Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Perttu Raivio

Samanaikainen dokumenttien muokkaus keskitetyssä taltiointijärjestelmässä

Insinöörityö 30.3.2010

Ohjaaja: projektipäällikkö Lauri Svan
Ohjaava opettaja: yliopettaja Kirsti Äystö

Metropolia Ammattikorkeakoulu **Tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Perttu Raivio Samanaikainen dokumenttien muokkaus keskitetyssä taltiointijärjestelmässä 33 sivua 30.3.2010
Koulutusohjelma	Tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	projektipäällikkö Lauri Svan yliopettaja Kirsti Äystö

Insinöörityössä tutkittiin samanaikaista, hajautettua dokumenttien muokkausta keskitetyssä dokumenttien taltiointijärjestelmässä. Tarkoituksena oli selvittää tällaiseen muokkaukseen liittyvät ongelmakohdat ja tutkia niiden ratkaisuja sekä algoritmeja.

Aluksi dokumentoitiin olemassa olevan järjestelmän rakennetta ja käytettyjä tekniikoita. Sitten tunnistettiin keskeisin samanaikaisen dokumenttien muokkauksen ongelma, yhdenmukaisuuden hallinta. Tähän liittyvät rikkomukset eriteltiin ja todettiin joidenkin perinteisten ratkaisumenetelmien heikkoudet näiden torjumisessa.

Sitten tutustuttiin menetelmään nimeltä toiminnallinen muunnos, jonka avulla yhdenmukaisuuden rikkomuksia voidaan torjua. Esiteltiin toiminnallisen muunnoksen periaatteet ja teoria ja tutustuttiin muutamaan esimerkkiin käytössä olevista ohjelmistoista, joissa toiminnallista muunnosta on hyödynnetty.

Lopuksi vielä pohdittiin, miten toiminnallista muunnosta voidaan hyödyntää otettaessa samanaikainen dokumenttien muokkaus käyttöön nykyistä taltiointijärjestelmää laajennettaessa.

Liiketaloudellisista ja strategisista syistä tuloksista ei ole hyötyä nykyisen taltiointijärjestelmän jatkokehityksessä, joka on keskeytetty. Saavutettu tietämys on kuitenkin arvokas nykyisen järjestelmän korvaavan, uuden taltiointijärjestelmäprojektin kehitystyössä.

Hakusanat	samanaikaisuus, hajautettu tietojenkäsittely, toiminnallinen muunnos

Helsinki Metropolia University of Applied Sciences Abstract

Author	Perttu Raivio
Title	Concurrent editing in a centralized document repository system
Number of Pages	33
Date	30 March 2010
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Lauri Svan, Project Manager Kirsti Äystö, Principal Lecturer

This study examined issues concerning concurrent editing of documents in a centralized document repository system. The purpose was to identify the problems involved and study the solutions and algorithms used to solve them.

First, the existing repository system's architecture and the technologies involved were documented. Then the central problem of concurrent document editing, consistency maintenance was identified, as well as the types of consistency violations. Some traditional solutions were listed and the reasons why they do not work in a concurrent editing system were given.

Then a technique for solving consistency violations, called the operational transformation, was introduced. The principles and theory of operational transformation were discussed and some current applications of the technology were introduced.

Finally, the use of operational transformation in the further development of the current repository system was discussed.

For business and strategic reasons the current repository project has been put on hold, but the knowledge gained in this study will be useful in the development of a new repository

Keywords	concurrency, distributed computing, operational transformation

Sisällys

1 Johdanto

Tässä insinöörityössä selvitetään samanaikaista dokumenttien muokkausta keskitetyssä taltiointijärjestelmässä. Samanaikaisella dokumenttien muokkauksella on tarkoitus laajentaa ns. *Repository*-projektin yhteydessä toteutetun keskitetyn taltiointijärjestelmän toiminnallisuutta. Repository-projektin toteutuksesta on huolehtinut Tieto oyj asiakkaanaan Nokia oyj.

Keskitetty taltiointijärjestelmä on palvelinpohjainen järjestelmä, joka tallentaa dokumentteja keskitetysti, automaattisesti ja edellyttämättä käyttäjältä erityisiä toimia. Taltiointijärjestelmä huolehtii myös versionhallinnasta. Dokumentit on tuotettu graafisen Eclipse-kehitysympäristön avulla toteutetuilla editoreilla. Näitä editoreita ja siten myös taltiointijärjestelmää käyttävät etupäässä käyttöliittymä- ja graafiset suunnittelijat uusien matkapuhelinohjelmistojen suunnitteluun.

Dokumenttien samanaikaisella muokkauksella tarkoitetaan, että useammat käyttäjät voivat tehdä hajautetussa ympäristössä samanaikaisesti muutoksia samaan dokumenttiin siten, että kaikki muutokset päätyvät dokumentin lopulliseen, tallennettavaan versioon. Samalla käytännössä edellytetään sitä, että yhden käyttäjän tekemät muutokset näkyvät viipeettä muiden samaa dokumenttia muokkaavien käyttäjien editoreissa.

Tavoitteena tässä työssä on ensin dokumentoida olemassa olevan taltiointijärjestelmän pääpiirteet samanaikaisen muokkauksen näkökulmasta ja sitten selvittää samanaikaisen muokkauksen toteutuksen ongelmakohtia sekä tutkia niiden selvittämiseen tarvittavia ratkaisuja ja algoritmeja. Tämä selvitys toimii pohjana samanaikaisen muokkauksen toiminnallisuuden toteutukselle. Sen perusteella pitäisi kyetä aloittamaan tarvittavien ohjelmamoduulien suunnittelu ja toteutus.

Toteutettavan järjestelmän tavoitteet

Samanaikaiselle dokumenttien muokkaukselle on asetettu seuraavia toiminnallisia tavoitteita:

- Yhtä dokumenttia kohden on kerrallaan vain yksi jaettu istunto.
- Dokumentteja ei lukita – jos samaa dokumenttia yritetään ottaa muokattavaksi sen ollessa jo toisella käyttäjällä muokattavana, avataan jaettu istunto.
- Jaetut dokumentit ovat jatkuvasti yhtenäiset ja niihin tehdyt muutokset toteutetaan synkronisesti.
- Vasteajat eivät saa kasvaa liian suuriksi.
- Paikallisesti muutokset toteutuvat välittömästi.
- Editorin on voitava näyttää visuaalisesti, kun johonkin dokumenttiin ollaan tekemässä muutoksia.
- Järjestelmän käyttöönoton ja määrityksen sekä istuntojen avaamisen on oltava yksinkertaisia.

Tyypillisessä jaetussa muokkausistunnossa on korkeintaan joitain kymmeniä käyttäjiä, ja yleisimmissä tapauksissa yksi käyttäjä tekee muutoksia muiden seuratessa tilannetta.

2 Nykyinen teknologia ja ympäristö

2.1 OSGi

OSGi Alliance on kansainvälinen standardointijärjestö, jonka tavoitteena on kehittää Java-pohjainen, etähallittava palvelualusta. OSGi-sovelluskehys on täydellinen ja dynaaminen komponenttimalli, joka toteuttaa tavallisista Java/VM-ympäristöistä puuttuvan moduulijärjestelmän [1].

Sovelluksia tai komponentteja, jotka on paketoitu jakelua varten (*bundles*), voidaan asentaa, käynnistää, pysäyttää, päivittää ja poistaa etäältä ilman järjestelmän uudelleenkäynnistystä. OSGi määrittää Java-pakettien ja -luokkien hallinnan hyvin tarkasti.

2.2 Eclipse

Eclipse on yhteisöllinen hanke, jonka projektien tarkoituksena on kehittää muokattavista ja laajennettavista sovelluskehyksistä (*framework*) sekä työkaluista muodostuva avoimen lähdekoodin kehitysalusta. Eclipse-alustalle rakennettujen kehitysympäristöjen avulla voidaan hallita ohjelmistotuotteiden koko kehityskaarta suunnittelusta ja kehittämisestä toimittamisen kautta ylläpitoon.

Eclipse itsessään muodostaa rungon, johon voidaan lisätä toiminnallisuutta erilaisten liitännäisosien (*plugin*) avulla. Versiosta 3.0 lähtien Eclipse on ollut täysin OSGi-määritelmien mukainen ja liitännäisosat ovatkin määritelmän mukaisia OSGi-paketteja.

Eclipsen varaan voidaan rakentaa myös itsenäisiä sovelluksia, jotka hyödyntävät vain tarvittavia liitännäisosia. Tällaista ”minimi-Eclipseä” kutsutaan Rich Client Platformiksi (*RCP*) ja sen avulla rakennettua sovellusta – itsenäistä, graafisella käyttöliittymällä varustettua sovellusta – Rich Client Applicationiksi.

IBM käynnisti Eclipse-projektin vuonna 2001 ja nykyään sitä hallinnoi voittoa tavoittelematon The Eclipse Foundation -säätiö [2].

