LAPPEENRANNAN TEKNILLINEN YLIOPISTO
TIETOTEKNIIKAN OSASTO

TAPAHTUMANKÄSITTELY HAJAUTETUSSA YMPÄRISTÖSSÄ

Diplomityön aihe on hyväksytty Lappeenrannan teknillisen yliopiston tietotekniikan osaston osastoneuvoston kokouksessa 12.2.2003.

Työn tarkastajina toimivat professori Jari Porras ja DI Jyri Syväoja. Työn ohjaajana toimii DI Jyri Syväoja.

Lappeenrannassa 23.4.2003

Timo Mustonen
Punkkerikatu 7 B 25
53850 Lappeenranta
040 5267608

TIIVISTELMÄ

Tekijä: Mustonen, Timo

Nimi: **Tapahtumankäsittely hajautetussa ympäristössä**

Osasto: Tietotekniikan osasto

Vuosi: 2003

Paikka: Lappeenranta

Diplomityö. Lappeenrannan teknillinen yliopisto. 82 sivua, 30 kuvaa, 1 taulukko ja 1 liite.

Tarkastajat: Professori Jari Porras, DI Jyri Syväoja

Hakusanat: Tapahtuma, tapahtumankäsittely, ACID, DTP, hajautettu ympäristö

Tapahtumankäsittelyä pidetään yleisesti eräänä luotettavan tietojenkäsittelyn perusvaatimuksena. Tapahtumalla tarkoitetaan operaatiosarjaa, jonka suoritusta voidaan pitää yhtenä loogisena toimenpiteenä. Tapahtumien suoritukselle on asetettu neljä perussääntöä, joista käytetään lyhennettä ACID.

Hajautetuissa järjestelmissä tapahtumankäsittelyn tarve kasvaa entisestään, sillä toimenpiteiden onnistumista ei voida varmistaa pelkästään paikallisten menetelmien avulla. Hajautettua tapahtumankäsittelyä on yritetty standardoida useaan otteeseen, mutta yrityksistä huolimatta siitä ei ole olemassa yleisesti hyväksyttyjä ja avoimia standardeja. Lähimpänä tällaisen standardin asemaa on todennäköisesti X/Open DTP –standardiperhe ja varsinkin siihen kuuluva XA-standardi.

Tässä työssä on lisäksi tutkittu, kuinka Intellitel ONE –järjestelmän valmistajariippumatonta tietokanta-arkkitehtuuria tulisi kehittää, kun tavoitteena on mahdollistaa sen avulla suoritettavien tapahtumankäsittelyä vaativien sovellusten käyttäminen.

ABSTRACT

Author: Mustonen, Timo
Subject: **Transaction processing in distributed environment**
Department: Information technology
Year: 2003
Place: Lappeenranta

Master's Thesis. Lappeenranta University of Technology 82 pages, 30 figures, 1 table, and 1 appendix.

Supervisors: Professor Jari Porras, M.Sc. Jyri Syväoja
Keywords: Transaction, transaction processing, ACID, DTP, distributed environment

Transaction processing is currently recognized as the basic requirement for reliable data processing. Transaction is defined as series of operations and its execution can be seen as one logical event. Four basic rules have been set to transaction processing applications. These rules are also referred as ACID.

The need for reliable transaction processing increases in distributed environments. The main reason for this is that outcome of global transactions cannot be evaluated based on local operations and their return values. Distributed transaction processing has been subject to standardization on several occasions. However, there is no open standard that would be recognized by the whole industry. X/Open DTP standard family and especially the XA standard is probably nearest to that position.

This thesis also explores how Intellitel ONE system's vendor independent database architecture should be improved in order to support applications that require transaction processing capabilities.

ALKUSANAT

Tämä työ on tehty Intellitel Communications Oy:ssä Lappeenrannan teknillisen yliopiston tietotekniikan osastolle syksyn 2002 ja kevään 2003 aikana.

Haluan kiittää ihmisiä, joiden ansiosta diplomityön tekeminen on ollut mahdollista organisaatiomuutosten ja vaikeiden aikojen koettelemassa ilmapiirissä. Kiitän diplomityön ohjaajaa ja toista tarkastajaa DI Jyri Syväojaa asiantuntevista neuvoista ja mielipiteistä diplomityöni suhteen, DI Antti Ollilaista lukuisista neuvoista diplomityön kirjoitusprosessia ja siihen liittyvää byrokratiaa koskien ja tekn. yo Jarkko Tistelgreniä neuvoista Intellitel ONE –tietokantajärjestelmää koskevissa kysymyksissä.

Haluan kiittää myös diplomityöni tarkastajaa, professori Jari Porrasta, hyvästä opetuksesta kuluneiden vuosien aikana, ja hyvistä neuvoista diplomityöhöni liittyen.

Lisäksi haluan kiittää pikkusiskoani, kauppat. yo Hannele Mustosta, työn oikolukemisesta, ja fil. yo Heli Puhakkaa avusta tiivistelmien ja englanninkielisten termien kanssa. Lopuksi haluan lausua kiitokset vanhemmilleni ja erityisesti avovaimolleni Hanna Hiipakalle koko opiskeluajan kestäneestä tuesta, turvasta ja kannustuksesta.

Lappeenrannassa 23.4.2003

Timo Mustonen

SISÄLLYSLUETTELO

LYHENTEET

2PC	Two-Phase Commit
2PL	Two-Phase Locking
ACID	Atomicity, Consistency, Isolation, Durability
ACSE	Application Control Service Element
AE	Application layer Entity
API	Application Programming Interface
ASE	Application Service Element
BE	Backend
CCR	Commitment, Concurrency and Recovery
CICS	Customer Information and Control System
CORBA	Common Object Request Broker Architecture
CRM	Communications Resource Manager
CVOPS	C-based Virtual OPerating System
DB	Database
DBMS	Database Management System
DCE	Distributed Computing Environment
DTP	Distributed Transaction Processing
EAI	Enterprise Application Integration
EJB	Enterprise Java Bean
FE	Frontend
ICODI	Intellitel Communications Database Interface
IDL	Interface Definition Language
IIOP	Internet Inter ORB Protocol
IMS	Information Management System
J2EE	The Java 2 Platform, Enterprise Edition
J2ME	The Java 2 Platform, Micro Edition
J2SE	The Java 2 Platform, Standard Edition
JTA	Java Transaction API
JTS	Java Transaction Service
MACF	Multiple Association Control Function

NIM	Network Intelligence Middleware
OCI	Oracle Call Interface
OMG	Object Management Group
ONC	Open Network Computing
ORB	Object Request Broker
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OTS	Object Transaction Service
P2P	Peer-to-peer
PDU	Protocol Data Unit
POA	Portable Object Adapter
SACF	Single Association Control Function
SAO	Single Association Object
SMP	Symmetric Multiprocessing
SNA	Systems Network Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
STDL	Structured Transaction Definition Language
TCP/IP	Transmission Control Protocol / Internet Protocol
TDL	Task Definition Language
TP	Transaction Processing
TPASE	Transaction Processing Application Service Element
VTASK	Virtual Task
VTT	Valtion Teknillinen Tutkimuslaitos
WS-C	Web Services Coordination
WSDL	Web Services Description Language
WS-TX	Web Services Transaction
XDR	eXternal Data Representation
XML	eXtended Markup Language

1 JOHDANTO

Tapahtumien (transaction) luotettava käsittely on etenkin liiketoimintasovelluksista puhuttaessa yksi nykyaikaisten tietojärjestelmien perusvaatimuksista. Tapahtumien avulla voidaan tehokkaasti ryhmitellä ja hallita järjestelmissä tapahtuvaa eri tekijöiden välistä vuorovaikutusta. Vuorovaikutus voi olla käyttäjän ja järjestelmän välistä, tai siihen voi osallistua kaksi tai useampia järjestelmäkomponentteja. Tapahtumien tarkoituksena on osaltaan lisätä järjestelmän tehokuutta, mutta ennen kaikkea niiden tehtävänä on tuoda järjestelmään semanttista luotettavuutta. Niiden avulla varmistetaan, että käyttäjän järjestelmälle antamat käskyt ja tehtävät joko suoritetaan kokonaisuudessaan tai jätetään kokonaan suorittamatta.

Tapahtumilla ja niiden käsittelyllä on merkittävä rooli asiakas/palvelin-arkkitehtuuriin perustuvissa järjestelmissä. Tehokkuusvaatimukset ja tietokoneiden määrän merkittävä lisääntyminen ovat kuitenkin ajan kuluessa johtaneet hajautettujen järjestelmien huomattavaan yleistymiseen. Tämä muutos on tuonut tullessaan suuria haasteita sekä näitä järjestelmiä käyttäville sovelluksille että itse järjestelmille. Käsiteltävän tiedon hajaantuminen useampaan fyysiseen tai loogiseen paikkaan aiheuttaa ongelmia perinteiselle ohjelmistomallille, jossa toimenpiteiden onnistuminen tai epäonnistuminen voidaan päätellä paikallisten operaatioiden avulla. Tästä johtuen tapahtumankäsittelyn merkitys kasvaa entisestään hajautetuissa järjestelmissä ja voidaankin sanoa, että tapahtumankäsittely on menetelmä, jolla hajautettu tiedonkäsittely ja –hallinta saadaan tehtyä luotettavaksi ja tehokkaaksi.

