enemmän kuin C:n ja D:n töiden yhdistämistä. Koska tulostajien lukumäärää T ei ole rajoitettu Nipussa mitenkään, voi T-1:n akselin lisääminen tulostustyöavaruuteen kuluttaa huomattavasti ohjelmiston ajonaikaisia resursseja.

**Yksittäisen työn[13] kustannus** on aina vakio 1,0 (katso liitteen 3 osa B). Tämä vakio kiinnittää kustannusten yleisen suuruusluokan. Kahden työn yhdisteen kustannusta laskettaessa otetaan lähtökohdaksi töiden erikseen laskettujen kustannusten summa. Tätä lähtöarvoa korjataan kertomalla se seuraavien kustannuksen osatekijöiden antamilla kertoimilla.

1. Töiden tulostajien käyttäjätunnukset.
2. Töiden koordinaatit tulostustyöavaruudessa (tähän sisältyy mahdollisen tulostimen pakottamisen huomioiminen).
3. Töiden sivumäärät.

Kertoimien tarkka käyttö esitetään alla olevassa pseudokoodissa. Kustannuksen osatekijöiden merkittävyyteen voidaan vaikuttaa asetuksissa olevilla analysoijan käyttämillä vakioilla (katso liittymäkuvauksen luku 3.3.2). Tietyt osatekijät voidaan myös jättää ottamatta huomioon asettamalla vakioiden arvot sopivasti (käyttämällä kerrointa "1.0"). Oletuksena

13 Yksittäinen työ voi olla tässä tapauksessa alkeistulostustyö tai töiden nippu.

osatekijöiden tärkeysjärjestys on yllä olevan listan mukainen (1. on tärkein).

Analysoijan käyttämät vakiot *DISTANCELIMIT* ja *MINDISTANCEMULTIPLIER* määrittävät matemaattisen funktion *getDistanceMultiplier(distance)*. Tämä funktio on kuvaus etäisyydeltä, joka tulostustyöavaruudessa sijaitsevien nippujen välissä on, kertoimelle, joka liittyy yllä olevaan nippujen yhdisteen kustannuksen kohtaan 2. Funktio määritellään seuraavasti, kun merkitään (lyhyyden vuoksi) A = *DISTANCELIMIT* ja B = *MINDISTANCEMULTIPLIER*.

$$getDistanceMultiplier\left(distance\right)=\frac{1-B}{A}\cdot distance+B$$

Funktion kuvaaja on *(distance, multiplier)*-koordinaatistossa suora, jonka kiinnittävät pisteet (*DISTANCELIMIT*; 1,0) ja (0,0; *MINDISTANCEMULTIPLIER*). Funktio toteuttaa käyttämiensä vakioiden kuvauksissa asetetut ehdot (luku 3.3.2).

Kahden työn yhdistettä luodessaan analysoija tekee **suosituksia siitä, millä tulostimella työ tulisi tulostaa**. Tavoitteena on tulostaa suuret työt nopeilla tulostimilla. Mitä suurempi nippu on, sitä kauemmas nipun voi laittaa tulostumaan ilman, että siitä koituu tulostajalle haittaa. Suuret niput nimittäin tulostuvat hitaammin, jolloin tulostajalla on enemmän aikaa saapua tulostimelle hakemaan työnsä.

Matemaattinen funktio *maxPrinterDistance* määrää, kuinka kauas tietyn kokoisen nipun voi laittaa tulostumaan. Kun nipussa on *p* sivua, niin tulostin, jonka nopeus on *s*, laitetaan nipun suositeltujen tulostinten listaan vain ja ainoastaan, jos tulostimen etäisyys nipun sisältämien tulostustöiden tulostajien fyysisten sijaintien keskiarvosta on korkeintaan *maxPrinterDistance(p, s)*, jossa

$$maxPrinterDistance\left(p,s\right)=\frac{K}{50}\cdot\frac{p}{50}\cdot s$$

ja *K* on asetustiedostosta luettava vakio *FASTPRINTERDISTANCE*. Funktio *maxPrinterDistance* toteuttaa vakiolle *FASTPRINTERDISTANCE* kappaleessa 3.3.2 asetetut ehdot. Funktio on lineaarisessa suhteessa argumentteihinsa sivumäärään ja tulostimen nopeuteen. Jos siis tarkastellaan funktiota pitäen sivumäärää vakiona ja muuttaen tulostimen nopeutta, niin sallittu etäisyys tulostimelle tullakseen suositelluksi kasvaa kaksinkertaiseksi, kun tulostimen nopeus kaksinkertaistuu. Vastaavasti pidettäessä tulostimen nopeutta vakiona ja

muutettaessa sivumäärää huomataan, että nippu tulee suositelluksi kaksi kertaa suuremmalla säteellä oleville tulostimille, kun nipun sivumäärä kaksinkertaistuu.

**Suositellut tulostimet talletetaan** nipun kenttään *optimalPrinters*. Tulostushallinta ottaa nämä suositukset huomioon tulostinta valitessaan. Kaikista nipulle suositeltavista tulostimista olisi mieluiten valittava nopein. Jos se ei ole jostain syystä käytettävissä, tulisi valita seuraavaksi nopein suositeltu tulostin ja niin edelleen. Tulostushallintaa ei kuitenkaan velvoiteta valitsemaan vain suositeltuja tulostimia.

Alla on pseudokoodiesitykset Tertun rajapinnan määräämistä analysoijan metodeista:

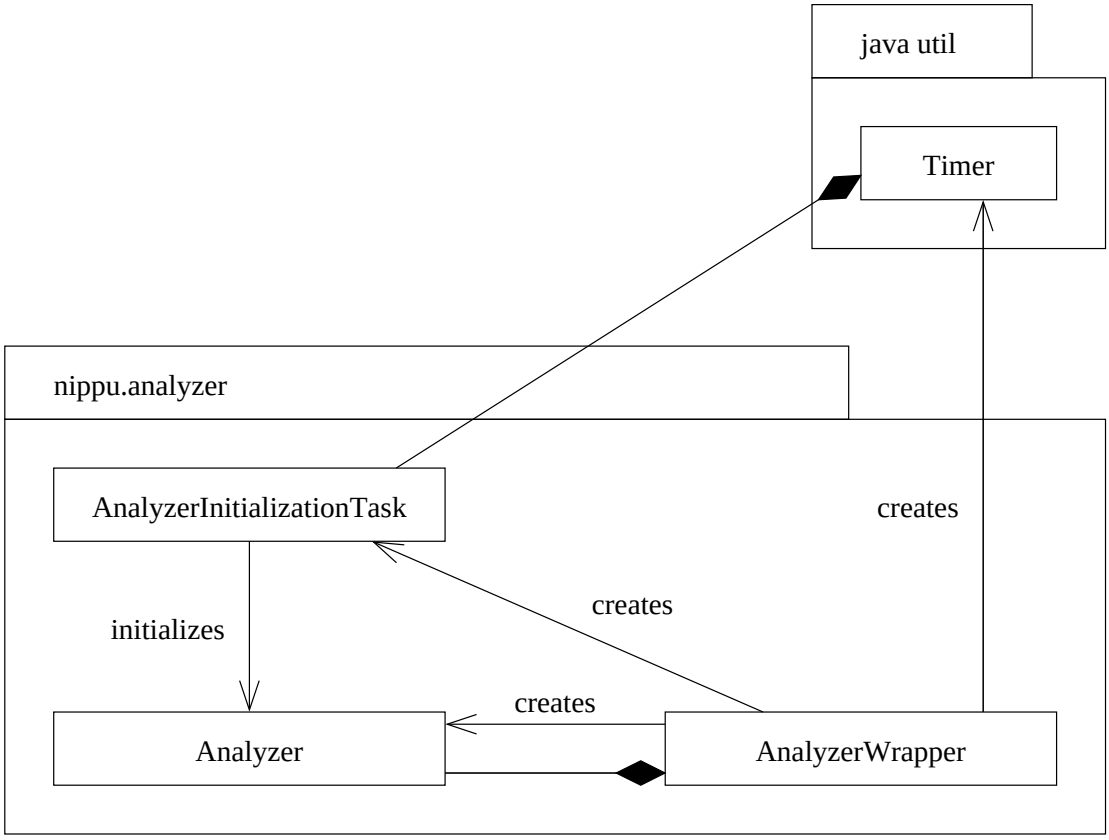
```
Analyzer.getCoordinates(Bundle nippu)
      if (koordinaatisto = tulostaja)
            return nipun tulostajan koordinaatit
      if (koordinaatisto = lähin tulostin)
            if (nipulla on pakotettu tulostin)
                  return pakotetun tulostimen koodinaatit
            else
                  return tulostajaa lähimmän tulostimen koordinaatit
      if (koordinaatisto = tulostaja ja lähin tulostin)
            return kohdista ”tulostaja” ja ”lähin tulostin” saatujen
              koordinaattien katenaatio

Analyzer.cost(Bundle nippu)
      return 1.0

Analyzer.combinedCost(Bundle nippu, Bundle nappu)

      // Lähtöarvona on erillisten nippujen kustannuksen summa.
      combinedCost := cost(data1) + cost(data2)

      // Tulostajien samuus vaikuttaa kustannukseen.
      if (kaikki nipun tulostajat ovat napun tulostajia tai toisinpäin)
            combinedCost *= SAMEUSERMULTIPLIER

      // Nippujen etäisyys vaikuttaa kustannukseen.
      distance := etäisyys pisteiden getCoordinates(nippu)
        ja getCoordinates(nappu) välillä
      combinedCost *= getDistanceMultiplier(distance)

      // Yhdistenipun koko vaikuttaa kustannukseen.
      if (nipun sivumäärä + napun sivumäärä > MAXNUMBEROFPAGES)
            combinedCost *= BIGBUNDLEMULTIPLIER

      // Samalle tulostimelle pakottaminen vaikuttaa kustannukseen.
      if (nipulla ja napulla on pakotettu tulostin
          ja se on molemmissa sama)
            combinedCost *= SAMEFORCEDPRINTERMULTIPLIER

      return combinedCost

Analyzer.fusion(Bundle nippu, Bundle nappu)
      uusnippu := nippu.union(nappu)
```

```
      // Päätellään yhdistenipulle suositeltavat tulostimet.
      optimalPrinters := empty
      foreach (tulostin in kaikki tunnetut tulostimet)
             if (uusnipun etäisyys tulostimeen
                 <= maxPrinterDistance(uusnipun sivumäärä, tulostimen
                 nopeus))
                    optimalPrinters.add(p)
      uusnippu.setOptimalPrinters(optimalPrinters)

      return newBundle

Analyzer.toString(Bundle nippu)
      return nippu.toString()
```

### 4.3.5 Analysoijan sisäinen rakenne

Analysoijan kääreluokan tekemässä tietoliikenteen alustuksessa tarvitaan **ajastinta**. Koska analysoijan tietoliikenteen alustus sisältää yhteyden muodostamisen tulostushallintaan, analysoijan alustuksen onnistumiseksi on välttämätöntä, että tulostushallinta on olemassa ja sen tietoliikenne on alustettu. Koska Tertun ajoittaja luo aina ensin analysoijan ja vasta sitten tietoliikenteen, luo *AnalyzerWrapper*-kääreluokka ajastimen, joka on luokan *java.util.Timer* ilmentymä. Ajastin kehottaa säännöllisin väliajoin *Analyzer*-ilmentymää alustamaan tietoliikenteensä. Ajastin lopettaa kehotusten antamisen vasta, kun tietoliikenne saadaan alustettua. Kehotusten antaminen on toteutettu luokassa *nippu.analyzer.AnalyzerInitializationTask*. Luokkien suhteet on merkitty kuvan 8 luokkakaavioon.

Analysoijalla on jokaista Tertun analysoijarajapinnan määrittelemää viittä palvelua kohti kolme metodia. Yksi niistä on tarkoitettu Tertun ajoittajan kutsuttavaksi. Se hyväksyy parametrikseen *Object*-viitteitä *byte[]*-olioihin, joihin on koodattu tietoliikenneprotokollan määräämällä tavalla nipun sarjallistetun esityksen merkkijono. Toinen metodi hyväksyy Bundle-olion sekä *AnalyzerReceptionProtocol*-olion. Tämä metodi on tarkoitettu analysoijan vastaanottoa kohti suunnatun tietoliikenteen kutsuttavaksi. Protokollaolio on yhteys takaisin metodikutsun aiheuttaneeseen vastaanottoon. Kolmas metodi toteuttaa palvelun varsinaisen toiminnallisuuden. Se hyväksyy Bundle-olion parametrinaan. Molemmat edelliset metodit kutsuvat tätä kolmatta metodia ja palauttavat sen palauttaman arvon.




*Kuva 8: analysoijan sisäisen rakenteen luokkakaavio.*

## 4.4 Tulostushallinta

### 4.4.1 Tulostushallinnan tehtävät ja toiminta

Tulostushallinnan keskeiset tehtävät ovat **hoitaa niputettujen töiden lopullinen tulostus** ja **välittää järjestelmän asetustiedot** analysoijalle ja käyttöliittymille. Tulostushallinta ottaa vastaan ajoittajan lähettämät tulostustyöt ja purkaa ne alkeistöiksi huolehtien, että nipun muodostaneet työt tulostuvat peräkkäin samalla tulostimella. Tulostushallinta lopullisesti päättää, mitä tulostinta käytetään, ja emuloi tulostusta valitulle tulostimelle.

Tulostushallinta **lukee yhteisestä asetustiedostosta** tulostinten nimet, tulostusnopeudet ja koordinaatit. Koko asetustiedosto talletetaan muistiin. **Asetustiedot välitetään** kaikille kysyjille (käyttöliittymät, analysoijat) tietoliikenneyhteyden välityksellä ilman pääsynhallintaa.

Kaikki **käyttöliittymän ilmentymät rekisteröityvät** tulostushallintaan tietoliikenneyhteyden välityksellä, ja tulostushallinta tuottaa niille yksilöivät tunnisteet. Kun työ on saapunut tulostushallintaan ja sopiva tulostin on valittu, tulostushallinta ilmoittaa työn tilasta käyttöliittymille. Ilmoitusviesti sisältää työn lähettäneen käyttöliittymän tunnisteen, jonka käyttöliittymät voivat tulkita haluamallaan tavalla. Tulostushallinta ilmoittaa käyttöliittymille vielä siitä, kun työ on tulostettu.

Tulostushallinta vastaa **Tertun arkkitehtuurin mukaista suorittajaa**, ja se toteuttaa Tertun *Executor*-luokan.

### 4.4.2 Tulostushallinnan liittymät

Ajoittaja kutsuu tulostushallintaa Tertussa määritellyn *Executor*-luokkarajapinnan kautta. Tämä toteutetaan *PrintingManagement*-luokalla, joka määritellään seuraavasti:

**PrintingManagement-luokka**

- **Määrittely**
  public class PrintingManagement implements Executor
- **Konstruktori**
  public PrintingManagement ()
- **Julkiset metodit**
  public boolean execute (Object req_data)
  Metodi suorittaa ajoittajalta saamansa palvelupyynnön. Metodi tulkitsee saamastaan merkkijonosta Bundle-olion ja sijoittaa sen oikeaan tulostusjonoon. Metodin paluuarvo on null.
  public LinkedList getPendingJobs()
  Metodi palauttaa työt, joita ei oltu saatu laitettua mihinkään tulostusjonoon ennen ajoittajan pysäytystä.
  public boolean getSchedulerStopped()
  Metodi palauttaa true, jos ajoittaja on pysäytetty, muuten false.
  public Vector getUIConnections()
  Metodi palauttaa yhteydet asiakkaisiin (käyttöliittymät ja analysoija) Vector-tietorakenteessa.

public int register(PrintingManagementUserInterfaceProtocol connection)
Tämän metodin avulla asiakkaat voivat rekisteröityä kuuntelijoiksi. Metodi palauttaa asiakkaan tunnisteen.
public void unregister(int listenerID)
Metodin avulla asiakkaat voivat perua rekisteröitymisensä.
public void changeSchedulerState(int newState)
Metodin avulla voidaan pyytää tulostushallintaa pysäyttämään (newState = 3) tai käynnistymään (newState = 4) ajoittaja.