2.3 Editorit

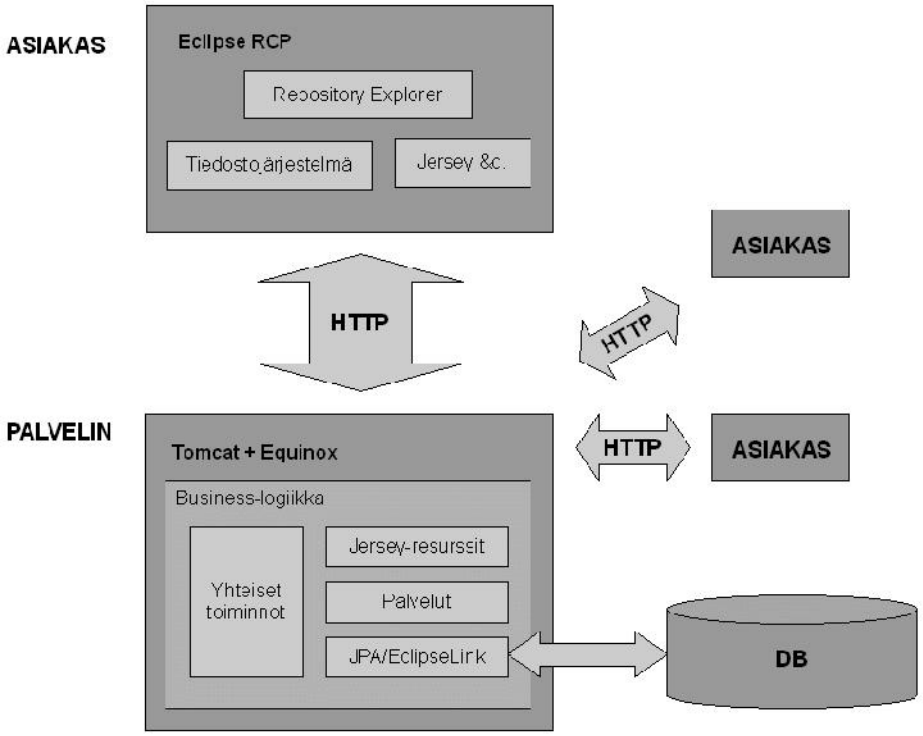
Asiakkaamme tarkoituksena on kehittää suunnittelijoiden käyttöön erilaisia Eclipse-alustalle toteutettuja editoreita, jotka on mukautettu nimenomaisiin suunnitteluprosessin vaiheisiin ja suunnittelijoiden tarpeisiin. Nämä editorit, joiden avulla suunnitellaan erilaisia käyttöliittymiä, käyttötapamalleja ym., muokkaavat erilaisia mallinnustietoja, kuten EMF- (Eclipse Modeling Framework) [3] tai GMF-muotoisia (Graphical Modeling Framework) [4] tiedostoja. Eclipse-ympäristössä näitä tiedostoja kutsutaan *resursseiksi*.

2.4 Taltiointijärjestelmä

Repository-projektissa on tavoitteena luoda yhtenäinen, keskitetty taltiointijärjestelmä erilaisille Eclipse-ympäristöön toteutettaville editoreille. Koska editorien aiotut käyttäjät ovat enemmän graafisen alan kuin tietotekniikan asiantuntijoita, taltioinnin on tapahduttava mahdollisimman automaattisesti ja intuitiivisesti (ns. KEVVM-malli – käyttäjältä ei voi vaatia mitään).

Taltiointijärjestelmän looginen rakenne vastaa tiedostojärjestelmän rakennetta. Tietyn projektin resurssit (tiedostot ja alihakemistot) on koottu säiliöihin (*Container*) ja käyttäjä voi valita, mitkä säiliöt hän ottaa käyttöön kehitysympäristössään. Taltiointijärjestelmä tukee myös tietojen kopiointia paikalliseen tiedostojärjestelmään (etätyöskentelyä varten) ja palautusta muokattuina takaisin keskitettyyn taltioon.

Taltiointijärjestelmän arkkitehtuuri on esitetty kuvassa 1.



Kuva 1. Taltiointijärjestelmän arkkitehtuuri

2.4.1 Asiakas

Taltiointijärjestelmä tarjoaa editoreille oman asiakasosan, jonka rajapintoja voi käyttää tavallisilla Java-metodikutsuilla. Editorien ei kuitenkaan tarvitse käyttää asiakasosaa, vaan ne voivat käyttää palvelinta suoraan REST-kutsuilla.

Asiakas toteuttaa oman versionsa Eclipsen tiedostojärjestelmästä (*EFS*) [5], joten sitä käytettäessä taltioidut tiedot näkyvät Eclipsen suuntaan samanlaisina kuin paikallisen tiedostojärjestelmän hakemistorakenne.

Asiakasosassa on myös erityinen puunäkymä taltiointijärjestelmään (Repository Explorer), jonka avulla käyttäjä voi hallita järjestelmän resursseja – esimerkiksi luoda uusia säiliöitä, poistaa niitä tai kopioida tietoja etäkäyttöä varten.

2.4.2 Palvelin

Taltiointijärjestelmän ytimen muodostaa palvelin, joka on toteutettu web-palvelimena. Editorit voivat käyttää joko palvelinrajapintaa suoraan käyttämällä REST-arkkitehtuurin mukaisia HTTP-kutsuja tai taltiointijärjestelmän asiakasosan rajapinnan kautta.

Palvelin toteuttaa saamiensa REST-kutsujen mukaisesti tarvittavat tietokantatoiminnot ja palauttaa niiden tuloksen asiakkaalle HTTP-vastauksena.

2.4.3 REST

Taltiointijärjestelmän palvelut on toteutettu ns. REST-arkkitehtuurin (Representational State Transfer) [6] mukaisesti.

REST on kevytrakenteinen arkkitehtuuri rakenteellisten resurssien käsittelyyn hajautetuissa web-ympäristöissä. Siinä jokaista resurssia vastaa vähintään yksi URI (*Uniform Resource Identifier*), joka yksilöi resurssin osoitteen ja nimen. REST-palvelut, kuten HTTP-protokolla, ovat tilattomia, eli jokainen palvelun HTTP-kutsu on itsenäinen, eikä asiakkaan ja palvelimen välille muodostu istuntoa.

Jokainen REST-kutsu sekä -vastaus sisältävät resurssin tunnisteen URI-muodossa. Tunnisteen avulla sovelluksen tilaa voidaan hallita, vaikka itse palvelu on tilaton.

REST toteuttaa CRUD-liittymän (Create-Retrieve-Update-Delete), jonka avulla resursseja voidaan luoda, hakea, muokata ja poistaa.

Taltiointijärjestelmässä käytetään Jersey-projektin java-pohjaista REST-toteutusta [7].

2.4.4 Tietokanta

Taltiointijärjestelmän tietokantayhteys on toteutettu EclipseLink JPA -palvelulla [8]. Se on Oraclen TopLink-palvelusta kehitetty säilyvyysratkaisu (*persistence solution*), joka kartoittaa järjestelmän oliot tietokantaolioiksi. Itse tietokantana voidaan käyttää erilaisia relaatiotietokantoja, käytössä ovat esimerkiksi Oracle, PostgreSQL ja Apache Derby.

3 Samanaikaisuuden ja yhdenmukaisuuden hallinta

3.1 Yhdenmukaisuuden hallinta

Merkittävin ongelma samanaikaisen resurssien muokkauksen toteutuksessa on yhdenmukaisuuden hallinta (*consistency maintenance*). Yhdenmukaisuuden hallinnalla tarkoitetaan sitä, että kaikissa samaa resurssia samanaikaisesti käsittelevissä editoreissa on näkymä resurssin senhetkiseen tilaan ja että editorilla tehdyt muutokset eli *operaatiot* toteutuvat resurssiin oikeanlaisina ja ilman ristiriitoja. Jälkimmäinen edellyttää samanaikaisten operaatioiden synkronointia. Edellinen edellyttää samanaikaisuuden hallintaa (*concurrency control*) [9] ja sen toteuttamiseen on kaksi ratkaisua: optimistinen ja pessimistinen samanaikaisuuden hallinta.

3.1.1 Optimistinen samanaikaisuuden hallinta

Optimistisella samanaikaisuuden hallinnalla (*Optimistic Concurrency Control*) tarkoitetaan sitä, että editori voi paikallisesti antaa vastineen sillä tehdylle muutokselle välittömästi ja lähettää ilmoituksen muutoksesta muille muokkausistunnon osapuolille. Jos samanaikaiset muutokset aiheuttavat ristiriitoja, on ilmoitusta muokattava sitä vastaanotettaessa niin, että se vastaa vastaanottajan tilaa ilmoituksen vastaanottohetkellä. Tällaista muutosilmoituksen ”korjaamista” kutsutaan toiminnalliseksi muunnokseksi (*Operational Transform*).

Optimistisen samanaikaisuuden hallinnan etuna on välitön palaute käyttäjälle, haittapuolena taas tilanteen synkronoinnin monimutkaisuus varsinkin, jos istuntoon osallistuu useita editoreita, joiden välinen tietoliikenne on suoraa.