1.1 Työn taustaa

Intellitel ONE -järjestelmä sisältää tietokanta-arkkitehtuurin, joka ei ole riippuvainen varsinaisesta tietokannasta tai sen toimittajasta. Tämän arkkitehtuurin varaan on rakennettu useita erilaisia sovelluksia ja samalla on huomattu, että tehokkaan tapahtumankäsittelyn puute arkkitehtuurissa aiheuttaa huomattavia lisävaatimuksia sen päälle rakennettaville sovelluksille. Tarvittava tapahtumankäsittely voidaan useissa

tapauksissa toteuttaa tietokantaproseduurien avulla. Tämä ratkaisumalli on kuitenkin ristiriidassa tietokantariippumattoman arkkitehtuurin kanssa. Ratkaisu aiheuttaa lisäksi ongelmia hajautettujen ympäristöjen tapauksessa, sillä tietokantaproseduurien sisältämä logiikka on rajoittunut joko yhteen tietokantaan tai yhden valmistajan tietokantoihin.

Yhteenvetona voidaankin sanoa, että Intellitel ONE:n tietokanta-arkkitehtuurista puuttuvat tapahtumankäsittelymekanismit rajoittavat sen käyttömahdollisuuksia ja vaikeuttavat sen avulla toteutettavien ja tapahtumankäsittelyominaisuuksia tarvitsevien sovellusten kehittämistä.

1.2 Työn tavoitteet

Työn tarkoituksena on selvittää millaisia ongelmia ja mahdollisia rajoituksia hajautetut järjestelmät aiheuttavat niissä suoritettavalle tapahtumankäsittelylle sekä tutkia ja kartoittaa hajautetuissa järjestelmissä käytettyjä tapahtumankäsittelytekniikoita, -standardeja ja –tuotteita. Työssä tutkitaan myös sitä, kuinka Intellitel ONE –järjestelmän tietokanta-arkkitehtuuriin voitaisiin lisätä tuki tehokkaalle, yksinkertaiselle ja luotettavalle tapahtumankäsittelylle. Ratkaisun tulisi lisäksi mahdollistaa Intellitel ONE -järjestelmän käyttö osana suurempaa hajautettua tapahtumankäsittelyjärjestelmää.

Työssä tuotetaan kattava selvitys hajautettujen ympäristöjen tapahtumankäsittelystä, tapahtumankäsittelyn toteuttamisesta Intellitel ONE –järjestelmän tietokanta-arkkitehtuuriin, sekä tehdään prototyyppitoteutus, jonka avulla erään mahdollisen ratkaisun toimivuutta voidaan arvioida.

2 TAPAHTUMAT JA NIIDEN KÄSITTELY

Tietojärjestelmien yhteydessä tapahtumalla voidaan tarkoittaa montaa eri asiaa. Yksi yleisesti käytetty määritelmä kertoo tapahtuman olevan jonkun ohjelman suoritus ja se johtaa johonkin lopputulokseen. Toinen taas määrittää tapahtuman joukoksi operaatioita. Määritelmiä on useita, mutta kaikkien perimmäinen tarkoitus on sama.

Tapahtuma käsitteen monimuotoisuus johtuu ainakin osittain siitä, kuinka siihen osallistuvat tahot kokevat sen. Loppukäyttäjä ei välttämättä näe tapahtumasta muuta kuin sen käynnistävän tapahtumapyynnön ja sen suorittamista seuranneen lopputuloksen, joten hänen kannaltaan tapahtuma voi tarkoittaa näitä asioita. Palveluntarjoajan tai järjestelmän ylläpitäjän käsite tapahtumasta voi taas olla hyvinkin erilainen, sillä hän näkee, mitä toimenpiteitä tapahtumapyyntö aiheuttaa itse järjestelmässä. Ohjelman kehittäjällä on myös oma katsontakantansa ja hän voi nähdä tapahtuman pikemminkin ohjelman lähdekoodina, kuin itse ohjelman suorituksena ja sen aiheuttamina seurauksina. [Ber1997] [Gra1993]

Tässä työssä tapahtuma määritellään joukoksi operaatioita, jotka muodostavat loogisesti yhden ja toisistaan riippuvaisen kokonaisuuden. Tapahtuman varsinaisesta suorittamisesta vastaa tapahtumaohjelma (transaction program). Tapahtumaohjelma taas on osa suurempaa kokonaisuutta, jota kutsutaan nimellä tapahtumankäsittelysovellus (transaction processing application). Tapahtumankäsittelysovellus sisältää sekä tapahtumaohjelmia että joukon muita osia, joita käytetään esimerkiksi tapahtumaohjelmien syötteen keräämiseen tai tulosten esittämiseen. Tapahtumankäsittelyjärjestelmä (transaction processing system) muodostaa kokonaisuuden, joka sisältää kaikki tapahtumien käsittelyyn tarvittavat osat. Tapahtumankäsittelyjärjestelmään kuuluu täten niin käyttöjärjestelmä, tietokonelaitteisto, tietokannat kuin muut tarvittavat sovelluksetkin. Tapahtumankäsittelyjärjestelmän koko voi vaihdella suuresti. Pienet järjestelmät voivat koostua yhdestä laitteesta ja siinä toimivista ohjelmista, kun taas suuret ja hajautetut järjestelmät voivat koostua ympäri maailmaa sijaitsevista palvelimista ja ne voivat käsitellä tuhansia tapahtumia sekunnissa. [Ber1997] [Gra1993]

Tapahtuman kulkua ohjataan pääasiassa kolmen eri operaation avulla:

- Käynnistä (start) operaatio määrittää uuden tapahtuman alkamisen. Tämän jälkeen suoritettavat operaatiot kuuluvat tähän määrättyyn tapahtumaan.
- Vahvista (commit) operaatio päättää tapahtuman ja ottaa voiman tapahtuman sisältämien operaatioiden tekemät muutokset.
- Peruuta (rollback) operaation avulla käynnissä oleva ja vahvistamaton tapahtuma voidaan peruuttaa siten, että kaikki tapahtumassa siihen asti suoritetut operaatiot perutaan. Peruutus tehdään usein atomäärisyysvaatimuksen täyttämiseksi automaattisesti silloin, jos jonkun tapahtumaan kuuluvan operaation suoritus on epäonnistunut.

Yksinkertainen esimerkki tapahtumasta ja sen hyödyllisyydestä on pankin maksuautomaatti. Käyttäjän maksaessa laskun hänen tililtään siirretään tietty määrä rahaa maksun saajan tilille. Tämä koko toimenpide voidaan nähdä yhtenä tapahtumana, vaikka siihen kuuluukin useita erillisiä osia:

1. Tarkasta onko käyttäjän tilillä riittävästi rahaa.
2. Tarkasta onko maksun saajan tili olemassa.
3. Vähennä määrätty summa maksajan tililtä.
4. Lisää määrätty summa maksun saajan tilille.

Pankin ohjelmiston tulee huolehtia siitä, että kaikki erilliset toimenpiteet suoritetaan onnistuneesti. Jos esimerkiksi ainoastaan viimeinen operaatio epäonnistuu, niin summa on hävinnyt maksajan tililtä mutta sitä ei ole lisätty maksun saajan tilille. Tämä on luonnollisesti tilanne, joka ei koskaan saisi toteutua. Tapahtumien ja sopivan tapahtumankäsittelysovelluksen avulla voidaan varmistua siitä, että koko tapahtuma suoritetaan alusta loppuun saakka. Mikäli joku tapahtumaan kuuluva toimenpide epäonnistuu, perutaan kaikkien toimenpiteiden vaikutus ja palautetaan käyttäjälle virheilmoitus.