Tulostushallinnalla on kääreluokka *PrintingManagementWrapper*, joka kutsuu *Printing-Management*-luokkaa. Kääreluokassa ei ole tässä ohjelmistoversiossa lisätoiminnallisuutta, mutta siitä voi olla hyötyä ohjelmiston myöhemmässä kehityksessä.

Tulostushallinta avaa kaikkiin IP-verkkorajapintoihin TCP-portin numero 7704, johon analysoija ja käyttöliittymät voivat ottaa yhteyden ilman pääsynvalvontaa. Pääkäyttäjän komennot hyväksytään kuitenkin vain localhost-verkkorajapintaan avatusta portista.

Käyttöliittymien tulee **rekisteröityä tulostushallintaan**, jotta ne saavat tiedon töistään ja jotta pääkäyttäjien käyttöliittymät saavat luvan tai kiellon lähettää ajoittajan käynnistys- ja pysäytyspyyntöjä ja palvelunsaannin maksimiviiveen muutospyyntöjä. Rekisteröinti palauttaa käyttöliittymälle yksilöivän, kokonaislukumuotoisen tunnisteen. Käyttöliittymä myös rekisteröityy pois tulostushallinnasta. Tässä viestinnässä käytetään liittymäkuvauksen luvussa 3.3.3 kuvattuja viestityyppejä 81 ja 82 (vastaanotto) sekä A1 ja A3 (lähetys). Samoin analysoija joutuu rekisteröitymään tulostushallintaan saadakseen tiedon järjestelmän tulostimista.

Tulostushallinta toimittaa rekisteröityneelle komponentille automaattisesti yhteisen asetustiedoston (viestityyppi D1) sekä tulostusjonojen tilan sillä hetkellä (sarja B5-viestejä). Sen jälkeen tulostushallinta toimittaa ilman eri pyyntöä tiedon jonoissa tapahtuvista muutoksista kaikille asiakkaille (viestityypit B5, B6) aina, kun jokin työ lisätään jonoon tai jonkin työn tulostus valmistuu. Tulostushallinta toimittaa tiedon myös ajoittajan ajotilan muuttumisesta (viestityyppi C1). Asiakas voi lopettaa viestien kuuntelun poistamalla rekisteröintinsä tulostushallinnasta. Tämä tapahtuu lähettämällä tulostushallintaan viesti, jonka tunnus on 82. Tulostushallinta vastaa siihen viestillä A3 eikä lähetä enää enempää viestejä tälle asiakkaalle.

Pääkäyttäjän liittymästä voi lähettää **ajoittajan pysäytys- tai käynnistyspyynnön** tulostushallinnalle (viestityyppi 85), jos pääkäyttäjän yhteys on otettu localhost-verkkorajapinnan kautta. Tulostushallinta ilmoittaa kaikille asiakkaille yhteisesti viestillä C1, jos ajoittajan ajotila muuttuu. Tulostushallinta saattaa myös oma-aloitteisesti muuttaa ajoittajan ajotilaa. Tulostushallinta itse rekisteröityy ajoittajan sisäiseksi asiakkaaksi ajoittajan pysäytystä varten ja lähettää ajoittajan ajotilan muutospyynnöt tämän tietoliikenneyhteyden kautta.

Tulostushallinta lukee käynnistyessään järjestelmän yhteisen asetustiedoston, joka sisältää kaikki tulostimia ja käyttäjiä koskevat tiedot, joten asetustiedosto on yksi tulostushallinnan liittymistä. Asetustiedosto tulkitaan käynnistyksen aikana, ja käynnistys epäonnistuu, jos asetustiedosto ei ole kelvollinen. Tulostushallinta lukee myös oman komponenttikohtaisen asetustiedostonsa, joka sisältää ajoittajan verkkoparametrit. Kummankin tiedoston syntaksin on oltava liittymäkuvauksen mukainen.

Tässä ohjelmistoversiossa töitä ei lähetetä eteenpäin oikeille tulostimille, vaan tulostushallinta itsessään emuloi tulostusta. Ulkoista liittymää tulostimiin siis ei ole.

### 4.4.3 Tulostushallinnan tietorakenteet

Tulostushallinta käyttää **tulostinten hallintaan ja simulointiin** *NippuPrinterSimulator*-luokkaa, joka on *NippuPrinter*-luokan aliluokka. Luokka sisältää ajastimen, jota käytetään tulostuksen simulointiin, ja linkitetyn listan, jossa jonossa olevia töitä säilytetään. *NippuPrinterSimulator*-olioita säilytetään Vector tietorakenteessa. Linkitetty lista on toteutettu *LinkedList*-tietorakenteen avulla.

Jokaisesta ajoittajan lähettämästä Bundlen *serialize*-merkkijonoversiosta luodaan uudelleen Bundle-olio. Tämä Bundle-olio puretaan PrintJob-olioiksi ja tallennetaan sopivan tulostimen jonoon.

**Tietoliikenneyhteyksiä** käyttöliittymiin ja analysoijaan säilytetään *UIConnection*-olioissa, jotka sisältävät tietoliikenneprotokollan ja käyttöliittymän tunnuksen. Nämä *UIConnection*-oliot säilytetään Vector tietorakenteessa.

### 4.4.4 Tulostushallinnan toimintalogiikka

Ajoittaja lähettää työt tulostushallintaan Bundle-nipuissa, joista ne puretaan yksittäisiksi tulostustöiksi. Tulostushallinta tekee lopullisen päätöksen, **millä tulostimella** nipun työt tulostetaan. Päätös tehdään seuraavalla logiikalla:

1. Jos on pakotettu tulostin, käytetään sitä.
2. Etsitään lähin tulostin nippuun liittyvästä optimaalisten tulostinten listasta ja käytetään sitä, jos sen jono on tyhjä.
3. Etsitään lähin tulostin, jonka jono on tyhjä.
4. Jätetään nippu odottamaan ja pysäytetään ajoittaja, kunnes jokin jono tyhjenee. Siirretään nipun työt tyhjentyneeseen jonoon ja käynnistetään ajoittaja.

Kohdissa kaksi ja kolme verrataan nipun koordinaatteja ja tulostinten koordinaatteja.

Jokaisella tulostimella on tulostusjonon lisäksi ajastin, jossa on käsittelyssä aina kerrallaan yksi työ. Ajastin simuloi tulostustapahtumaa. Ajastimeen liittyy *PMTimerTask*-luokka, joka laajentaa *TimerTask*-luokkaa ja sisältää ajastimen toiminnallisuuden. Ajastimen odotusajan arvo saadaan jakamalla työn sivumäärä tulostimen tulostusnopeudella. Kun ajastimen aika on kulunut loppuun, ajastin ilmoittaa työn valmistumisesta käyttöliittymille ja ottaa seuraavan työn käsittelyyn.