3.1.2 Pessimistinen samanaikaisuuden hallinta

Pessimistisellä samanaikaisuuden hallinnalla (*Pessimistic Concurrency Control*) tarkoitetaan sitä, että editorien on saatava kuittaus palvelimelta, ennen kuin ne voivat toteuttaa haluamansa muutokset resurssiin paikallisesti. Tällöin palvelin huolehtii tapahtumien synkronoinnista ennen niiden toteutusta, eikä ilmoituksia tarvitse muuntaa,

koska ristiriitoja aiempien ilmoitusten kanssa aiheuttaneet muokkauspyynnöt yksinkertaisesti hylätään.

Pessimistisen samanaikaisuuden hallinnan etuna on synkronoinnin helpottuminen, mutta käytännössä se uhkaa aiheuttaa niin suurta vasteaikojen kasvamista, että toiminnalliseen muunnokseen perustuva optimistinen järjestelmä on toivottavampi.

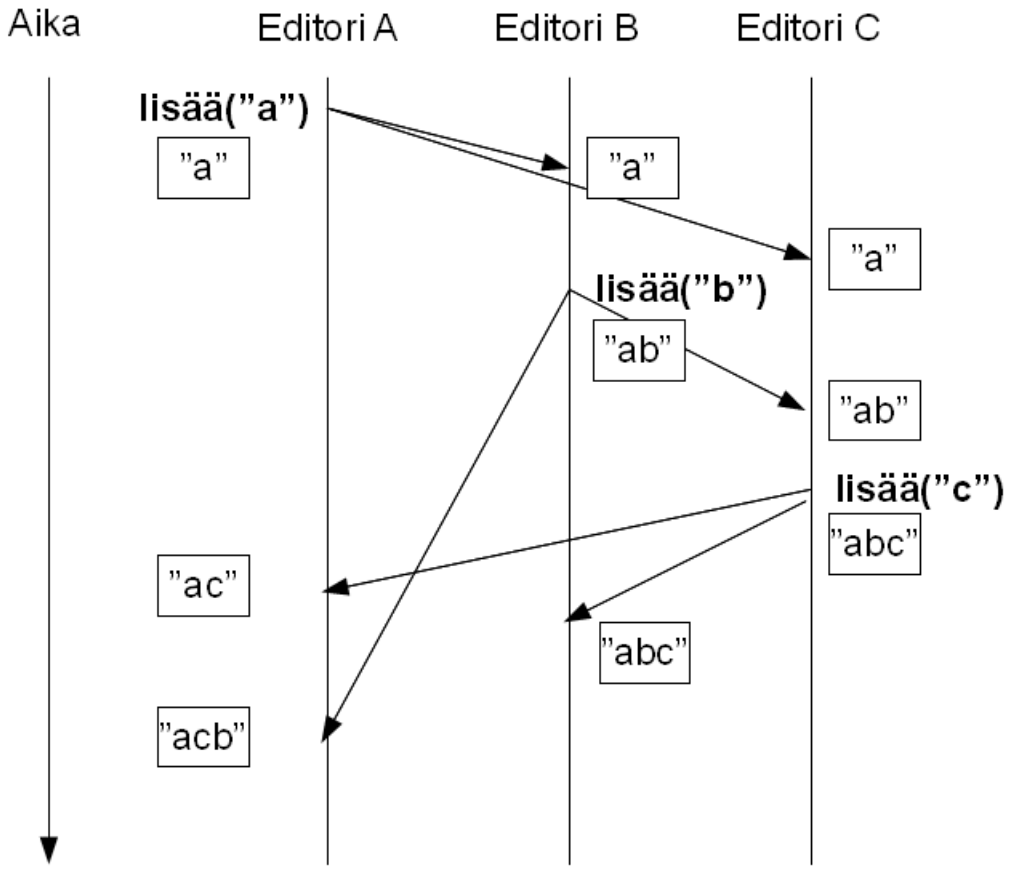
3.2 Yhdenmukaisuuden rikkomukset

Yhdenmukaisuuden rikkomuksia ovat divergenssi ja kausaliteetin sekä intention rikkomukset [10].

3.2.1 Divergenssi

Divergenssillä (*divergence*) tarkoitetaan sitä, että operaatiot saapuvat eri editoreihin eri viipeillä, jolloin ne saatetaan toteuttaa eri järjestyksissä ja lopputulokset poikkeavat toisistaan. Kuten kuvasta 2 havaitaan, poikkeavan käsittelyjärjestyksen vuoksi editorin A lopputilanne poikkeaa editoreiden B ja C lopputilanteesta.

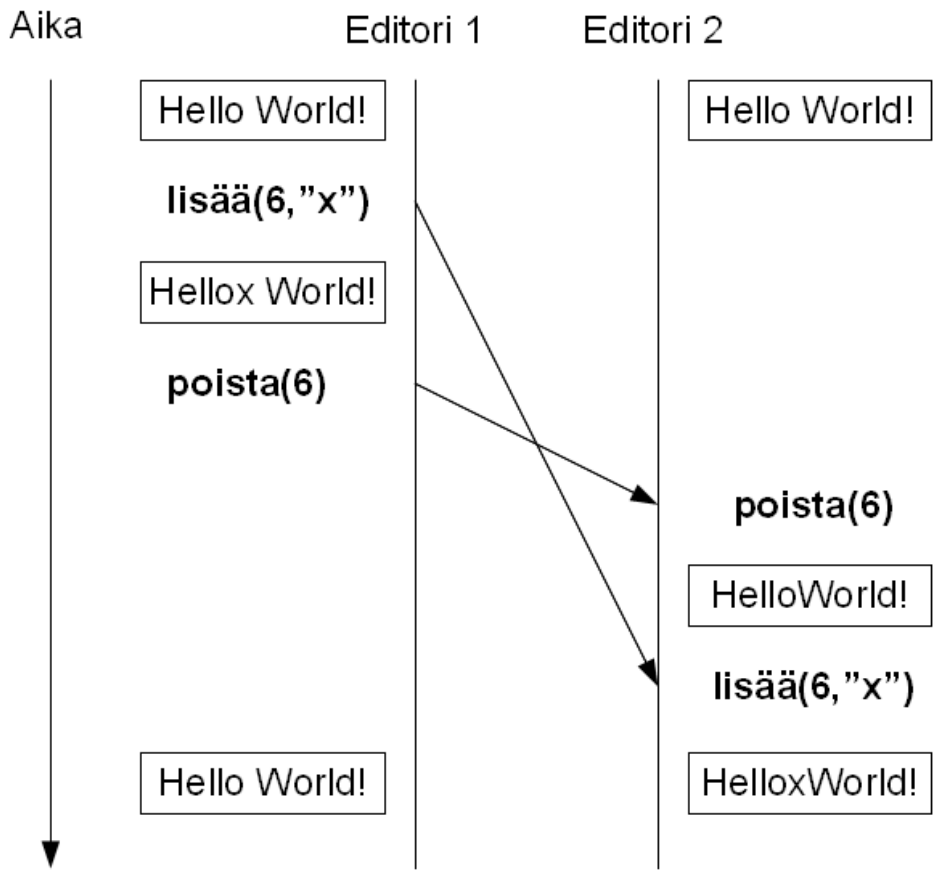
Hajautetun editointijärjestelmän voidaan sanoa olevan *lepotilassa*, kun kaikissa sen editoreissa on toteutettu sekä paikalliset että toisissa editoreissa luodut muokkaustapahtumat. Jos järjestelmän dokumenttien tilat ovat lepotilassa samat, järjestelmän voidaan sanoa olevan *konvergentti*, muutoin sitä vaivaa divergenssi.



Kuva 2. Divergenssi

3.2.2 Kausaliteetin rikkomus

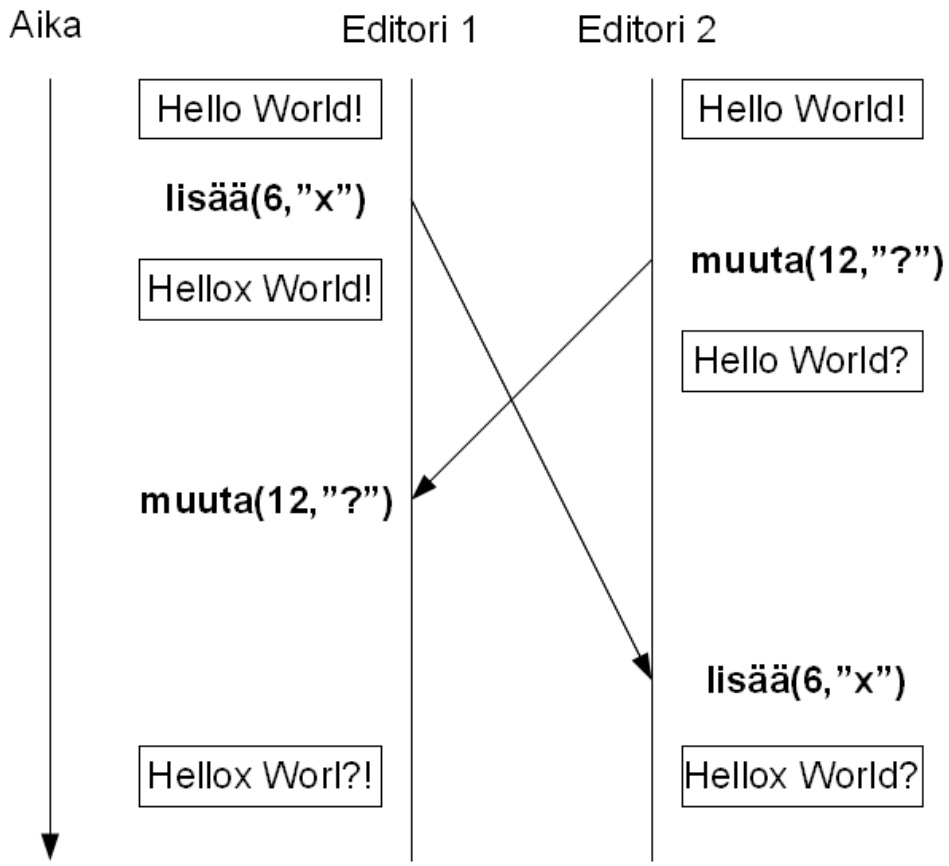
Kausaliteetin rikkomuksella (*causality-violation*) tarkoitetaan sitä, että peräkkäisiksi aiotut operaatiot saapuvat väärässä järjestyksessä, jolloin niiden syy- ja seuraussuhteet rikkoontuvat ja lopputulos on virheellinen (kuva 3). Kausaliteetin rikkomus on estettävissä esimerkiksi *sarjallistamalla* viestit, vaikkapa liittämällä niihin yksilöivä järjestysnumero.