2.1 ACID-vaatimukset

Tapahtumille on määritelty tiettyjä perusvaatimuksia, jotka niitä toteuttavien tapahtumaohjelmien tulee täyttää. Näistä perusvaatimuksista käytetään lyhennettä ACID. Lyhenne tulee englanninkielisistä sanoista:

- **Atomicity** eli atomäärisyys tarkoitettaa sitä, että tapahtuma käsitellään yhtenä kokonaisuutena, eikä se siten voi onnistua tai epäonnistua osittain. Mikäli yksikin tapahtumaan kuuluvista operaatioista epäonnistuu, niin koko tapahtuma tulkitaan epäonnistuneeksi. [Ber1997] [Gra1993]
- **Consistency** eli yhtenäisyysvaatimus määrää sen, että tapahtuman tulee säilyttää muokkaamansa resurssin yhtenäisyys. Eli jos muokattu resurssi, esimerkiksi tietokanta, on yhtenäisessä tilassa ennen tapahtuman suoritusta, niin sen tulee olla sitä myös tapahtuman suorituksen loputtua. Tapahtuman suorittaminen ei saa aiheuttaa vahinkoa muokattavan resurssin yhtenäisyydelle. [Ber1997] [Gra1993]
- **Isolation** eli eristyneisyysvaatimuksen täyttävässä järjestelmässä tapahtumia voidaan suorittaa rinnakkain ilman, että ne sotkevat toistensa suoritusta tai tuottavat vääränlaisia tuloksia. Tulokset ovat siis samanlaiset kuin jos tapahtumia ajettaisiin peräkkäin yksi kerrallaan. Mikäli järjestelmä toteuttaa eristyneisyysvaatimuksen, niin tapahtumaa suorittavan ohjelman ei tarvitse huolehtia mahdollisista toisista tapahtumista ja niiden vaikutuksesta käsiteltäviin resursseihin. Eristyneisyys saavutetaan esimerkiksi tietokannoista tuttujen lukitusmekanismien avulla. Tapahtuman vaikutukset näkyvät toisille tapahtumille vasta vahvistuksen jälkeen. [Ber1997] [Gra1993]
- **Durability** eli kestävyys tarkoittaa saavutettujen tulosten kestävyyttä tai säilyvyyttä. Mikäli tapahtuma suoritetaan onnistuneesti loppuun saakka, niin kestävyysvaatimuksen mukaan tulosten pitää säilyä vaikka järjestelmässä tapahtuisikin joku virhe. Käytännössä tämä tarkoittaa tulosten tallentamista johonkin pysyvään muistiin, esimerkiksi kovalevylle. Tällöin järjestelmän kaatuminen ja uudelleen käynnistäminen ei vaaranna aiemmin saavutettuja tuloksia. [Ber1997] [Gra1993]

2.2 Tapahtumien luokittelu

Tapahtumia voidaan luokitella monin eri tavoin. Toiminnallisella tasolla ne voidaan jakaa kahteen eri luokkaan. Aktiivisella tapahtumalla (online transaction) tarkoitetaan tapahtumaa, joka tekee määrättyjä toimenpiteitä aktiivisen käyttäjän (online user) puolesta. Vastaavasti passiivinen tapahtuma (offline transaction) on tapahtuma, jonka käsittely kestää niin kauan, ettei käyttäjän oleteta odottavan tapahtuman valmistumista. Tällaiset tapahtumat suoritetaan yleensä niin kutsuttuina eräajoina eli joukko kertyneitä tapahtumia suoritetaan jonakin määrättynä ajanhetkenä. [Ber1997]

Toiminnallisuuden lisäksi tapahtumia voidaan luokitella niiden rakenteen mukaan. Rakenteellisesti tapahtumat voidaan jakaa karkeasti kolmeen eri luokkaan. Luokkia voidaan helposti määritellä enemmänkin mutta käytännössä kaikki luokat ovat peräisin kolmesta pääluokasta tuoden mukanaan vain joitain pieniä eroavaisuuksia. Rakenteellisesti tapahtumat voivat siis olla joko yksitasoisia (flat), sisäkkäisiä (nested) tai ketjutettuja (chained).

2.2.1 Yksitasoinen tapahtuma

Yksitasoiset tapahtumat voivat sisältää mielivaltaisen määrän yksittäisiä operaatioita ja ne tarjoavat sovelluksille tavan yhdistää operaatioita loogisiksi kokonaisuuksiksi ACID-periaatteiden mukaan. Se suoritetaanko tapahtuman sisältämät operaatiot peräkkäin tai rinnakkain ei ole tärkeää vaan kaikki sisäinen toiminta voidaan piilottaa sovelluksen käyttämältä logiikalta. Kuva 1 on esitetty yksitasoinen tapahtuma, joka koostuu neljästä erillisestä operaatiosta.

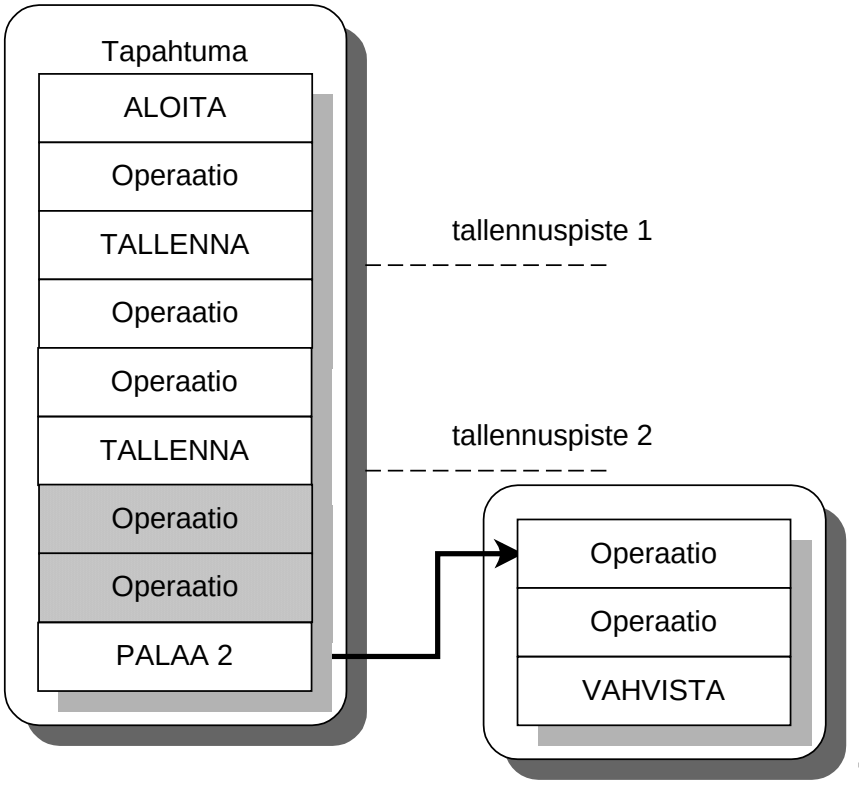
Kuva 1. Yksitasoinen tapahtuma.

Yksitasoinen tapahtuma saa nimensä siitä, että yhteen tapahtumaan sisältyy vain yksi kontrollitaso. Aloita, vahvista ja peruuta kontrollit ovat yhdellä ja samalla tasolla, joten tapahtuma joko vahvistetaan tai perutaan kokonaisuudessaan. Yksitasoinen tapahtuma ei salli tapahtumien osittaista vahvistamista.

Yksitasoiset tapahtumat ovat rakenteeltaan kaikkein yksinkertaisin ja helppokäyttöisin tapahtumaluokka. Sen ansiosta ne ovatkin kaikkein käytetyin luokka, etenkin jos asiaa tarkastellaan toteutuksien määrän avulla. Muut luokat perustuvat yksitasoiseen tapahtumaluokkaan tuoden siihen laajennuksia, joiden avulla voidaan mallintaa monimutkaisempia reaalimaailman ongelmia.

Yksitasoisessa tapahtumaluokassa on kuitenkin myös merkittäviä rajoituksia. Mikäli yksikin operaatio epäonnistuu, on sovelluksen joko palattava taaksepäin korjatakseen epäonnistunut operaatio tai peruttava koko tapahtuma, jolloin kaikki tapahtumassa tehty työ menee hukkaan. Ongelma ei ole merkittävä lyhyiden tapahtumien tapauksessa mutta se korostuu sitä mukaa, kun tapahtumat monimutkaistuvat ja niiden sisältämien operaatioiden määrä kasvaa. Yksi epäonnistunut operaatio ei useinkaan tarkoita sitä, että tapahtumassa aiemmin tehty työ olisi ollut turhaa. [Gra1993]

Yksi tapa ongelman ratkaisemiseen on käyttää jotain kehittyneenpää tapahtumaluokkaa, mutta monesti niitä ei ole joko käytettävissä, tai niiden tuoma etu ei ole riittävän hyvä niiden tuomaan kompleksisuuteen verrattuna. Useissa tapauksissa parempi vaihtoehto olisi, jos tapahtumaohjelma voisi palata aiempaan vaiheeseen saman tapahtuman sisällä. Tämän ongelman ratkaisemiseksi yksitasoisiin tapahtumiin on lisätty tallennuspisteitä, joiden tarkoituksena on toimia eräänlaisina virstanpylväinä joihin voi helposti palata mikäli siihen ilmenee tarvetta. Tallennuspisteet määrätään erillisellä operaatiolla. Operaatio palauttaa tapahtumaohjelmalle tunnisteen, jonka avulla se voi palata johonkin aiemmin tallennettuun tilaan. Kuva 2 esittää yksitasoisen tapahtuman, jossa on käytetty tallennuspisteitä. Tapahtuma on kesken suorituksen palannut tallennuspisteen 2 osoittamaan kohtaan ja jatkanut siitä uudelleen eteenpäin. [Gra1993]