Seuraava pseudokoodi kuvaa tulostushallinnan toimintaa:

**Tulostimen valinta uudelle nipulle**

```
if (jossain nipun työssä pakotettu tulostin)
      valitse tulostimeksi pakotettu tulostin
else if (jollain nipun optimalprinters-listan tulostimella tyhjä jono)
      valitse nipun koordinaatteja lähin tulostin
else if (joku tulostinjono tyhjä)
      valitse se tulostin, johon on lyhin matka nipun koordinaateista
        ja lisää työt jonoon
else
      merkitse nippu odotustilaan
      pysäytä ajoittaja
      lähetä kuuntelijoille viesti C1
if (tulostin valittu)
      for (nipussa olevat työt)
             lisaa työ valitulle tulostimelle
             lähetä kaikille kuuntelijoille viesti B5
```

**Työn lisääminen jonoon**

```
if (tulostimen jonossa on työ)
      lisää työn jonon perään
else
      lisää työ jonoon
      ajastin.aika := työn sivumäärä / tulostimen nopeus
```

**Ajastimen ajan käytyä loppuun**

```
lähetä kaikille kuuntelijoille viesti B6

poista työ ajastimesta

if (tulostusjonossa töitä)
      kopioi jonon päästä työ ajastimeen
      ajastin.aika := työn sivumäärä / tulostimen nopeus
      poista työ jonosta
else if (ajoittaja pysäytetty)
      käynnistä ajoittaja
      lähetä kuuntelijoille viesti C1
      käsittele odotustilassa oleva nippu
```

## 4.5 Tietoliikennekomponentti

### 4.5.1 Tietoliikennekomponentin tehtävät ja toiminta

Tietoliikenteen tehtävä on **välittää tietoa muiden komponenttien välillä**. Arkkitehtuurikuvauksen (luku 3.2) mukaisesti tietoliikenneyhteyksiä on lähes kaikkien ohjelmistokomponenttien välillä. Nipun tietoliikennekomponentti hallinnoi näitä yhteyksiä.

Nipun tietoliikennekomponentti jakautuu **yleiseen osaan** ja **erikoistettuihin osiin**. Yleinen osa sisältää sellaisen toiminnallisuuden, joka on yhteistä kaikille Nipun tietoliikenneyhteyksille. Erikoistettuja osia on useita. Kullakin tietoliikennettä käyttävällä Nipun komponentilla on yksi tietoliikennekomponentin erikoistettu osa. Tämä osa sisältää kyseiselle komponentille ominaisen tietoliikennetoiminnallisuuden. Yleinen ja erikoistettu osa toimivat yhdessä toisiaan hyödyntäen.

Tietoliikennekomponentin erikoistetut osat koostuvat kukin kahdenlaisista luokista: **protokollaluokista** ja **viestinvälitysluokista**. Protokollaluokat on nimetty tyyliin *XxxYyy-*

*Protocol* ja viestinvälitysluokat tyyliin *XxxYyyServices*, jossa *Xxx* on sen Nipun komponentin nimi, jonka yhteydessä luokat toimivat, ja *Yyy* on kyseessä olevan yhteyskumppanin nimi. Nimeämiskäytäntö mukailee Terttua.

**Protokollaluokkien tehtävä** on muuntaa metodikutsuilla välitettävät viestit tavuvirrassa kulkeviksi viesteiksi ja päinvastoin. Tätä varten protokollaluokissa on useita **lähetysmetodeja** sekä yksi **vastaanottometodi**. Lähetysmetodien nimet alkavat aina sanalla *send*, ja vastaanottometodin nimi on *handleMessage*. Kukin lähetysmetodi osaa lähettää yhdentyyppisen viestin. Vastaanottometodi osaa tulkita tavuvirrasta kaikki kyseisen tietoliikenneyhteyden viestit. Vastaanottometodi kutsuu oman viestinvälitysluokkansa sopivaa metodia tulkittuaan viestin.

**Viestinvälitysluokkien tehtävä** on välittää vastaanotetut, metodikutsuiksi muunnetut viestit eteenpäin sellaisille tahoille, jotka ovat niistä kiinnostuneita. Mitään muuta viestinvälitysluokan ei ole tarkoitus tehdä. Erityisesti sen ei pidä itse lähettää vastausviestejä. Kaikki viestin aiheuttama toiminnallisuus tulisi tapahtua vasta niissä luokissa, joille viestinvälitysluokka välittää viestin. Viestinvälitys on erotettu protokollasta omaksi luokakseen, jotta viesteihin reagoimisen toteuttaminen olisi selkeämpää. Viestinvälitysluokka on sisääntuleviin viesteihin reagoinnin ylempi taso, kun taas protokollaluokka on sen alempi taso.

Eräs tietoliikennekomponentin erityistehtävä on huolehtia Javan alkeistyyppien int ja float sekä *String*-olioiden oikeanlaisesta koodauksesta tietoliikenneviesteihin ja niistä pois. Tätä tarvitaan etupäässä sisäisesti tietoliikenteessä mutta myös erikseen analysoijassa ja tulostushallinnassa Tertun *Analyzer*- ja *Executor*-rajapintojen kautta siirtyvän tiedon tulkitsemisessa.

Jokaisesta Nipun komponenttia vastaavasta tietoliikennekomponentin erikoistetusta osasta tehdään yksi ilmentymä kutakin kyseisen komponentin kyseistä erikoistettua osaa vastaavaa tietoliikenneyhteyttä kohti. Toisin sanoen jos esimerkiksi vastaanotolla on yhteys kahteen käyttöliittymään, niin vastaanotolla on tällöin kaksi eri ilmentymää luokasta *ReceptionUserInterfaceProtocol* ja kaksi eri ilmentymää luokasta *ReceptionUserInterfaceServices*.

Nipun **tietoliikennekomponentti perustuu Terttuun**. Pakkauksen *terttu.server.network*

luokat *Protocol* ja *TerttuProtocol* ovat mukana sellaisenaan Nipun protokollaluokkien yläluokkina. Nipun pakkauksen *nippu.communications* tärkeimmät kaksi luokkaa on rakennettu Tertun pakkauksen *terttu.server.network* lähdekoodia vahvasti hyödyntäen. Nimittäin *Communications* pohjautuu *TerttuServerTCP*-luokkaan ja *CommunicationsThread TerttuServerThread*-luokkaan. Luokkarakenteet eivät muilta osin vastaa täysin toisiaan. Nipussa on pyritty pitämään tietoliikenneyhteyksien toteutukset rakenteeltaan mahdollisimman yhtenevinä, yleiskäyttöisinä ja yksinkertaisina kuitenkaan poikkeamatta merkittävästi Tertun arkkitehtuuriratkaisuista.

### 4.5.2 Tietoliikennekomponentin liittymät

Tässä kappaleessa esitellään **protokolla- ja viestinvälitysluokat** erikseen kullekin Nipun komponentille. Protokollaluokista esitetään ne metodit, joita kutsumalla kyseinen Nipun komponentti voi lähettää haluamiaan viestejä yhteyskumppanilleen. Viestinvälitysluokista esitetään ne metodit, joita tietoliikennekomponentti kutsuu aina, kun metodia vastaava viesti saapuu tietoliikenneyhteyttä pitkin Nipun komponentille. Metodien nimien on tarkoitus olla riittävän kuvaavia, jotta metodin tarkoitus voidaan ymmärtää. Viestinvälitysluokan toteuttajan vastuulle jää suorittaa kunkin viestin velvoittamat toiminnot ja lähettää mahdolliset vastausviestit.

**Tietoliikenteen liittymä käyttöliittymälle** koostuu luokista *UserInterfaceReceptionProtocol*, *UserInterfacePrintingManagementProtocol*, *UserInterfaceReceptionServices* ja *UserInterfacePrintingManagementServices*, jotka määritellään seuraavasti:

```
class UserInterfaceReceptionProtocol
      void sendNewBundle(Bundle bundle)
      void sendMaxServiceDelayChangeQuery(int newMaxServiceDelay)
      void sendListenerRegistration()
      void sendListenerUnregistration(int listenerID)

class UserInterfacePrintingManagementProtocol
      void sendListenerRegistration()
      void sendListenerRegistration(int listenerType)[14]
      void sendListenerUnregistration(int listenerID)
      void sendSchedulerStateChangeQuery(int newState)

class UserInterfaceReceptionServices
      void newBundleReceived(int returnValue)
```