Kuva 3. Kausaliteetin rikkomus

3.2.3 Intention rikkomus

Intention rikkomuksella (*intention-violation*) tarkoitetaan sitä, että suunnilleen samanaikaisesti luotujen operaatioiden vuoksi jokin operaatio kohdistuukin väärään kohteeseen, eli kohteen tila on ehtinyt muuttua ennen operaatiota, eikä enää vastaa operaation alkutilaa (kuva 4). Koska operaatioiden lähde on eri, ei sarjallistamisesta ole apua intention rikkomusten torjunnassa.



Kuva 4. Intention rikkomus

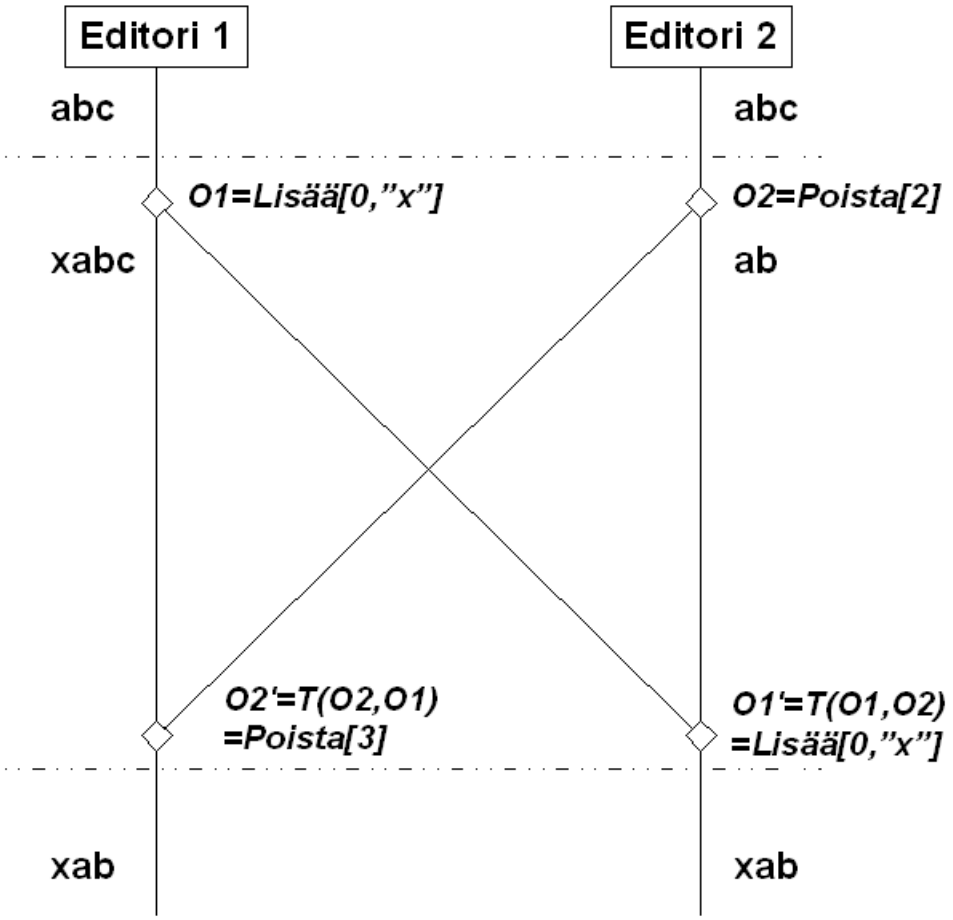
3.3 Perinteisiä ratkaisukeinoja

Aikojen saatossa yhdenmukaisuuden rikkomusten ratkaisuksi on esitetty joitain keinoja, jotka tarkemmin tutkien osoittautuvat vajavaisiksi. Editorit voidaan pakottaa operaatioiden luonnin **vuorotteluun**. Tämä merkitsee kuitenkin samanaikaisuuden menettämistä, eikä siten kelpaa ratkaisuksi. Erilaiset **lukitusmenetelmät** eivät todellisuudessa kykene estämään mitään yhdenmukaisuuden rikkomuksista ilman vuorottelun pakotusta, joten nekään eivät sovellu tarpeeseen.

Operaatioiden sarjallistamisella (*serialisation*) esimerkiksi numeroinnin avulla voidaan estää divergenssi ja kausaliteetin rikkomus, mutta ei intention rikkomusta.

3.4 Toiminnallinen muunnos

Niin sanotulla toiminnallisella muunnoksella tai OT-muunnoksella (*Operational Transformation*) voidaan ratkaista yhdenmukaisuuden rikkomukset. Toiminnallisella muunnoksella tarkoitetaan sitä, että ennen kuin operaatio toteutetaan, se muunnetaan ns. *OT-algoritmilla*, joka muuttaa operaation parametreja aiemmin toteutetut operaatiot huomioonottaen niin, että operaatio kohdistuu oikein.



Kuva 5. Toiminnallinen muunnos

3.4.1 Esimerkki OT-muunnoksesta

Kuvitellaan tilanne, jossa kahdessa editorissa käsitellään samanaikaisesti tiedostoa, jonka sisältö on merkkijono ”*abc*” (kuva 5). Editorit 1 ja 2 luovat samanaikaisesti seuraavat operaatiot:

O1 = Lisää[0, ”x”] (lisää merkki ”x” merkkijonon kohtaan ”0”)
O2 = Poista[2] (poista kohdasta ”2” merkki (eli ”c”))

Oletetaan, että editorissa 1 operaatiot suoritetaan järjestyksessä *O1 – O2*. Operaation *O1* jälkeen dokumentissa on merkkijono ”*xabc*”. *O1*:n suorituksen jälkeen *O2* on muunnettava ottamaan huomioon *O1* muunnosfunktiolla *T*, joka tässä tapauksessa kasvattaa paikkaparametria yhdellä siksi, että *O1* on kasvattanut merkkijonoa yhdellä merkillä ennen paikkaparametrin osoittamaa paikkaa. Joten *O2*:sta tulee *O2' = Poista[3]*. Kun muunnettu operaatio *O2'* suoritetaan merkkijonolle ”*xabc*”, poistetaan merkki ”*c*” (kuten oli tarkoituskin) ja tuloksena on merkkijono ”*xab*”. Jos muunnosta ei tehtäisi, operaation suorittaminen poistaisi merkin paikasta ”*2*”, eli kirjaimen ”*b*”, jolloin lopputulos olisi virheellisesti ”*xac*”.

3.4.2 Toiminnallisen muunnoksen teoriaa

Seuraavassa on kuvattu toiminnallisen muunnoksen teoriaa pääasiassa *High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System* -dokumentin [9] pohjalta.

Muunnosfunktion perusteet

Toiminnallinen muunnos perustuu muunnosfunktioon:

muunnos(a, p) = { a', p' }

jossa *a* ja *p* ovat alkuperäiset asiakkaan ja palvelimen muutosviestit. Viestien *a'* ja *p'* on oltava sellaiset, että jos asiakas toteuttaa viestit *a* ja *p'* ja palvelin viestit *p* ja *a'*, ne päätyvät samaan tilaan. Lisäksi funktion on tietysti pyrittävä toteuttamaan haluttu muutos mahdollisimman tarkkaan halutunmuotoisena. Esimerkiksi funktio

muunnos(a, p) = { poista kaikki, poista kaikki }

toteuttaa edellisen vaatimuksen, mutta ei toimi järin hyvin OT-muunnosfunktiona.