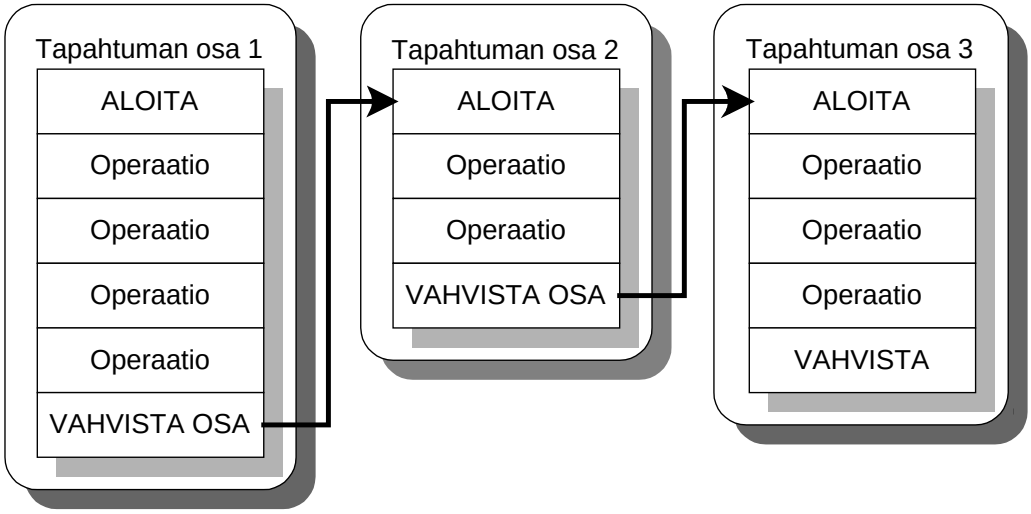
Kuva 2. Yksitasoinen tapahtuma ja tallennuspisteet.

Tallennuspisteitä käytettäessä atomäärisellä operaatiolla tarkoitetaan pikemminkin kahden tallennuspisteen välistä osaa kuin koko tapahtumaa. Osat ovat kuitenkin

rakenteellisesti riippuvaisia edeltäjistään ja koska niiden suoritusjärjestys on määrätty, ainoastaan viimeinen osa voi vahvistaa koko tapahtuman. Tallennuspisteet voivat olla myös pysyviä eli tapahtuman tila tallennetaan pysyvään muistiin kuten kiintolevylle. Tällaisten tallennuspisteiden etu on niiden hyvä virheiden sieto, mutta käytännön toteutus on usein vaikeaa. Tästä syystä pysyvät tallennuspisteet ovat melko harvinaisia. [Gra1993]

2.2.2 Ketjutettu tapahtuma

Ketjutettu tapahtumaluokka muistuttaa toiminnaltaan tallennuspisteiden käyttöä, korvaten tallennuspisteet tapahtuman sisällä tehtävillä vahvistuksilla. Toisin sanoen tapahtumaohjelma vahvistaa muutokset, jotka tapahtumassa on siihen mennessä tehty mutta pysyy silti saman tapahtuman sisällä ja ennen kaikkea säilyttää mahdolliset tietokantakursorit tai muut resurssikohtaiset tunnisteet itsellään. On tärkeää, että tapahtumaohjelma ei hukkaa aiemmin käsiteltyjä resursseja tällaisen kesken tapahtuman tehdyn vahvistuksen tapahtuessa, sillä yleensä tapahtuma jatkaa ainakin osittain samojen resurssien parissa työskentelyä. Kesken tapahtuman tehtävää vahvistusta varten on olemassa oma vahvistusoperaatio, joka kertoo järjestelmälle, että samaa tapahtumaa jatketaan heti vahvistuksen jälkeen. Tämän ansiosta järjestelmä osaa pitää tarvittavat resurssit varattuina kyseiselle tapahtumalle, eikä päästä toisia tapahtumia muuttamaan dataa. Tämä olisi vaarana, mikäli käytettäisiin erikseen vahvista ja aloita operaatioita. Kuva 3 esittää ketjutetun tapahtuman, joka koostuu kolmesta erillisestä osasta. [Gra1993]



Kuva 3. Ketjutettu tapahtuma.

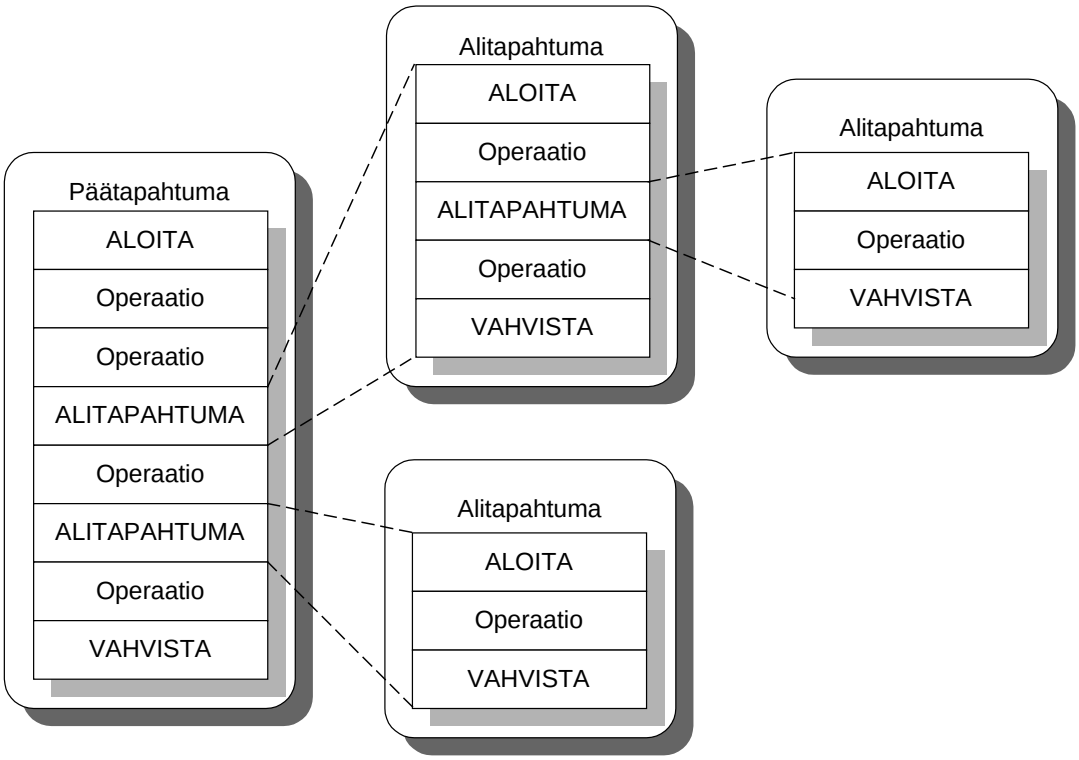
Ketjutettujen tapahtumien haittapuoli on ilmeinen. Samalla kun yksi osa tapahtumaa vahvistetaan, tapahtumaohjelma luopuu oikeudestaan peruuttaa kyseinen osa tapahtumaa. Peruutus voidaan tehdä vain sillä hetkellä aktiivisena olevalle tapahtuman osalle. Ei ole olemassa yksiselitteistä vastausta kysymykseen: Mitä tapahtumaohjelman pitäisi tehdä, jos yksi osa ketjutettua tapahtumaa epäonnistuu? Ketjutetut tapahtumat ovatkin tavallaan täysin erillisiä tapahtumia, jotka jakavat yhteisiä resursseja. Tästä syystä ketjun osat tulisi suunnitella siten, että niiden välillä olisi mahdollisimman vähän riippuvuuksia. Tällöin yhden osan epäonnistuminen ja peruuttaminen ei vaikuta niin paljon muiden osien tulokseen. [Gra1993]

Ketjutetuilla tapahtumilla on myös omat vahvuutensa. Koska tapahtuman sisällä tehdään vahvistuksia, tapahtuma voi näiden vahvistusten ansiosta vapauttaa lukot, joita se ei enää tarvitse. Tämä voi olla erittäin tärkeä ominaisuus järjestelmissä, joissa suorituskyvyllä on suuri painoarvo. Ketjutetut tapahtumat eivät myöskään ole niin herkkiä järjestelmän vikaantumiselle kuin ei-pysyviä tallennuspisteitä käyttävät yksitasoiset tapahtumat, sillä järjestelmän vikaantuminen tai kaatuminen aiheuttaa yleensä automaattisen peruutuksen kaikille käynnissä oleville tapahtumille. Tavallisista tallennuspisteistä ei ole hyötyä tällaisessa tilanteessa, mutta oikein suunniteltu ja toteutettu ketjutettu tapahtuma voi jatkaa työtään edellisestä vahvistuksesta alkaen. [Gra1993]

2.2.3 Sisäkkäinen tapahtuma

Myös sisäkkäiset tapahtumat ovat sukua yksitasoisille tapahtumille ja tallennuspisteille, mutta niiden tapa esittää usea samaan tapahtumaan kuuluva osa eroaa toisistaan. Sisäkkäiset tapahtumat muodostavat keskenään hierarkkisen rakenteen, jossa operaatiot voidaan kohdistaa joko koko tapahtumalle tai vaihtelevalle joukolle alitapahtumia.