14 Luokassa voi olla saman niminen metodi määritelty kahteen kertaan. Kyseessä ei ole virhe vaan Javan ns. method overloading -menetelmän käyttö: Parametriton versio tekee samaa kuin parametrillinen, se vain antaa parametrille oletusarvon 0. Näin metodia käyttävän ohjelmoijan ei tarvitse kirjoittaa joka paikkaan nollaa parametriksi, jos hänelle on yhdentekevää, mikä parametrin arvo on.

```
      void maxServiceDelayChanged(int newMaxServiceDelay)
      void registrationAccepted(int listenerID, int listenerType)
      void unregistrationAccepted(int returnValue)

class UserInterfacePrintingManagementServices
      void configuration(String configuration)
      void registrationAccepted(int listenerID, int listenerType)
      void unregistrationAccepted(int returnValue)
      void printJobQueued(PrintJob printJob, String printer)
      void printJobPrinted(PrintJob printJob)
      void schedulerStateChanged(int newState)

class ReceptionAnalyzerServices
      void error()
```

**Tietoliikenteen liittymä vastaanotolle** koostuu luokista *ReceptionUserInterfaceProtocol*, *ReceptionAnalyzerProtocol*, *ReceptionSchedulerProtocol*, *ReceptionUserInterface-Services*, *ReceptionAnalyzerServices* ja *ReceptionSchedulerServices*, jotka määritellään seuraavasti:

```
class ReceptionUserInterfaceProtocol
      void sendNewBundleReceived(int returnValue)
      void sendMaxServiceDelayChanged(int newMaxServiceDelay)
      void sendRegistrationAccepted(int listenerID)
      void sendUnregistrationAccepted(int returnValue)

class ReceptionAnalyzerProtocol
      void sendGetCoordinatesQuery(Bundle bundle)
      void sendCostQuery(Bundle bundle)
      void sendCombinedCostQuery(Bundle bundle1, Bundle bundle2)
      void sendFusionQuery(Bundle bundle1, Bundle bundle2)
      void sendToStringQuery(Bundle bundle)

class ReceptionSchedulerProtocol
      void sendNewBundle(Bundle bundle)

class ReceptionUserInterfaceServices
      void newBundle(Bundle bundle)
      void maxServiceDelayChangeQuery(int newMaxServiceDelay)
      void listenerRegistration(int listenerType)
      void listenerUnregistration(int listenerID)

class ReceptionAnalyzerServices
      void getCoordinatesAnswer(int bundleID, float[] coordinates)
      void costAnswer(int bundleID, float cost)
      void combinedCostAnswer(int bundle1ID, int bundle2ID, float cost)
      void fusionAnswer(int bundle1ID, int bundle2ID, Bundle fusion)
      void toStringAnswer(int bundleID, String bundleString)

class ReceptionSchedulerServices
      void newBundleReceived(int returnValue)
```

**Tietoliikenteen liittymä analysoijalle** koostuu luokista *AnalyzerReceptionProtocol*, *AnalyzerPrintingManagementProtocol*, *AnalyzerReceptionServices* ja *AnalyzerPrinting-ManagementServices*, jotka määritellään seuraavasti:

```
class AnalyzerReceptionProtocol
      void sendGetCoordinatesAnswer(int bundleID, float[] coordinates)
      void sendCostAnswer(int bundleID, float cost)
      void sendCombinedCostAnswer(int bundle1ID, int bundle2ID,
                                  float cost)
      void sendFusionAnswer(int bundle1ID, int bundle2ID, Bundle
                            fusion)
      void sendToStringAnswer(int bundleID, String bundleString)

class AnalyzerPrintingManagementProtocol
      void sendListenerRegistration()
      void sendListenerRegistration(int optionalData)
      void sendListenerUnregistration(int listenerID)

class AnalyzerReceptionServices
      void getCoordinates(Bundle bundle)
      void cost(Bundle bundle)
      void combinedCost(Bundle bundle1, Bundle bundle2)
      void fusion(Bundle bundle1, Bundle bundle2)
      void toString(Bundle bundle)

class AnalyzerPrintingManagementServices
      void configuration(String configuration)
      void registrationAccepted(int listenerID)
      void unregistrationAccepted(int returnValue)
      void printJobQueued(PrintJob printJob, String printer)
      void printJobPrinted(PrintJob printJob)
      void schedulerStateChanged(int newState)
```

**Tietoliikenteen liittymä tulostushallinnalle** koostuu luokista *PrintingManagementUser-InterfaceProtocol* ja *PrintingManagementUserInterfaceServices*. Näitä samoja luokkia käytetään tietoliikenneyhteyksiin sekä käyttöliittymän että analysoijan suuntaan. Luokat määritellään seuraavasti:

```
class PrintingManagementUserInterfaceProtocol
      void sendConfiguration(String configuration)
      void sendRegistrationAccepted(int listenerID, int listenerType)
      void sendUnregistrationAccepted()
      void sendPrintJobQueued(String printer, PrintJob printJob)
      void sendPrintJobPrinted(PrintJob printJob)
      void sendSchedulerStateChanged(int newState)

class PrintingManagementUserInterfaceServices
      void listenerRegistration(int listenerType)
      void listenerUnregistration(int listenerID)
      void schedulerStateChangeQuery(int newState)
```

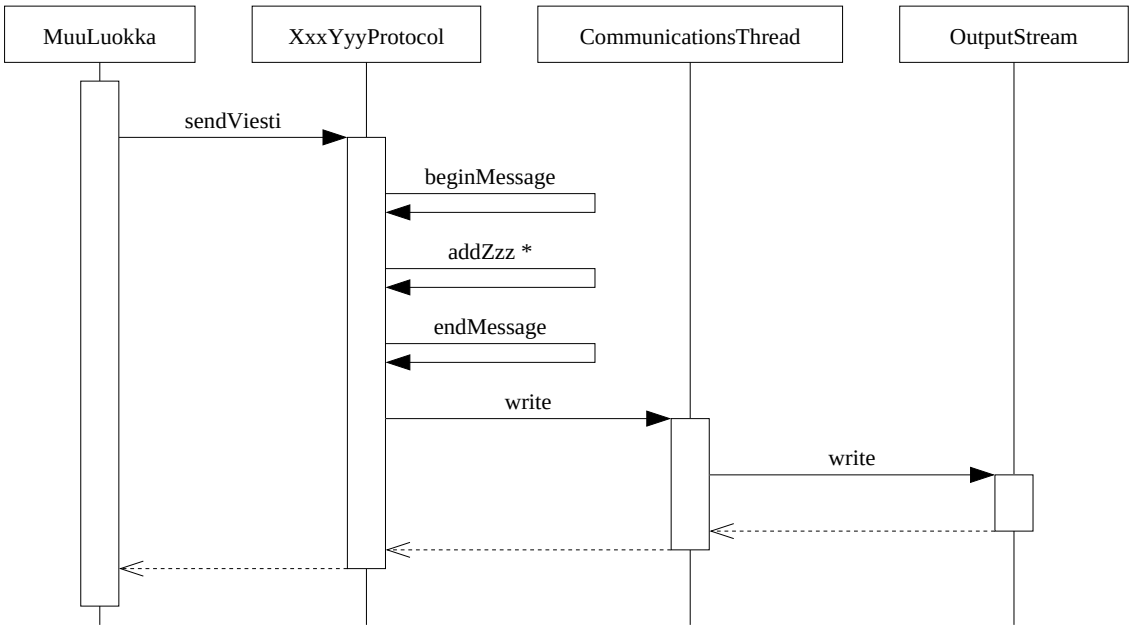
```
class PrintingManagementSchedulerProtocol
      void sendListenerRegistration(int listenerType)

class PrintingManagementSchedulerServices
      void registrationAccepted()
```

### 4.5.3 Tietoliikennekomponentin tietorakenteet

Tietoliikennekomponentti ei käytä mitään erityisiä tietorakenteita tietokuvauksessa mainittujen luokkien lisäksi.