Esimerkiksi yhden merkin poiston muunnosfunktio on varsin yksinkertainen:

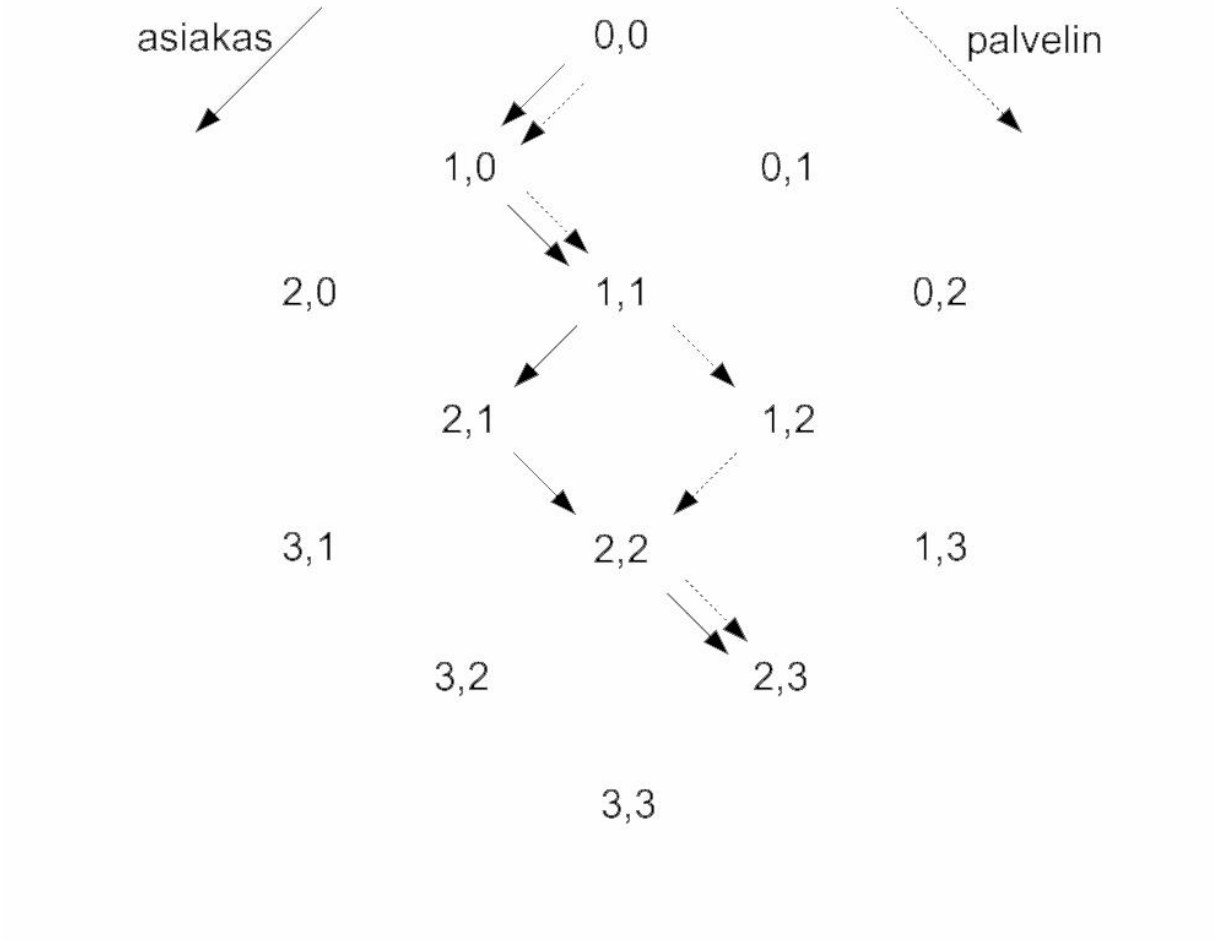
muunnos(poista x, poista y) =
{ poista x-1, poista y } jos x > y
{ poista x, poista y – 1 } jos x < y
{ no-op, no-op } jos x = y

Tässä funktiossa jäljempänä dokumentissa olevan merkin indeksistä vähennetään 1 aiemman poiston kompensoimiseksi. Muiden operaatioiden muunnosfunktiot eivät ole yhtä yksinkertaisia.

Tila-avaruus

Jupiter-dokumentissa [9] kuvataan dokumentin tila eräänlaisena tila-avaruutena, jonka läpi dokumentin tila kulkee sekä asiakkaassa että palvelimessa, päätyen lopulta samaan tilaan, vaikka kuljettu reitti olisi erilainen.

Kuvassa 6 jokaisen tilan nimi *a,p* kertoo käsiteltyjen asiakas- ja palvelinviestien määrän. Jos asiakas on tilassa *(2,3)*, se on luonut ja käsitellyt kaksi omaa viestiään sekä vastaanottanut ja käsitellyt kolme viestiä palvelimelta.

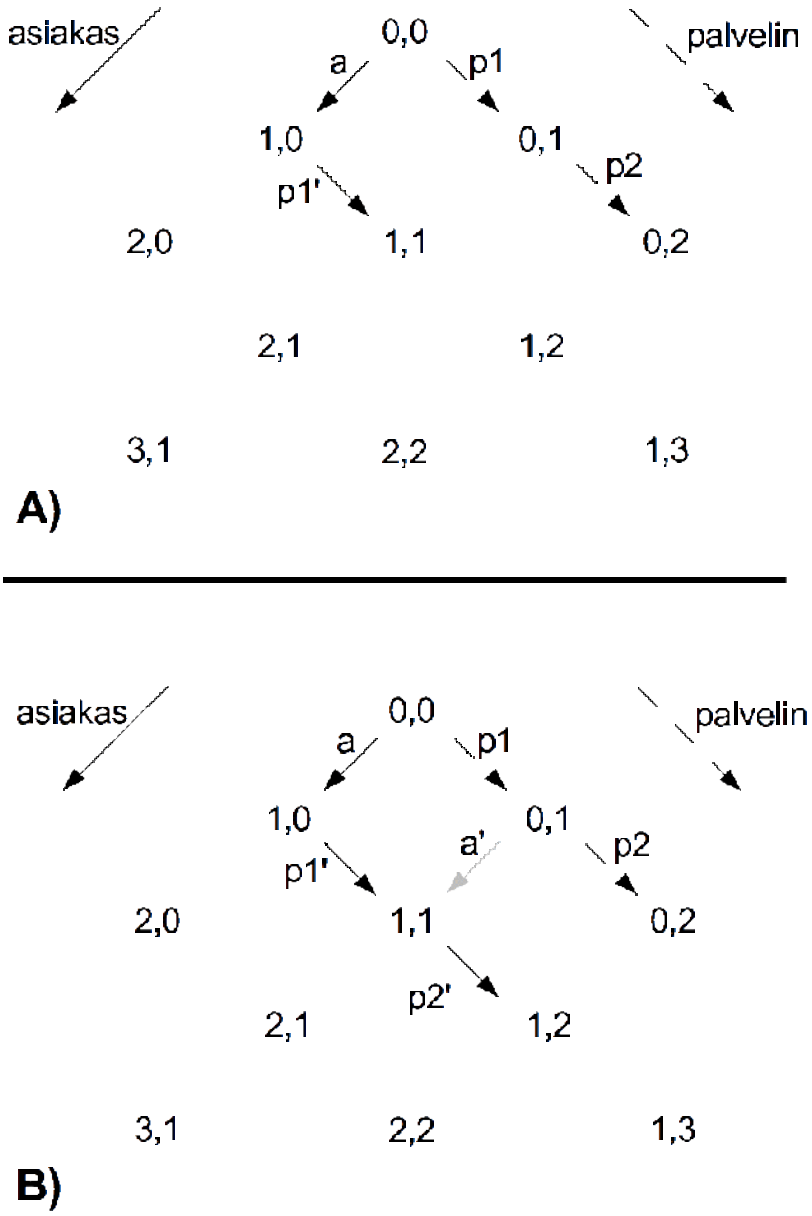


Kuva 6. Tila-avaruus

Viestejä toteutettaessa asiakas ja palvelin siirtyvät alaspäin tila-avaruudessa. Jos operaatiot suoritetaan samassa järjestyksessä (ristiriitoja ei ilmene), siirtyminen tapahtuu samaa reittiä pitkin. Jos palvelin ja asiakas luovat operaatioita samanaikaisesti, reitit erkanevat. Kuvassa 6 asiakas ja palvelin siirtyvät samaa reittiä tilaan *(1,1)* suorittamalla ensin asiakkaan operaation ja sitten palvelimen operaation. Tässä tilassa asiakas ja palvelin suorittavat eri operaatiot, minkä seurauksena ne siirtyvät tiloihin *(2,1)* ja *(1,2)*. Tällöin molemmat vastaanottavat toistensa viestit ja muunnosfunktion avulla siirtyvät kumpikin taas samaan tilaan *(2,2)*. Lopuksi palvelin luo operaation, jonka seurauksena sekä asiakas että palvelin siirtyvät tilaan *(2,3)*.