Ylimmällä tasolla on päätapahtuma, sisältäen kaiken mikä tapahtumaan kuuluu. Sen alle on järjestetty mielivaltainen määrä alitapahtumia, alitapahtumien alitapahtumia ja niin edelleen. Alitapahtumat voivat siis olla joko yksitasoisia tai sisäkkäisiä. Yhdessä ne muodostavat puuta muistuttavan hierarkian, joka käsittää kaikki tapahtumaan kuuluvat osat. Alitapahtumien muodostama puumainen hierarkia on esitetty kuvassa 4. Alitapahtumia käsitellään omina yksilöinään, eli ne voidaan perua tai vahvistaa. Ne ovat kuitenkin riippuvaisia itseään ylemmän tason tapahtumasta, eikä niiden vahvistaminen tule voimaan ennekuin ylempi taso vahvistetaan. Tästä johtuen koko tapahtuma vahvistetaan vasta sitten, kun päätapahtuma vahvistetaan. Toisaalta tapahtuma näkee omien alitapahtumiensa tulokset vasta, kun ne vahvistavat muutoksensa. Tästä johtuen tapahtuma vahvistetaan vasta, kun kaikki sen alitapahtumat on vahvistettu. Vahvistusjärjestys menee siis puussa alhaalta ylöspäin ja lopulta päätapahtuman vahvistaminen ottaa voimaan kaikki tapahtuman sisällä tehdyt toimenpiteet. Alitapahtumat toteuttavat kolme ensimmäistä neljästä ACID-vaatimuksesta. Viimeisen toteuttaminen jää päätapahtuman vastuulle. Tämä johtuu siitä, että minkä tahansa tapahtuman peruminen peruu samalla kaikki sen alitapahtumat. [Gra1993]



Kuva 4. Sisäkkäisen tapahtuman muodostama hierarkia.

Sisäkkäiset tapahtumat ovat tehokas tapa hallita sovelluksia, joiden rakenne on suuri ja monimutkainen. Tietojärjestelmien pilkkomisella pienenpiin osiin (modularization) ja sisäkkäisillä tapahtumilla on keskenään vahva suhde. Hyvin suunnitellut ja oikein toteutetut ohjelmistomoduulit käsittelevät dataa ainoastaan oman rajapintansa kautta. Mikäli globaaleja muistialueita ei käytetä, moduulin suorituksen epäonnistuminen ei voi rikkoa ylemmän tason tietorakenteita ja moduulien suorituksen valvominen on helppoa. Käytännön sovelluksissa on kuitenkin usein mukana joku resurssi, jota voidaan pitää globaalina muuttujana. Usein tämä resurssi on tietokanta, jota useampi kuin yksi ohjelmistomoduuli voi käsitellä. Tällöin moduulien aiheuttamien seurausten valvominen käy lähes mahdottomaksi ilman tapahtumankäsittelyä ja sisäkkäisiä tapahtumia. [Gra1993]

Monitasoiset (multi-level) tapahtumat ovat sisäkkäisten tapahtumien vapaampi muoto. Monitasoiset tapahtumat sallivat alitapahtumien vahvistamisen ennen ylemmän tason vahvistamista mutta samalla ne olettavat, että järjestelmässä on olemassa alitapahtumaa vastaava kompensoiva tapahtuma. Kompensoivan tapahtuman avulla alitapahtuman

tulokset voidaan perua, mikäli niitä ylempi tapahtuma perutaan. Monitasoisilla tapahtumilla voidaan saavuttaa huomattavasti parempi suorituskyky tietyissä tilanteissa, sillä alitapahtumien vahvistaminen vapauttaa niiden käsittelemät resurssit toisten tapahtumien käyttöön. Tämä asettaa kuitenkin myös vaatimuksia, sillä entä jos tapahtuma T2 tuhoaakin tapahtuman T1 alitapahtuman T1.1 tekemät muutokset ennen kuin tapahtuma T1 vahvistetaan? Tällainen tilanne rikkoisi selvästi tapahtuman T1 atomäärisyysvaatimusta, eikä sitä voi täten sallia. Monitasoisia tapahtumia käytettäessä tuleekin esille vaatimus käsiteltävän datan rakenteesta. Tavalliset sisäkkäiset tapahtumat eivät aseta mitään vaatimuksia alitapahtumien toiminnalle ja käsiteltävälle datalle, sillä kaikki tapahtuman käsittelemä data on lukittu kunnes päätapahtuma vahvistetaan. Monitasoisten tapahtumien käyttö taas vaatii, että alitapahtumat käsittelevät vain ja ainoastaan tiettyä osaa resursseista.

2.3 Rinnakkaiset tapahtumat ja yhtäaikaisuudenhallinta

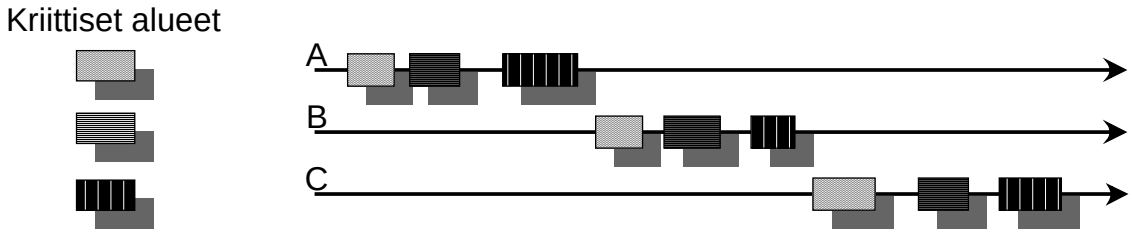
Eristyneisyys vaatimuksen mukaan tapahtumia tulee voida suorittaa rinnakkain, ilman että tapahtumat häiritsevät toisiaan tai että tapahtuman tulee omassa logiikassaan huomioida rinnakkainen suoritus. Tämä vaatimus kohdistuu siis tapahtumankäsittelystä vastaavalla järjestelmälle. Järjestelmän tulee suorittaa tapahtumia siten, että ainoastaan yksi tapahtuma pääsee kerrallaan järjestelmässä oleville kriittisille alueille (critical region). Kriittiseksi alueeksi kutsutaan resurssia, jota ainoastaan yksi tapahtuma voi käsitellä yhtäaikaisesti. [Bac1992]

2.3.1 Kriittiset alueet ja sarjoitettavuus

Monissa käytännön sovelluksissa tapahtumat käsittelevät toisistaan riippumatonta dataa eli järjestelmässä ei ole kriittisiä alueita. Tällöin tapahtumia voidaan suorittaa rinnakkain ilman ongelmia tai erityistoimenpiteitä, esimerkiksi henkilötietoja tai pankkitilejä käsittelevät tietojärjestelmät voivat ovat tällaisia. Jos jokainen järjestelmässä käytettävä tapahtuma käsittelee eri dataa, niitä kaikkia voidaan suorittaa

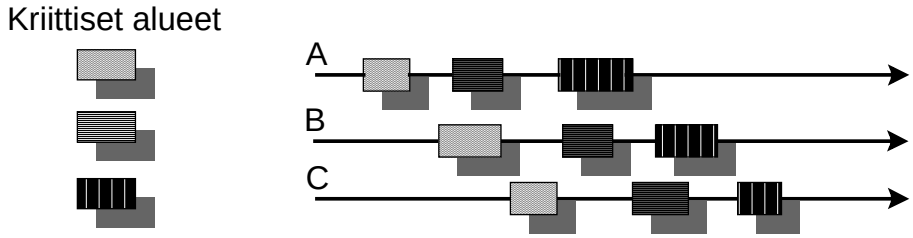
yhtäaikaisesti ilman että ne vaikuttavat toisiinsa. Ongelmia syntyy, mikäli useampi kuin yksi tapahtuma pyrkii käsittelemään samaa kriittistä aluetta.