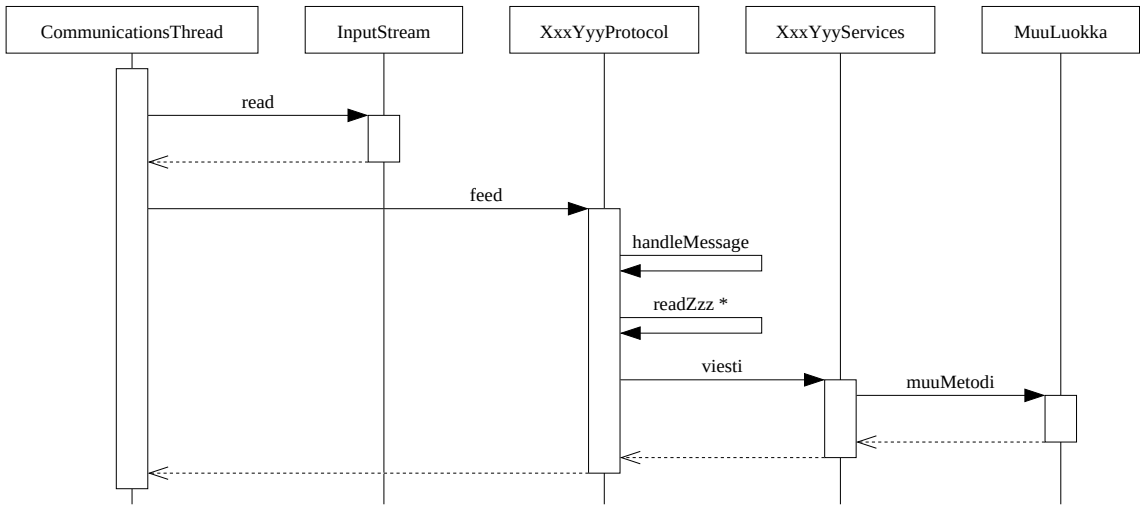
### 4.5.4 Tietoliikennekomponentin toimintalogiikka

**Tietoliikenneviestin lähettäminen** tapahtuu Nipussa kuvan 9 sekvenssikaavion mukaan. Kuvassa *MuuLuokka* voi olla mikä tahansa tietoliikennekomponentin ulkopuolinen luokka. Kun *MuuLuokka* haluaa lähettää viestin jollekin yhteyskumppanille, se kutsuu oikean protokollaluokan oikean ilmentymän oikeaa lähetysmetodia (kuvassa nimellä *sendViesti*). Protokollaluokka muuntaa lähetysmetodilla annetun viestin tavuvirraksi käyttäen yläluokan *NippuProtocol* metodeja *beginMessage*, *addInt*, *addFloat*, *addString*, *addBytes* ja *endMessage* (merkitty kuvaan lyhyesti *addZzz*). Tällä tavalla luotu tavuvirta ohjataan tietoliikennesäikeelle metodilla write. Tietoliikennesäie siirtää tavuvirran Javan valmiiseen kalustoon kuuluvan *Socket*-luokan ulospäin menevään tavuvirtaan *write*-metodilla.



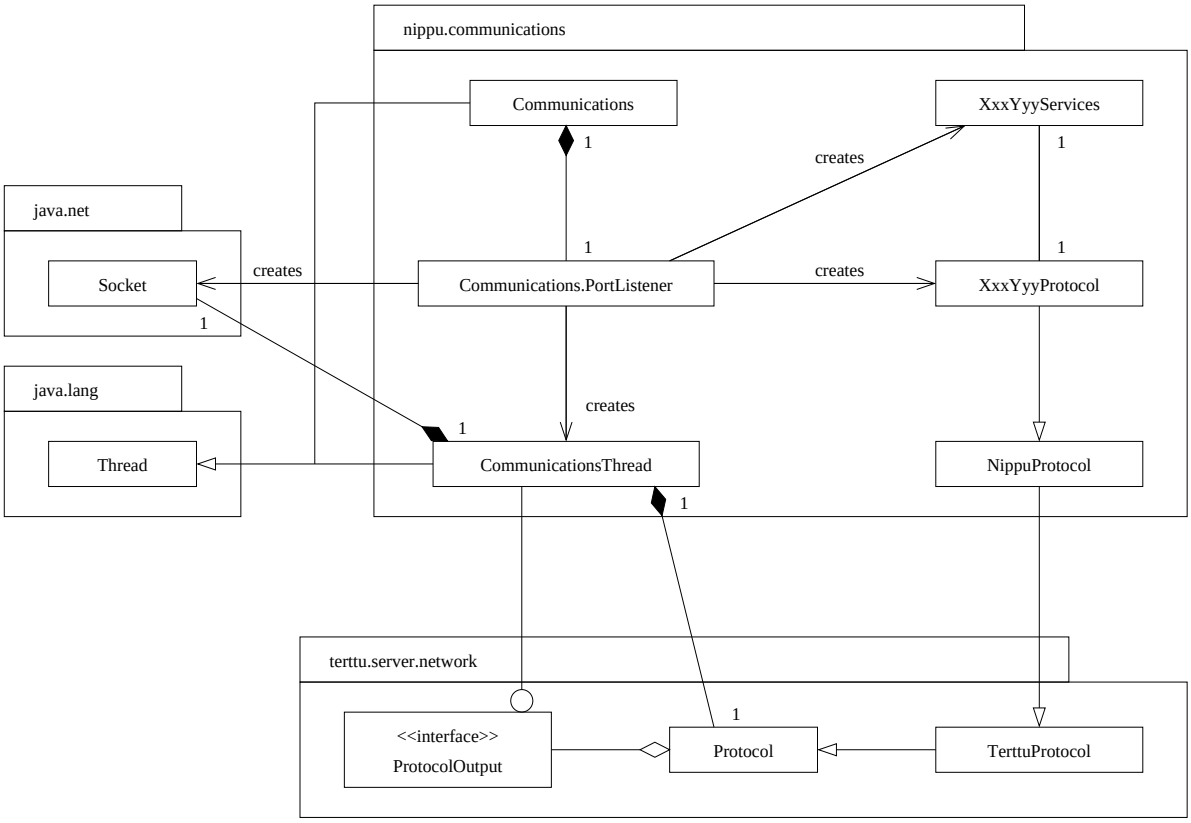

*Kuva 9: sekvenssikaavio tietoliikenneviestin lähettämisestä.*

**Tietoliikenneviestin vastaanottaminen** tapahtuu Nipussa kuvan 10 sekvenssikaavion mukaan. Kuvassa *MuuLuokka* voi olla mikä tahansa tietoliikennekomponentin ulkopuolinen luokka ja *muuMetodi* on joku sen metodi. Tietoliikenneviesti saapuu Javan valmiiseen kalustoon kuuluvaan *Socket*-luokkaan sen sisäänpäin virtaavaa tavuvirtaa pitkin. *CommunicationsThread* käy ajoittain lukemassa tavuvirran tuomia tavuja metodilla *read*. Se siirtää tavut eteenpäin protokollaluokalle *feed*-metodilla, joka sijaitsee protokollaluokan yläluokassa *TerttuProtocol*. Kun protokollaluokka on saanut riittävästi tavuja, se voi tulkita niistä viestin. Viestin tulkinta aloitetaan kutsumalla protokollaluokan metodia *handleMessage.* Se tulkitsee tavuista viestin *NippuProtocol*-luokan tarjoamilla metodeilla *readInt, readFloat, readString* ja *readBytes* (merkitty kuvaan lyhyesti *readZzz*). Viesti välitetään viestinvälitysluokalle sopivalla metodilla (tässä nimeltään *viesti*). Viestinvälitysluokka suorittaa viestin vaatimat toimenpiteet kutsumalla tietoliikennekomponentin ulkopuolisten luokkien metodeja.




*Kuva 10: sekvenssikaavio tietoliikenneviestin vastaanottamisesta.*

Kuvassa 11 on luokkakaavio Nipun tietoliikennekomponentin luokista. Luokka *CommunicationsThread* vastaa Tertun luokkaa *TerttuServerThread*.




*Kuva 11: luokkakaavio Nipun tietoliikennekomponentin luokista. Nimet XxxYyyProtocol ja XxxYyyServices tarkoittavat Nipun useita protokolla- ja viestinvälitysluokkia.*

**Tietoliikennekomponentin yleinen osa** koostuu muutamasta luokasta ja rajapinnasta, jotka on merkitty kuvan 11 luokkakaavioon. Kutakin tietoliikenneyhteyttä varten on olemassa yksi tietoliikennesäie eli luokan *CommunicationsThread* ilmentymä. Se periytyy Javan valmiista *Thread*-luokasta. Säie kuuntelee jatkuvasti saapuvaa tavuvirtaa Javan peruskalustoon kuuluvan *Socket*-luokan avulla. Tertusta peräisin oleva *ProtocolOutput*-rajapinta ilmentää mahdollisuutta kirjoittaa tavuja tavuvirtaan.