Järjestelmä lisää viesteihin operaation alkutilan. Samanaikaisuuden hallinta havaitsee ristiriidat poikkeavista alkutiloista ja ratkaisee ne muunnosfunktion avulla. Algoritmin tarkoitus on taata, että asiakas ja palvelin saavuttavat saman lopputilan riippumatta siitä, kuinka kauas toisistaan ne etääntyvät tila-avaruudessa.

Muunnosfunktio toimii poikkeaville reiteille tila-avaruudessa, kun lähtökohta on sama tila, ja sen tuloksena päädytään taas samaan tilaan. Jos asiakkaan ja palvelimen tilat erkanevat yhtä askelta etäämmälle, tilanne muuttuu monimutkaisemmaksi (kuva 7). Tilanteessa ***A)*** asiakas on toteuttanut operaation *a* ja vastaanottanut ristiriidan aiheuttavan viestin *p1*. Muunnosfunktiosta saadun viestin *p1* seurauksena asiakas siirtyy tilaan *1,1*. Tämän jälkeen palvelin luo operaation *p2*, jonka alkutila on *0,1* ja osoittaa siten, ettei palvelin ole toteuttanut operaatiota *a*. Tässä tilanteessa asiakas ei voi käyttää funktiota *muunnos(a, s2)*, koska operaatioiden *a* ja *s2* alkutilat eivät ole samat.



Kuva 7. Järjestysmuutos tila-avaruudessa

Ratkaisu on esitetty kuvassa ***B)***. Kun asiakas laskee operaation *s1'*, sen on tallennettava myös muistiin muunnosfunktion palauttama operaatio *a'*. Tämä on hypoteettinen operaatio, jonka avulla asiakas *olisi voinut* siirtyä tilasta *(0,1)* tilaan *(1,1)*. Kun asiakas ottaa vastaan viestin *p2*, se voi muistamansa operaation *a'* avulla laskea

muunnos(a', p2) = { a'', p2' }

Toteuttamalla operaation *p2'* asiakas päätyy tilaan *(1,2)*. Jos palvelin on nyt toteuttanut asiakkaan viestin, se myös on tilassa *(1,2)*. Ellei se ole sitä toteuttanut, sen seuraavan operaation alkutila on *(0,3)*, joten myös operaatio *a''* on tallennettava muistiin.

Algoritmi

Jupiter-dokumentissa [9] on luonnosteltu OT-algoritmia kokonaisuudessaan. Seuraava pseudokoodi kuvaa ristiriitojen ratkaisualgoritmin sekä asiakkaassa että palvelimessa.

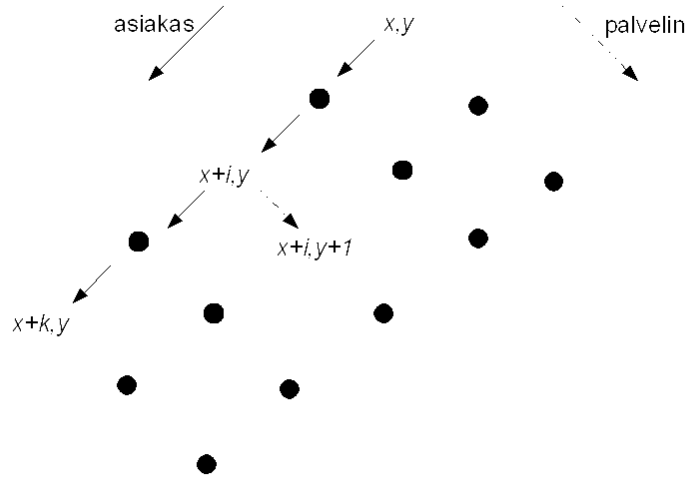
```
int luodutViestit = 0;    /* luotujen viestien määrä */
int vastViestit = 0;      /* vastaanotetut viestit */
jono lähtevät = {};

Luo(op) {
  toteuta op paikallisesti;
  lähetä(op, luodutViestit, vastViestit);
  lisää(op, luodutViestit) jonoon lähtevät;
  luodutViestit = luodutViestit + 1;
}

Vastaanota(viesti) {
  /* ohita kuitatut viestit */
  for v in (lähtevät) {
      if (v.luodutViestit < v.vastViestit)
          poista v jonosta lähtevät
  }
  /* OLETUS viesti.omatViestit == vastViestit */
  for i in [1..pituus(lähtevät)] {
      /* muunna uusi ja jonossa olevat viestit */
      {viesti, lähtevät[i]} =
          muunnos(viesti, lähtevät[i]);
  }
  toteuta viesti.op paikallisesti;
  vastViestit = vastViestit + 1;
}
```

Tilannetta voidaan tarkastella asiakkaan kannalta (asiakkaan ja palvelimen toiminta on identtistä), kuten kuvassa 8 on esitetty. Viimeisin tunnettu palvelimen tila on *(x,y)*,

jonka jälkeen asiakas on luonut ja lähettänyt *k* viestiä, joten asiakkaan tila on *(x+k,y)*. Nämä viestit ovat tallessa jonossa *lähtevät*. Pseudokoodin muuttujan *omatViestit* arvo on *x+k* ja muuttujan *vastViestit* arvo on *y*.

Uuden viestin luonti tässä tilanteessa on yksinkertaista: toteutetaan paikallisesti operaatio, joka muuttaa tilaksi *(x+k+1, y)*, lähetetään viesti siitä palvelimelle ja lisätään se lähtevien viestien jonoon.

Palvelinviestiä vastaanotettaessa tiedetään, että sen alkutilan täytyy olla jossain välillä *(x,y)* ja *(x+k,y)*, nämä mukaan lukien, riippuen siitä, kuinka monta asiakkaan *k*:sta viestistä palvelin on toteuttanut ennen oman viestinsä luontia. Oletetaan, että palvelin-viestin alkutila on *(x+i,y)* ja sen seurauksena palvelin siirtyy tilaan *(x+i,y+1*). Ensin asiakas voi unohtaa tallentamansa viestit tilasta *(x,y)* tilaan *(x+i,y)* tarpeettomina. Sen jäkeen kaikille tallennetuille viesteille suoritetaan muunnosfunktio saapuneen viestin kanssa. Tuloksena asiakas saa viestijonon, jonka avulla siirrytään tilasta *(x+k,y)* tilaan *(x+k,y+1)*, joka suoritetaan paikallisesti. Jonon jokaisen viestin muunnos tallennetaan, jolloin asiakkaalla on uusi alkutilannetta vastaava tilanne, viestijono viimeisimmästä tunnetusta palvelintilasta *(x+i,y+1)* senhetkiseen asiakkaan tilaan *(x+k,y+1)*. Asiakas on valmis luomaan tai vastaanottamaan seuraavan viestin.



Kuva 8. Muunnosalgoritmin toiminta tila-avaruudessa

Toiminnallinen muunnos Google Wave -protokollassa

Google Wave on esimerkki edistyneestä hankkeesta, jossa hyödynnetään toiminnallista muunnosta jaetun tiedon samanaikaisuuden ja yhdenmukaisuuden hallitsemiseksi. Se on hakukoneensa avulla maailmanmaineeseen ponnahtaneen Googlen yhteisöllinen projekti yhteistyö- ja kommunikointituotteen luomiseksi. Google Wavessä *Wave* on kokoelma resursseja, *Wavelet*tejä, operaatiot määrittelevät Waveletin tilan saman palvelimen asiakkaissa ja palvelinten välisen samanaikaisuuden hallinnan puolestaan määrittelee *Google Wave Federation* -arkkitehtuuri.

Google Wave -protokollaa [11] varten edellä esitettyjä toiminnallisen muunnoksen periaatteita on hieman viritelty. Suurin muutos on, ettei asiakas saa lähettää uutta viestiä ennen kuin se on vastaanottanut kuittauksen edellisestä viestistä palvelimelta. Jupiterin mukaisessa toiminnallisessa muunnoksessa palvelimen on pidettävä yllä erillisiä tila-avaruustietoja jokaista asiakas-palvelin -paria varten. Tämä rasittaa sekä palvelimen muistia että laskentaa, koska muunnokset on laskettava jokaiseen tila-avaruuteen erikseen.

Google Waven muunnoksessa palvelin kuittaa saamansa viestin sitten, kun se on muuntanut sen, toteuttanut sen omaan dokumenttikopioonsa ja jaellut muunnosviestin muille asiakkaille. Tällä välin tehdyt muutosoperaatiot asiakas voi tallentaa puskuriin myöhemmin viestittäviksi. Kun asiakas saa kuittauksen, se voi lähettää seuraavan muutosviestin siten, että palvelimen todellinen tila on valmiiksi alkutilana. Tällöin palvelimen ei tarvitse pitää tallessa kuin yksi tila-avaruus, jossa on sen toteuttamien operaatioiden historiatieto. Vastaanotetut viestit on vain muunnettava tätä historiaa vasten ja sitten jaeltava asiakkaille.

Haittapuolena Google Waven kuittausvaatimuksessa on se, että asiakas näkee muiden asiakkaiden tuottamia operaatiorypäitä vain palvelimen ja muiden asiakkaiden välisen kommunikointi- ja käsittelykierroksen muodostaman jakson välein. Palvelimen keveyden ja yksinkertaisuuden tuomaa etua on pidetty kuitenkin suunnittelussa merkittävämpänä tekijänä kuin tämän viipymän tuomaa haittaa.