Tekniikka, jolla varmistetaan, että tapahtumat eivät pääse yhtäaikaisesti kriittisille aluille, on nimeltään sarjoittaminen (serialization). Sarjoittaminen tarkoittaa tapahtumien suoritusta, jolla on sama vaikutus, kuin tapahtumien peräkkäisellä suorittamisella (serial execution). Yksinkertaisin sarjoitettu malli onkin sellainen, jossa tapahtumat suoritetaan yksi kerrallaan. Kuva 5 esittää tilanteen, jossa kolme tapahtumaa A, B ja C suoritetaan peräkkäin yksi kerrallaan. [Bac1992]



Kuva 5. Tapahtumien peräkkäinen suorittaminen. [Bac1992]

Edellä mainittu tapa tulee kuitenkin harvoin kysymykseen, sillä järjestelmän suorituskyky kärsii tällaisesta ratkaisusta kohtuuttomasti. Ratkaisu ei myöskään skaalaudu prosessorien määrän funktiona, sillä minä tahansa ajanhetkellä järjestelmässä suoritetaan vain yhtä tapahtumaa. Tästä johtuen ratkaisu on huono etenkin hajautetuissa järjestelmissä, koska kaikki muut järjestelmän osat odottaisivat yhdessä osassa käynnissä olevan tapahtuman suoritusta. Tämän takia on järkevää limittää tapahtuman sisältämät operaatiot siten, että yhtä kriittistä aluetta pääsee käsittelemään kerrallaan vain yksi operaatio. Operaatioita, jotka eivät käsittele samaa kriittistä aluetta, voidaan suorittaa rinnakkain. Kyseessä on edelleen sarjoitettu tapahtumien suoritus, sillä ristiriitaiset operaatiot suoritetaan toisiinsa nähden yksi kerrallaan. Kuva 6 esittää, kuinka tapahtumia voidaan suorittaa sarjoitetusti limittämällä niiden operaatiot keskenään. Limitetystä mallista käytetään yleisesti termiä liukuhihna (pipeline).



Kuva 6. Tapahtumien limitetty suorittaminen.

2.3.2 Resurssien lukitseminen

Yleisin tapa sarjoittamisen toteuttamiseen on resurssien varaaminen eli lukitseminen. Lukko voi olla tyypiltään joko luku- tai kirjoituslukko ja niiden toimintaperiaate on hyvin yksinkertainen. Tapahtuma tai sen operaatio lukitsee aina resurssin, jota se käsittelee käyttäen tarkoituksesta riippuen joko luku- tai kirjoituslukkoa. Lukulukkoa ei voida asettaa, mikäli toisella tapahtumalla tai operaatiolla on kirjoituslukko kyseiseen resurssiin. Kirjoituslukon tapauksessa lukittavassa resurssissa ei saa ennestään olla kummankaan tyyppistä lukkoa. [Bac1992]

Lukitseminen vaikuttaa negatiivisesti järjestelmän suorituskykyyn mutta on toisaalta välttämätön mekanismi tapahtumien sarjoittamiselle ja siten eristyneisyys vaatimuksen toteuttamiselle. Pelkkä resurssien lukitseminen ei kuitenkaan välttämättä takaa tapahtumien oikeaoppista sarjoittamista. Myös lukkojen vapauttamisen ajoituksella on tärkeä rooli sarjoittamisen tavoittelussa. Esimerkiksi jos kaksi tapahtumaa T1 ja T2 käsittelevät kahta resurssia X ja Y seuraavasti:

1. T1 lukee resurssia X.
2. T2 lukee resurssia Y.
3. T1 kirjoittaa resurssia Y.
4. T2 kirjoittaa resurssia X.

Merkitään nyt lukuoperaatiota Ln:llä ja kirjoitusoperaatiota Kn:lla, joissa n on tapahtuman numero. Tällöin operaatiot suoritetaan järjestyksessä L1(X) L2(Y) K1(Y)

K2(X). Tapahtumat eivät ole sarjoitettavissa, sillä ainoat sarjoitettavat tapahtumaketjut olisivat T1T2 tai T2T1, eli L1(X) K1(Y) L2(Y) K2(X) tai L2(Y) K2(X) L1(X) K1(Y). Ongelma aiheutuu siitä, että tapahtuma T1 vapauttaa lukkonsa resurssissa X ennen resurssin Y lukitsemista. Ongelman ratkaisuun on olemassa menetelmä nimeltä kaksivaihelukitus (two-phase locking). Sen mukaan tapahtuman pitää lukita kaikki käsittelemänsä resurssit ennen kuin se voi vapauttaa yhtään niistä. [Ber1997] Ensimmäinen vaihe koostuu siis lukkojen saamisesta ja toinen niiden vapauttamisesta.

Kaksivaihelukitus auttaa tapahtumien sarjoittamisessa, mutta se aiheuttaa myös ongelman nimeltä ikilukot (deadlock). Ikilukkotilanteessa kaksi tai useampaa tapahtumaa ajautuu tilaan, jossa ne eivät keskinäisistä lukituksista johtuen voi edetä vaan kaikki jäävät odottamaan toistensa lukitsemien resurssien vapautumista. Äsken käsitellyt tapahtumat T1 ja T2 ajautuvat ikilukkoon, sillä T1:llä on lukulukkoa X:ään ja T2:lla Y:hyn samalla kun T1 pyrkii saamaan kirjoituslukkoa Y:hyn ja T2 vastaavasti X:ään. Tästä johtuen molemmat odottavat, että toinen tapahtuma vapauttaisi niiden tarvitseman lukon. Kumpikaan tapahtumista ei voi kuitenkaan kaksivaihelukituksen säännön perusteella vapauttaa vain yhtä lukkoaan. Tästä syystä ainoa mahdollisuus ikilukkotilanteesta selviämiseen on yhden tai useamman ikilukkoon osallistuneen tapahtuman peruminen, jolloin kaikki niiden hallussaan pitämät lukot vapautetaan. [Bac1992]

Ikilukkojen ennaltaehkäisyyn on olemassa menetelmiä, jotka perustuvat siihen, että tapahtumalle ei anneta lukkoa joka johtaisi myöhemmin ikilukon syntymiseen. Tapahtumankäsittelyn yhteydessä tämä kuitenkin rajoittaisi rinnakkaisuutta merkittävästi, ja tästä syystä sillä olisi erittäin huono vaikutus järjestelmän suorituskykyyn. Tästä johtuen käytännössä kaikki tapahtumankäsittelyjärjestelmät sallivat ikilukkojen tapahtumisen. Ikilukot voidaan kuitenkin tunnistaa niiden tapahtuessa, jolloin järjestelmä voi automaattisesti perua tarvittavat tapahtumat. Ikilukkojen tunnistaminen voidaan tehdä joko tapahtumien suoritusaikaan tai odotusgraafeihin (wait-for graph) perustuen. [Bac1992] [Ber1997]

Suoritusaikaan perustuva ikilukkotunnistus päättelee tapahtuman joutuneen ikilukkoon, mikäli sen suoritus kestää pidempään, kuin suurin sallittu suoritusaika. Sen etuina on yksinkertainen toteutus, joka toimii helposti myös hajautetuissa ympäristöissä. Toisaalta on olemassa mahdollisuus, että järjestelmä peruu tapahtumia, joiden suoritus vain kestää normaalia kauemmin ja jotka eivät ole oikeasti ikilukossa. Ongelmaa voidaan lieventää kasvattamalla tapahtumien suurinta sallittua suoritusaikaa, mutta se johtaa toiseen ongelmaan. Mikäli tapahtuma ajautuukin ikilukkoon heti suorituksensa alussa, huomaa järjestelmä sen vasta pitkän odotusajan kuluttua. [Ber1997]

Odotusgraafeihin perustuva menetelmä pitää yllä suunnattua graafia tapahtumien välisistä odotussuhteista. Graafin solmut ovat tapahtumia ja kaaret kertovat onko tapahtuma pysähtynyt odottamaan toisen tapahtuman hallussa olevaa lukkoa. Mikäli graafiin muodostuu silmukka, niin silmukkaan kuuluvat tapahtumat ovat ajautuneet ikilukkoon. [Ber1997]

2.3.3 Optimistinen yhtäaikaisuudenhallinta

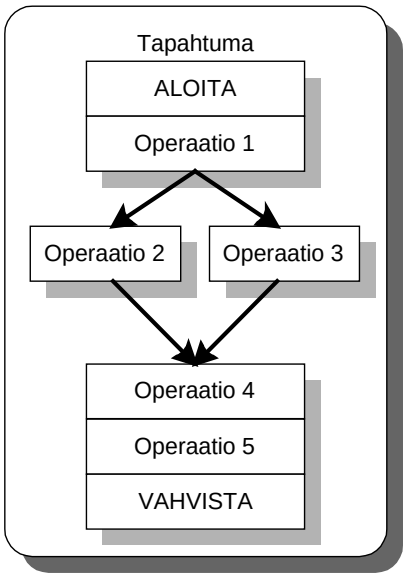
Optimistiset yhtäaikaisuudenhallintamenetelmät perustuvat siihen oletukseen, että tapahtumat eivät suurella todennäköisyydellä käsittele samoja kriittisiä alueita. Optimistisilla menetelmillä pyritään vähentämään reaaliaikaisten järjestelmien viiveitä ja parantamaan siten niiden suorituskykyä ja käytettävyyttä. [Bac1992]

Teknisesti optimistiset yhtäaikaisuudenhallintamenetelmät toimivat siten, että muutoksia ei tehdä varsinaiseen resurssiin tapahtuman suorituksen aikana, vaan tapahtumaa käsitellään välimuistissa. Kun tapahtuma vahvistetaan, sen muutokset tarkastetaan ja päätellään voidaanko tapahtuman suoritusta pitää sarjoitettavana toisten yhtä aikaa suoritettavana olleiden ja aiemmin vahvistettujen tapahtumien kanssa. Jos tapahtuman suorituksen todetaan olleen sarjoitettava tai jonkun muun eristyneisyysehdon täyttävä, muutokset otetaan käyttöön varsinaisessa resurssissa. [Bac1992]

Erona kaksivaihelukituksella saavutettuun sarjoitettavuuteen on se, että tapahtumat eivät jää lukkoon odottamaan resurssien vapautumista vaan suorittavat muutoksensa välittömästi. Mikäli joku toinen tapahtuma on kuitenkin ehtinyt muuttaa samoja resursseja, tapahtumalle palautetaan vahvistuksen yhteydessä virheilmoitus. Kaksivaihelukituksen yhteydessä ei vahvistuksen aikaisia virheitä voi tapahtua, mutta tapahtumat saattavat olla lukossa pitkiäkin aikoja tai pahimmassa tapauksessa jäädä ikilukkoon. Monet tietokantapalvelimet käyttävätkin oletuksena optimisista yhtäaikaisuudenhallintamenetelmää eristyneisyyden varmistamiseen.