Tertun luokat *Protocol* ja *TerttuProtocol* sekä Nipun luokka *NippuProtocol* toteuttavat Nipun protokollaluokkien tarvitsemat perustoiminnot. *Protocol* ja *TerttuProtocol* hoitavat tietoliikennepakettien matalan tason käsittelyn. *NippuProtocol* tarjoaa helppokäyttöisen rajapinnan, jolla Javan int- ja float-tietotyypit sekä *String*-oliot voidaan muuntaa tavumuotoisiksi tietoliikennepaketeiksi ja takaisin. *NippuProtocol* automatisoi useita *TerttuProtocol*-luokkaan liittyviä asioita, joita luokkaa käyttävän ohjelmoijan täytyy muistaa. Näitä ovat viestin yleisrakenteesta huolehtiminen, viestin täydentäminen siten, että sen pituus on neljällä jaollinen, sekä viestin data-osan sisältämien tietotyyppien indekseistä huo-

lehtiminen.

### 4.5.5 Tietoliikennekomponentin käyttöönotto

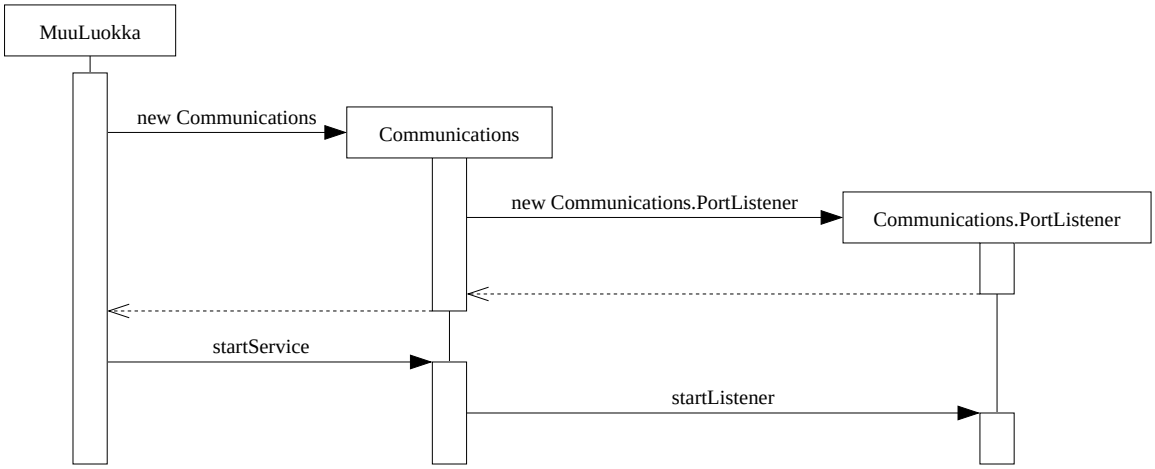
Tietoliikennekomponentilla on kaksi käyttötapaa: **palvelimen toteuttaminen ja asiakkaan toteuttaminen**. Palvelin alustetaan luomalla ilmentymä luokasta *Communications* ja käynnistetään kutsumalla metodia *Communications.startService* kuvan 12 sekvenssikaavion mukaan. Asiakas alustetaan luomalla ilmentymä luokasta *CommunicationsThread* ja käynnistetään kutsumalla metodia *CommunicationsThread.start* kuvan 13 sekvenssikaavion mukaan. Palvelimen päässä yhteyden muodostaminen hoidetaan automaattisesti kuvan 14 sekvenssikaavion mukaan.

Jotta tietoliikennekomponentin käytäntöön soveltaminen olisi helpompaa, pakkaukseen *nippu.communications.example* on tehty protokolla- ja viestinvälitysluokista **mallipohjat** *CatDogProtocol* ja *CatDogServices*. Niiden Java-lähdekoodi sisältää vaiheittaiset sovellusohjeet sekä yleisimmistä virheistä varoittavia kommentteja, joiden avulla malleista pitäisi olla helpohkoa tuottaa protokolla- ja viestinvälitysluokkia todelliseen käyttöön. Mallipohjat eivät ole käännettävissä eivätkä ne liity Nippu-järjestelmän tai minkään muunkaan ohjelmiston toiminnallisuuteen.

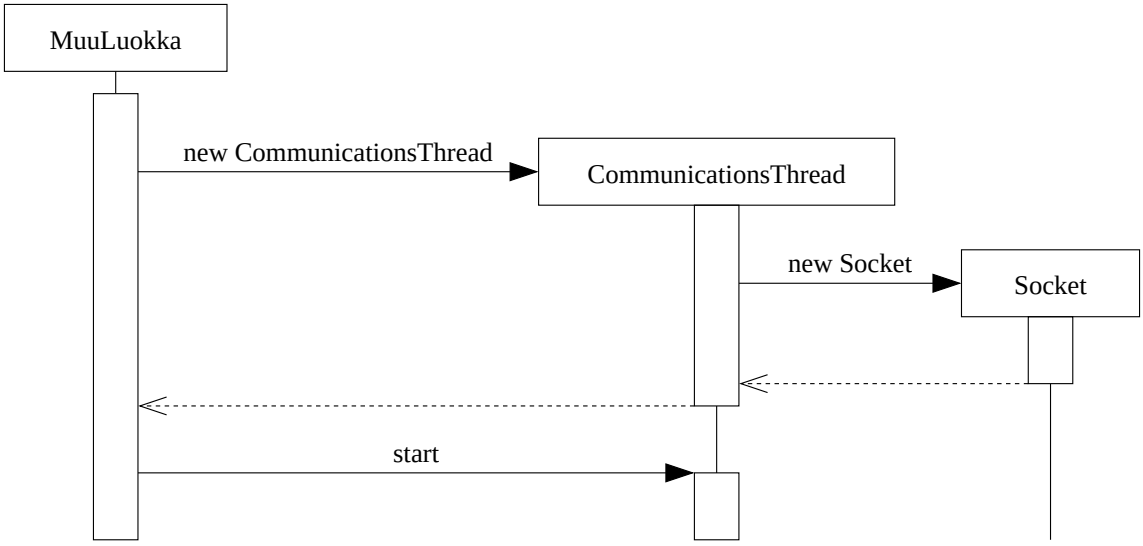
Tietoliikennekomponentti sisältää ajon aikana useita rinnakkaisia **säikeitä**. Tämän vuoksi on tärkeää muistaa rinnakkaisuuden aiheuttamat synkronointivaatimukset. Tietoliikennekomponentin käyttäjää tämä koskee siten, että viestinvälitysluokkien kutsumissa metodeissa käsitellyt kentät täytyy suojata esimerkiksi Javan *synchronized*-ilmaisulla. Jos vaikkapa kuvitteellinen *CatDogServices.modifyMilk*-metodi kutsuu luokan *Cup* ilmentymän metodia *setMilk*, joka puolestaan muuttaa *Cup*-olion kenttää *milk*, tulisi se toteuttaa esimerkiksi seuraavasti.

```
public class CatDogServices {
      private Cup cup;
      public modifyMilk(Milk newMilk) {
            cup.setMilk(newMilk);
      }
}
```