ECF, synkronointi ja Cola

Eclipse Communication Framework [12] on Eclipse-hankkeen projekti, jonka tarkoituksena on toteuttaa sovelluskehys erilaisten Eclipse-sovellusten välisen tietoliikenteen helpottamiseksi ja yhdenmukaistamiseksi. Se on omiaan juuri samanaikaisuuden hallinnan tietoliikennetarpeisiin ja sen osana on sync-paketti, jonka varaan voi rakentaa etädokumenttien synkronointijärjestelmiä, sekä tekstidokumenteille että mallinnustiedoille. ECF-projektin puitteissa on toteutettu myös *DocShare*-järjestelmä tekstitiedostojen samanaikaisen muokkauksen esimerkiksi. DocShare käyttää Cola-nimistä algoritmia [13] yhdenmukaisuuden hallintaan. Cola-synkronointialgoritmi perustuu Jupiter-dokumentissa luonnosteltuihin periaatteisiin – asiakas-palvelin -malliin, toiminnalliseen muunnokseen ja kahdenkeskiseen tietoliikenteeseen.

Kahdenkeskinen tietoliikenne

Yhdenmukaisuuden hallinta on mahdollista suorittaa *n-suuntaisessa tietoliikenteessä* editorien välillä, mutta koska käytössä on joka tapauksessa asiakas-palvelin -arkkitehtuuri, on huomattavasti yksinkertaisempaa hallita kaksisuuntaista tietoliikennettä palvelimen ja yhden editorin välillä. Tämä johtaa siihen, että palvelin pitää yllä ajantasaista tilannetietoa dokumentin tilasta ja huolehtii operaatioiden jakelusta kaikille istunnossa oleville editoreille. Kahdenkeskisessä tietoliikenteessä kausaliteetin rikkomukset voidaan ehkäistä käyttämällä tietoliikenneprotokollaa, joka estää tietojen saapumisen väärässä järjestyksessä.

Kaksisuuntaisella tietoliikenteellä toteutettu yhdenmukaisuuden hallinta toimii periaatteessa seuraavasti:

- Asiakas luo operaation.
- Asiakas toteuttaa operaation paikallisesti.
- Asiakas lähettää palvelimelle viestin operaatiosta.
- Palvelin vastaanottaa operaatioviestin.
- Mikäli operaatio aiheuttaa ristiriidan: palvelin **muuntaa** operaation.
- Palvelin toteuttaa operaation paikallisesti.

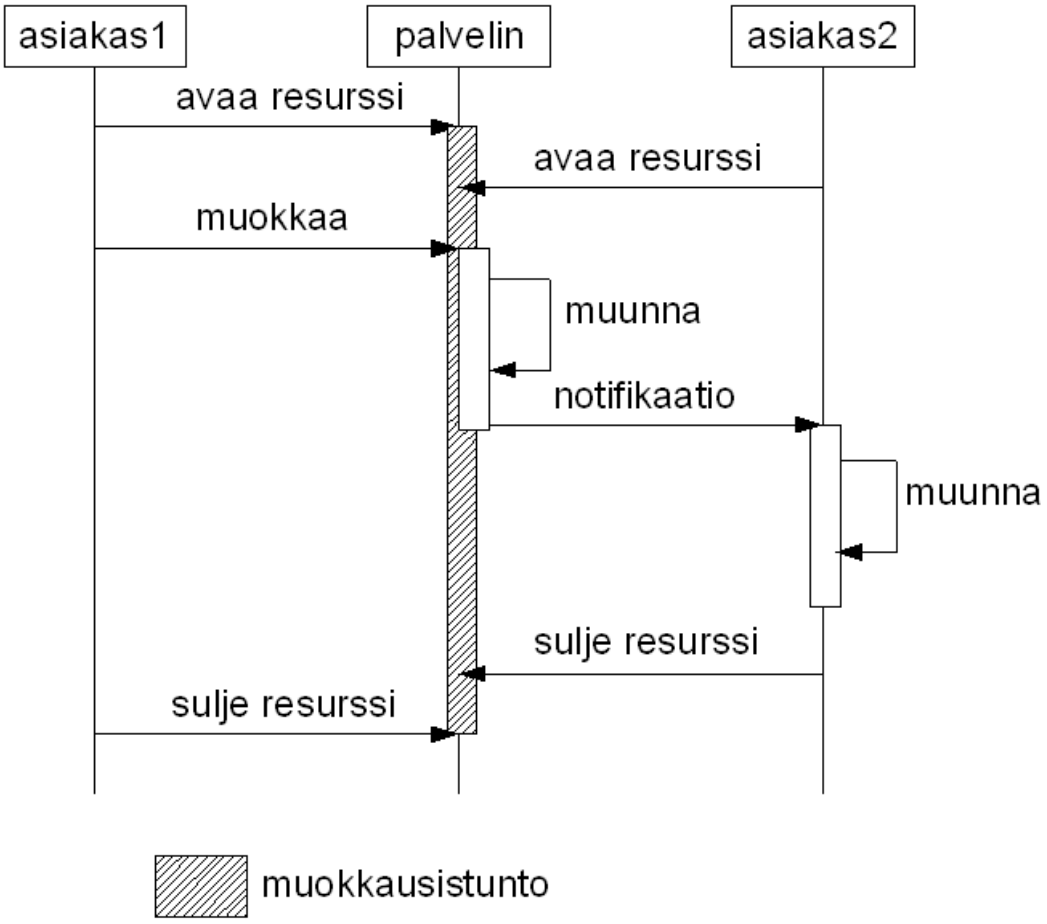
- Palvelin lähettää viestin kaikille muille asiakkaille operaatiosta.
- (Jokainen muu) asiakas vastaanottaa operaation.
- Mikäli operaatio aiheuttaa ristiriidan: asiakas **muuntaa** operaation.
- Asiakas toteuttaa operaation paikallisesti.

Tässä mallissa palvelimessa on teoriassa jatkuvasti ikään kuin muokattavan dokumentin pääversio, mutta synkronointi asiakkaiden kanssa tapahtuu pelkästään muutoksia kuvaavien operaatioiden avulla.

3.5 Repository-projektin ratkaisu

Tutkitaan lopuksi, miten edellä havaittua voidaan hyödyntää nykyisen Repository-taltiointijärjestelmän toiminnallisuuden laajentamiseen.

3.5.1 Arkkitehtuuri

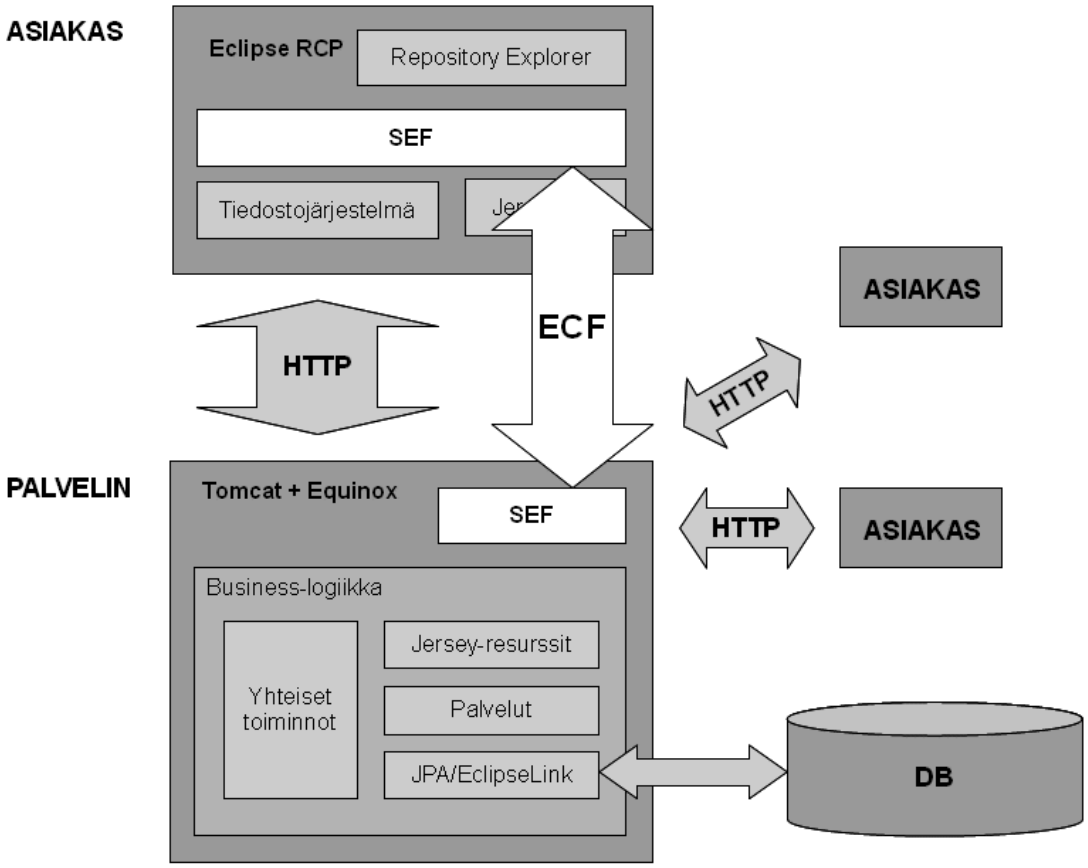


Kuva 9. Muokkausistunto

Koska Repository-projekti joka tapauksessa perustuu asiakas-palvelin -arkkitehtuuriin, on luonnollinen arkkitehtuurivalinta Jupiterin kaltainen, kahdenkeskeiseen tieto-liikenteeseen perustuva järjestelmä. Olemassa olevassa asiakkaassa jaettu dokumenttien käsittely tunkeekin lonkeronsa tiedostojärjestelmän (EFS) toteutukseen, jossa resurssien avausyrityksen yhteydessä tarkistetaan, onko kyseinen resurssi jo avattuna jollain toisella asiakkaalla ja joko käynnistetään jaettu muokkausistunto tai liitytään jo olemassa olevaan istuntoon. Toinen mahdollisuus on jaettujen istuntojen avaaminen eksplisiittisellä liittymäkutsulla, jolloin samanaikainen tiedoston avaaminen ei käynnistäisi jaettua istuntoa.