2.3.4 Riippuvuudet ja suoritusjärjestyksen määrääminen

Yhden tapahtuman sisältämät operaatiot suoritetaan yksi kerrallaan määrätyssä järjestyksessä, ellei järjestetystä ole erikseen jätetty määrittämättä. Järjestystä ei kannata määrittää, mikäli operaatioilla ja niiden järjestyksellä ei ole mitään keskinäistä riippuvuutta. Tällöin tapahtumankäsittelyjärjestelmä voi suorittaa edellä mainitun kaltaiset operaatiot rinnakkain ilman pelkoa siitä, että niiden ja siten koko tapahtuman tulos vaarantuu. Rinnakkaisella suorituksella saavutetaan tehokkuushyötyä varsinkin hajautetuissa-, rinnakkaisissa- tai moniprosessoriympäristöissä. Kuva 7 esittää tapahtuman, jossa operaatioiden 2 ja 3 suoritusjärjestystä ei ole määrätty. Tällöin tapahtumankäsittelyjärjestelmä voi suorittaa ne haluamassaan järjestyksessä tai vaikka yhtäaikaisesti. Molemmat on kuitenkin suoritettava operaation 1 jälkeen ja ennen operaatiota 4. [Bac1992]

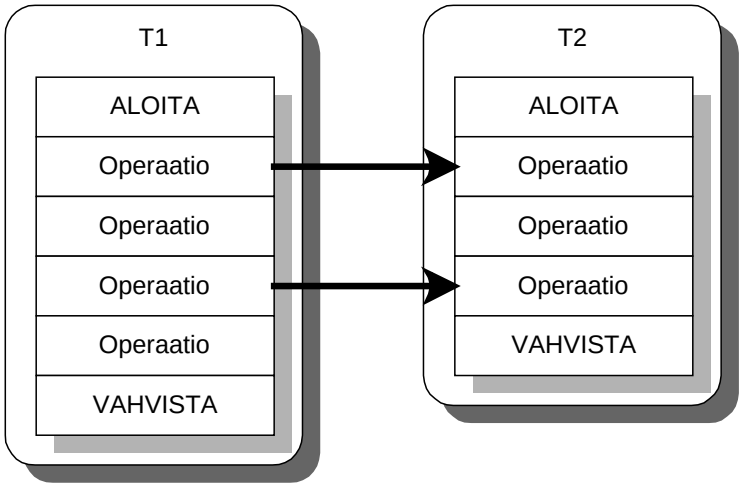


Kuva 7. Tapahtuman sisäiset rinnakkaiset operaatiot.

Useamman tapahtuman tapauksessa tilanne ei ole aivan niin yksinkertainen. Aina ei riitä että varmistetaan operaatioiden sarjoittaminen kriittisten alueiden suhteen, sillä tapahtumat voivat sisältää teknisten riippuvuuksien lisäksi myös loogisia riippuvuuksia. Käytetään esimerkkinä tilannetta, jossa tapahtuma T lukee dataa resurssista R. T1 saadaan kuitenkin suorittaa vasta kun tapahtuma T2 on käynyt kirjoittamassa luettavan datan resurssiin R. Jos tapahtuman T1 dataa lukeva operaatio suoritetaankin ennen tapahtuman T2 dataa kirjoittavaa operaatiota, molemmat tapahtumat kyllä palauttavat onnistuneen lopputuloksen, mutta lopputulos ei ole oikea. Mikäli tapahtumankäsittelyjärjestelmä ei tarjoa keinoja tapahtumien välisten riippuvuuksien kuvaamiseen, ei tapahtumia T1 ja T2 saa antaa käsiteltäväksi yhtäaikaisesti. Tapahtuma T1 voidaan suorittaa vasta kun tapahtuma T2 on suoritettu loppuun saakka. Tapahtumien suoritusjärjestyksen määrää järjestelmässä oleva ajastuksesta huolehtiva komponentti (scheduler). Sen vastuulla on tapahtumien, ja niiden sisältämien operaatioiden, suorituksen jakaminen toisille komponenteille. [Bac1992]

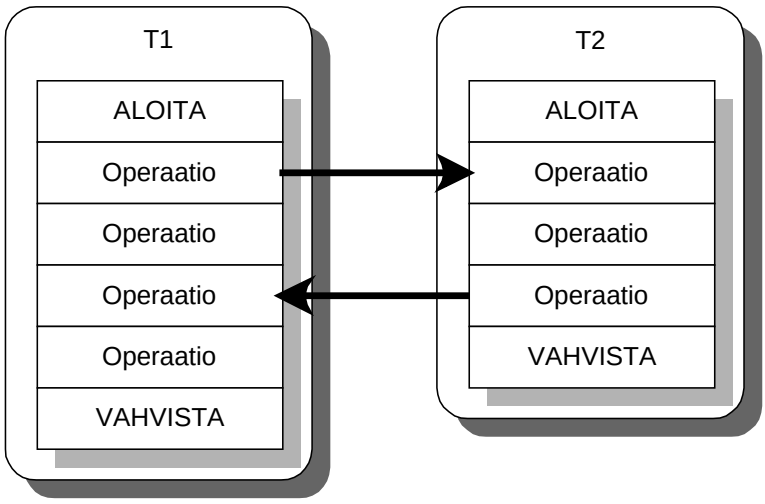
Kahden eri tapahtuman välillä voi olla riippuvuuksia joko toisesta toiseen tai molemmista molempiin. Tapahtumat, joiden operaatioiden välillä on kuvan 8 esittämä suhde, ovat sarjoitettavia. Tällaiset tapahtumat voidaan suorittaa peräkkäin kuvan 5 osoittamalla tavalla. Tapahtuma T1 sisältää operaatioita, jotka täytyy suorittaa ennen

tapahtuman T2 määrättyjä operaatioita eli tapahtuma T1 voitaisiin suorittaa kokonaisuudessaan ennen T2:n suoritusta.



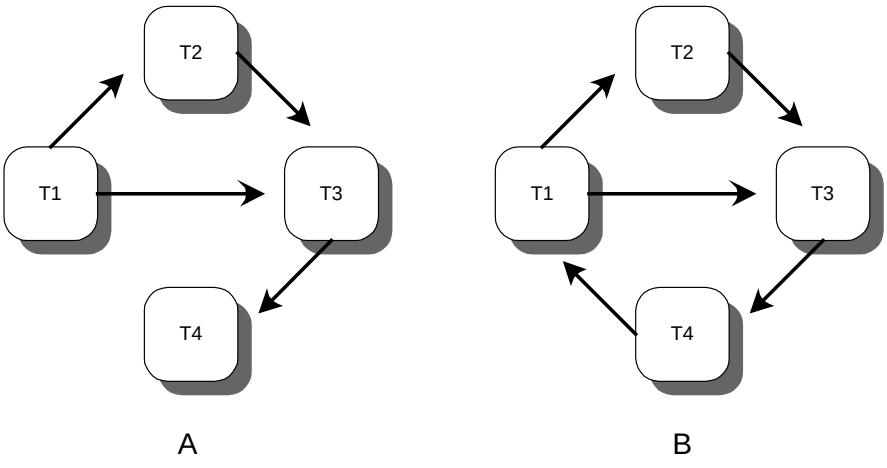
Kuva 8. Sarjoitettava tapahtumien suoritus.

Mikäli tapahtuman T1 operaatioilla on samanaikaisesti vastaavanlaisia riippuvuuksia tapahtuman T2 operaatioihin, on tilanne erilainen. Tällaisia tapahtumia ei voi suorittaa peräkkäin, sillä molempien tapahtumien suorittaminen vaatii osia toisen tapahtuman suorittamisesta. Tästä syystä tapahtumat eivät myöskään ole sarjoitettavia. Kuva 9 esittää tapahtumat T1 ja T2, joiden välillä riippuvuuksia molempiin suuntiin. Kumpaakaan ei voida suorittaa yksinään ilman toisen tapahtuman rinnakkaista suorittamista.