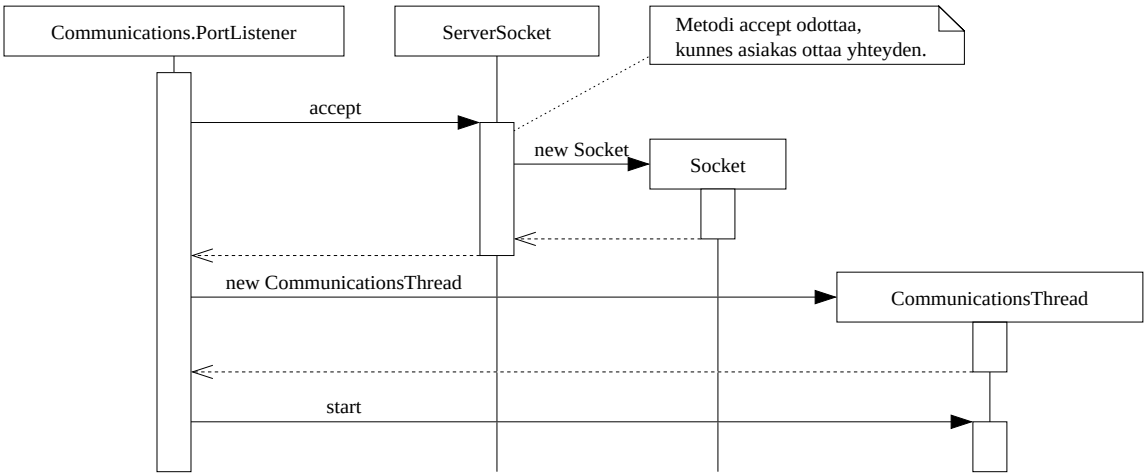
```
public class Cup {
      private Milk milk;
      public setMilk(Milk newMilk) {
            synchronized (this.milk) {
                  this.milk = newMilk;
            }
      }
}
```




*Kuva 12: Sekvenssikaavio palvelimen käynnistämisestä. Sekä Communications- että PortListener-oliot jäävät suoritukseen omina säikeinään.*




*Kuva 13: Sekvenssikaavio asiakkaan käynnistämisestä ja yhteyden muodostamisesta palvelimeen. Yhteys on muodostunut Socket-ilmentymän synnyttyä. CommunicationsThread-olio jää suoritukseen omana säikeenään.*




*Kuva 14: Sekvenssikaavio yhteyden muodostamisesta tietoliikenteen palvelin-puolella.*

# 5 Kehitystä ja ylläpitoa koskevia huomautuksia

## 5.1 Bundle-luokka

Bundle-luokan ***sort*-metodi** toimii siten, että se järjestää Bundlen sisältämät PrintJobit Vector-taulukon vierekkäisiin indekseihin vain tulostajan käyttäjätunnuksen perusteella. Tämän järjestyksen perusteella käytännössä tulostushallinta päättää töiden tulostusjärjestyksen nipun sisällä. *Sort*-metodia voisi kehittää siten, että se järjestää työt nipun sisällä myös esimerkiksi sivumäärien mukaan.

## 5.2 Tietoliikenneprotokolla

Nipun tietoliikenteellä on **vakioyläraja vastaanotettavien viestien koolle**. Raja on oletusarvoisesti sata kilotavua, ja sen pitäisi riittää kaikkeen toimintaan jopa silloin, kun niput sisältävät satoja tai jopa tuhansia tulostustöitä. Nippujen työmäärän kasvaessa tietoliikenteen raja tulee kuitenkin lopulta vastaan. Se ilmenee ajonaikaisena virheilmoituksena "Protokolla tukehtui. Yhteys suljetaan".

Viestien koon ylärajaa voi kasvattaa lähdekoodia muuttamalla. Muutos tulee tehdä tiedostoon nippu/communications/NippuProtocol.java seuraavaan kohtaan:

```
public NippuProtocol() {
      this(100 * 1024);
}
```

Luokan *NippuProtocol* parametrittomassa konstruktorissa kutsutaan toista konstruktoria, jolle annetaan viestien enimmäiskoko tavuina. Tätä lukua voi kasvattaa tarpeen mukaan.

**Tertun tietoliikenteellä on oma vakioylärajansa** vastaanotettavien viestien koolle. Sitä voi muuttaa asetustiedostosta terttu.conf. Siellä on oletusarvoisesti rivi, joka ilmaisee viestien enimmäiskoon tavuina: tcpserver.MaxMessageSize=100000. Tätä lukua voi kasvattaa tarpeen mukaan. On järkevää pitää Tertun ja Nipun viestien koolle asettamat rajat samoina.

Tietoliikenneprotokollan **viestiä EF** voitaisiin käyttää kaikissa tietoliikenneyhteyksissä, mutta käytännössä niitä lähettää vain analysoija vastaanotolle. Vastaavasti vastaanotto on

ainoa komponentti, joka osaa tulkita EF-viestin (ja myös vain silloin, kun lähettäjä oli analysoija).

Tietoliikenteessä oletetaan, että kaikki viestit **saapuvat perille virheettöminä**. Vastaanottovaiheessa ei nimittäin tarkisteta tietoliikenneviestien sisältöä, jos otsake on tunnettu, vaan viesti vain yritetään suoraan tulkita. Tämä on tuotantokäyttöä ajatellen erittäin vaarallinen ominaisuus.

Vaikka Nipun tietoliikenneyhteyksiin ei liity todennusta eikä salausta, yhteyksien **todennus ja salaus** on mahdollista myöhemmin erikseen toteuttaa jollain ulkopuolisella tuotteella. Koska tietoturvaominaisuudet on kuitenkin pääosin sivuutettu ohjelmiston suunnittelun alkuvaiheessa, tuotteen mahdollinen laajentaminen tuotantokäyttöön sopivaksi saattaa aiheuttaa ennalta arvaamattomia tietoturvaongelmia. Tietoliikenneyhteyksien todennuksen ja salauksen puuttuminen ei ole Nipun virhe tai puute vaan vaatimusmäärittelyyn perustuva ominaisuus. (Katso käyttöohjeen luku 1.3.)

## 5.3 Terttu

On todettu, että kun Tertun ajoittajalla ei ole tulostustöitä ja sille lähetetään uusia töitä, ensimmäinen työ päätyy aina suoritukseen, ennen kuin seuraavia ehditään lähettää. Tämän vuoksi **ajoittaja ei koskaan yhdistä ensimmäistä työtä** muihin töihin. Koska tämä on ristiriidassa Nipun vaatimusmäärittelyn kanssa, on vastaanoton toimintaan jouduttu lisäämään mahdollisuus lähettää työt ajoittajaan valmiiksi niputettuina.

Koska työt joudutaan joka tapauksessa lähettämään ajoittajaan valmiiksi niputettuina, olisi itse asiassa mahdollista jättää koko Terttu pois. On mahdollista, että Terttu ei ehkä ole paras mahdollinen ydin Nipun kaltaiselle järjestelmälle, jossa palvelupyynnöt ovat eksakteja tulostustöitä. Tähän liittyvää tarkastelua esitetään enemmän liitteessä 3.

## 5.4 Kääreluokat

Analysoijalla ja tulostushallinnalla on kääreluokat, joiden kautta näitä luokkia kutsutaan. Kääreluokissa ei ole tässä ohjelmistoversiossa mitään lisätoiminnallisuutta, mutta niistä saattaa olla hyötyä myöhemmässä kehityksessä. Kääreluokat otettiin toteutusvaiheessa

käyttöön siitä syystä, että luokkien tietoliikenteen alustuksessa oli ongelmia, kun alustus suoritettiin yhdellä konstruktorin kutsulla. Toteutuksen loppuvaiheessa alustus saatiin toimimaan ilman kääreluokkiakin, mutta kääreluokat jäivät silti käyttöön.

## 5.5 Asetustiedostot

**Analysoijan vakioiden** ei tarvitsisi olla yhteisessä asetustiedostossa, vaan ne voisivat olla analysoijan komponenttikohtaisessa asetustiedostossa. Ne ovat nyt yhteisessä asetustiedostossa siitä syystä, että analysoijan komponenttikohtainen asetustiedosto otettiin käyttöön vasta toteutuksen loppuvaiheessa.

Asetustiedostot tarkistetaan monenlaisten **syntaksivirheiden** varalta, mutta kaikkia syntaksivirheitä ei havaita. Tämä johtuu ajanpuutteesta toteutusvaiheessa.