Istuntoa avattaessa palvelimen samanaikaisen muokkauksen ohjelmamoduuli rekisteröi itsensä kuuntelijaksi dokumentissa tapahtuville muutoksille. Asiakas puolestaan rekisteröityy kuuntelijaksi palvelimelta tuleville notifikaatioille, joiden avulla viestitään muutoksista muokattavassa resurssissa (kuva 9).

Samanaikaisen muokkauksen toiminteet voidaan kuvitella tavallaan sekä palvelin- että asiakaspuolella oleviksi, keskenään ECF:n kautta kommunikoiviksi ohjelman osiksi, jotka muodostavat samanaikaisen muokkauksen ohjelmakehyksen (kuva 10).



Kuva 10. Shared Editing Framework

3.5.2 ECF-synkronointi

ECF:n mukana tuleva DocShare-esimerkki on valmiiksi Eclipse-ympäristössä toteutettu ratkaisu tekstitiedostojen samanaikaiseen muokkaukseen, joten sitä voidaan käyttää pohjana synkronoinnin ja toiminnallisen muunnoksen toteuttamiseen Repository-järjestelmässä.

DocShare käyttää synkronoinnin toteutukseen `org.eclipse.ecf.sync`-pakettia, joka tarjoaa synkronointipalveluille ohjelmointikehyksen ja implementoi Cola-algoritmin tekstitiedostojen synkronointiin toiminnallista muutosta hyväksikäyttäen. Colan ja DocSharen tarjoamaa mallia hyödyntäen on toteutettava myös mallinnustiedostojen (EMF, GMF) synkronointi. DocShare rekisteröi itsensä dokumentin tai

tekstin valintatilan muutosten kuuntelijaksi toteuttamalla `IDocumentListener`- ja `ISelectionChangedListener`-liittymät. Näistä paikallisista muutoksista luodaan viestit etäkäyttäjille lähetettäväksi ja muunnettaviksi. Lisäksi DocShare kuuntelee ECF-kanavien tapahtumia, eli käytännössä ottaa vastaan muutosviestejä etäkäyttäjiltä. Myös näille suoritetaan toiminnallinen muunnos ennen niiden toteutettamista.

Tarvittava uusi toiminnallisuus liittyy siis mallinnustietojen synkronointistrategian suunnitteluun (toiminnallinen muunnos myös mallinnustiedoille) sekä istuntojen käynnistämiseen liittyviin toimiin.

3.5.3 Toiminnallinen muunnos

Mallinnustietojen toiminnallinen muunnos on suunniteltava puhtaalta pöydältä, sillä vaikka fyysisesti mallinnustiedostot ovat XML-muotoisia, eli tekstitiedostoja, loogisesti niitä ei voi käsitellä tekstitasolla, vaan on otettava huomioon tietosisällön semanttinen merkitys ja syntaktiset kokonaisuudet. Malli voi myös koostua useasta tiedostosta. Kuitenkin periaatteet ovat samat kuin tekstitiedon toiminnallisessa muunnoksessa, käsiteltävät tiedot vain ovat merkkien tai merkkijonojen sijasta suurempia, loogisia kokonaisuuksia.

Tämän synkronoinnin toteutukselle voi katsoa mallia ECF:n `org.eclipse.ecf.internal.sync.doc.cola`-paketista, jossa on toteutettu Cola-algoritmin mukainen toiminnallinen muunnos tekstitiedostoille.

3.5.4 Sovellusliittymä

Repositoryn sovellusliittymän (API) kannalta uudet toiminnot liittyvät tiedostojen tai mallien avaamiseen. Toimintamahdollisuuksia on oikeastaan kaksi:

- Tiedostojärjestelmän muutos tehdään siten, että aina tiedostoa tai mallia avattaessa tarkistetaan, onko se jo avattu johonkin toiseen editoriin. Jos tieto on jo muokattavana, avataan automaattisesti jaettu istunto. Tai vaihtoehtoisesti jo ensimmäinen tiedon avaaja käynnistää istunnon, johon myöhemmin avaavat liittyvät.

- Jaettu istunto avataan vain erityisestä kutsusta. Kutsu voidaan lähettää käyttöliittymästä joko tiedostoa avatessa tai jo avatun tiedoston kohdalla. Tässä tapauksessa voidaan toteutus tehdä myös niin, että tiedostoa voidaan samanaikaisesti käsitellä eri istunnoissa, jolloin siitä jää tietokantaan eri versiot.

4 Yhteenveto

Tässä insinöörityössä tutkittiin samanaikaiseen dokumenttien muokkaukseen liittyviä ongelmia ja niiden selvittämiseen soveltuvia ratkaisuja ja algoritmeja.

Merkittävimmäksi ongelmakohdaksi havaittiin yhdenmukaisuuden ja samanaikaisuuden hallinta eri muokkaajien välillä. Tunnistettiin kolme yhdenmukaisuuden rikkomuksen perustapausta, divergenssi sekä kausaliteetin ja intention rikkomukset, ja selvitettiin, miksi tyypilliset perinteiset ratkaisukeinot eivät auta estämään näitä samanaikaisessa dokumenttien muokkauksessa.

Sitten tutkittiin menetelmää nimeltä *toiminnallinen muunnos*, jonka todettiin soveltuvan yhdenmukaisuuden rikkomusten estämiseen. Toiminnallisen muunnoksen periaatteita ja teoriaa selvitettiin laajemmin ja tutustuttiin myös joihinkin sitä hyödyntäviin käytössä oleviin ratkaisuihin, kuten Google Wave –protokollan toiminnalliseen muunnokseen. Lopuksi todettiin, miten toiminnallista muunnosta voidaan hyödyntää, kun samanaikainen dokumenttien muokkaus otetaan käyttöön nykyisessä taltiointijärjestelmässä.

Insinöörityössä tehty selvitys toimii hyvänä lähestymiskohtana tämän melko laajan ongelma-alueen kysymyksiin ja pohjana käytännön toteutuksen suunnitteluun, joko olemassa olevan taltiointijärjestelmän puitteissa tai yleisemminkin.

Lähteet

1 OSGi Alliance. (WWW-dokumentti.) <http://www.osgi.org/Main/HomePage>. Luettu 29.3.2010.

2 Eclipse.org. (WWW-dokumentti.) <http://www.eclipse.org/>. Luettu 29.3.2010

3 Eclipse Modeling Framework. (WWW-dokumentti.) <http://www.eclipse.org/modeling/emf/>. Luettu 28.3.2010

4 Graphical Modeling Framework. (WWW-dokumentti.) <http://www.eclipse.org/modeling/gmf/>. Luettu 12.3.2010

5 Eclipse File System. (WWW-dokumentti.) <http://wiki.eclipse.org/index.php/EFS>. Luettu 22.3.2010

6 Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. 2000.

7 Jersey: RESTful Web services made easy. (WWW-dokumentti.) <https://jersey.dev.java.net/>. Luettu 22.3.2010

8 EclipseLink. (WWW-dokumentti.) <http://www.eclipse.org/eclipselink/>. Luettu 28.3.2010

9 Nichols, David A.; Curtis, Pavel; Dixon, Michael, Lamping, John. High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System. 1995.

10 Sun, C.; Yang, Y.; Zhang, Y; Chen, D. A Consistency Model and Supporting Schemes for Real-time Cooperative Editing Systems. 1996.

11 Wang, David ja Mah, Alex. Google Wave Operational Transformation. (WWW-dokumentti.) <http://www.waveprotocol.org/whitepapers/operational-transform>. Luettu 29.3.2010

12 Communication Framework Project. (WWW-dokumentti.) <http://www.eclipse.org/ecf/>. Luettu 12.3.2010

13 RT Shared Editing. (WWW-dokumentti.) <http://wiki.eclipse.org/RT_Shared_Editing>. Luettu 29.3.2010