Kuva 9. Ei-sarjoitettava tapahtumien suoritus.

Tapahtumien ja niiden operaatioiden riippuvuuksia voidaan kuvata sarjoituskaavion (serialization graph) avulla. Sarjoituskaaviossa solmut ovat tapahtumia ja suunnatut kaaret solmujen välissä osoittavat riippuvuutta. Kuva 10 esittää kaksi sarjoituskaaviota. Kaavion A tapahtumien suoritus voidaan sarjoittaa, sillä sarjoituskaavioon ei muodostu silmukoita. Tapahtumat voitaisiin siis suorittaa yksi kerrallaan järjestyksessä T1, T2, T3, T4. Kaavion B tapahtumien suoritusta sen sijaan ei voida sarjoittaa, sillä niiden välille ei voida määrittää yksikäsitteistä suoritusjärjestystä. [Bac1992]



Kuva 10. Sarjoituskaavio. [Bac1992]

3 HAJAUTETTUJEN YMPÄRISTÖJEN TAPAHTUMANKÄSITTELY

Tähän mennessä tässä työssä on tarkasteltu lähinnä yksittäisten tapahtumien ominaisuuksia ja suorittamista, sekä tapahtumien rinnakkaisen suorittamisen tuomia ongelmia. Tässä kappaleessa laajennetaan katsontakantaa ja käsitellään tapahtumankäsittelyä hajautetuissa ympäristöissä.

Järjestelmien hajauttaminen useampaan osaan, eli alijärjestelmään, on monissa tapauksissa hyvä keino suorituskyvyn lisäämiseksi. Paremman suorituskyvyn hakeminen onkin yleisin syy järjestelmien hajauttamiseen. Käyttäjämäärien ja käsiteltävän tiedon kasvaessa yksittäiset palvelimet eivät enää pysty hoitamaan kaikkea niihin kohdistuvaa kuormaa. Tällöin järjestelmä voidaan jakaa osittain itsenäisiin alijärjestelmiin, jotka käsittelevät tietyn osan koko järjestelmään kohdistuvasta työstä. Työn jakaminen voidaan tehdä monien eri kriteerien perusteella. Käyttäjät voidaan esimerkiksi jakaa järjestelmässä siten, että yksi alijärjestelmä vastaa tietystä osasta käyttäjiä ja hoitaa siten suurimman osan käyttäjän antamista tehtävistä. Yksi alijärjestelmä ei kuitenkaan voi hoitaa kaikkea työtä, vaan se joutuu joissain tapauksissa kommunikoimaan myös toisten alijärjestelmien kanssa. Mikäli tätä kommunikaatiota ei tarvittaisi, voitaisiin pikemminkin puhua täysin erillisistä järjestelmistä kuin hajautetusta järjestelmästä.

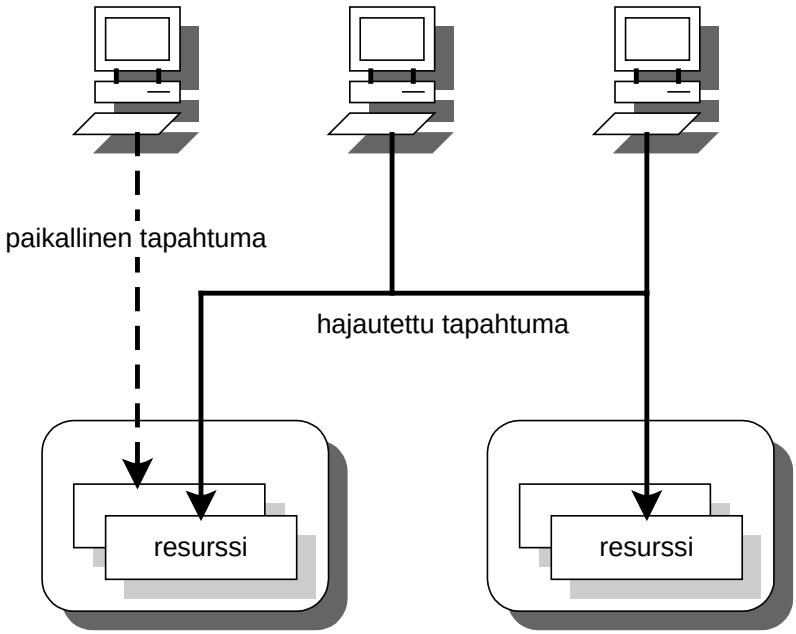
Suorituskyvyn lisäksi hajauttamisella voidaan saavuttaa muitakin etuja. Syitä järjestelmien pilkkomiseen ja hajauttamiseen voivat olla muun muassa luotettavuus- tai tietoturvatekijät. Hajauttamisen avulla järjestelmä voidaan pilkkoa loogisesti sopiviin alijärjestelmiin esimerkiksi maantieteellisen sijainnin tai asiakkaiden suhteen. Tällöin tietyn asiakkaan resurssit voidaan pitää yhdessä rajatussa paikassa ja niiden käyttöä hajautetun järjestelmän muista osista voidaan säännöstellä. Hajauttamisella voidaan lisäksi monissa tapauksissa parantaa järjestelmän luotettavuutta, sillä yhden alijärjestelmän vikaantuminen ei useinkaan tarkoita koko järjestelmän kaatumista.

Hyötyjen vastapainoksi hajauttaminen tuo myös paljon uusia vaatimuksia ja ongelmia järjestelmiä hyödyntäville sovelluksille ja niissä suoritettavalle tapahtumankäsittelylle. Mahdollisten ongelmien ratkaiseminen on kuitenkin ensiarvoisen tärkeää, sillä tapahtumankäsittelyn merkitys korostuu entisestään hajautetuissa ympäristöissä. Perinteinen, paikallisten aliohjelmakutsujen tulosten perusteella toimiva malli ei riitä takamaan järjestelmänlaajuisten toimenpiteiden onnistumista. Tietokantojen ja tapahtumien kanssa pitkään työskennellyt tekniikan tohtori Jim Gray onkin todennut seuraavasti: ”*Ajatus hajautetuista järjestelmistä ilman tapahtumanhallintaa on kuin yhteisö ilman lakeja. Lakeja ei välttämättä tarvita, mutta erimielisyyksien sattuessa tarvitaan menetelmä ongelmien selvittelyyn*”[1]. [Orf1996] Voidaan sanoa, että hajautettujen järjestelmien tapauksessa tapahtumat eivät ole enää pelkästään liiketoimintasovellusten työkalu. Ne ovat pikemminkin suunnittelutapa ja niiden avulla voidaan taata järjestelmien oikeaoppinen toiminta.

3.1 Hajautuksen aiheuttamat ongelmat

Hajautetussa järjestelmässä voidaan suorittaa paikallisia tapahtumia siten, ettei järjestelmän hajautus vaikuta tapahtumankäsittelyyn millään tavalla. Tällöin puhutaan paikallisista tapahtumista (local transaction). Paikallinen tapahtuma on yhden osapuolen ohjaama tapahtuma, joka käsittelee ainoastaan yhtä järjestelmän resurssia. Onkin tärkeää erottaa paikalliset tapahtumat hajautetuista tapahtumista (global transaction, distributed transaction), joihin voi osallistua monta sovellusta ja jotka yleensä käsittelevät useaa järjestelmän resurssia. [Bjö1991] [Ber1997] Kuva 11 esittää hajautetussa järjestelmässä suoritettavien paikallisen ja hajautetun tapahtuman eron.

[1] Kirjoittajan vapaa suomennos alkuperäisestä englanninkielisestä tekstistä.



Kuva 11. Paikallinen ja hajautettu tapahtuma.

Hajauttamisella voidaan saavuttaa järjestelmään luotettavuutta, sillä yhden alijärjestelmän vikaantuminen ei useinkaan kaada koko järjestelmää. Hajautettujen tapahtumien osalta alijärjestelmän tai alijärjestelmien välisen tiedonsiirtokanavan vikaantuminen voi kuitenkin olla vakava ongelma, sillä yhden alijärjestelmän tai tiedonsiirtokanavan vikaantuminen voi estää koko tapahtuman suorituksen. Tämän lisäksi alijärjestelmien välisen tiedonsiirtokanavan tai itse alijärjestelmien vikaantuminen kesken tapahtumien suorituksen voi aiheuttaa vakavia ongelmatilanteita. Tapahtumaa ohjaava ohjelma tai ohjelmat saavat kyllä järjestelmältä ilmoituksen tiedonsiirtokanavan vikaantumisesta, mutta ne eivät voi olla varmoja ehdittiinkö tapahtuman operaatiot suorittaa ennen vikaantumista vai ei. Tämän takia hajautetut tapahtumankäsittely-ympäristöt tarvitsevat myös tehokkaat mekanismit virheistä palautumiseen. Tapahtumalokeja (transaction log) käytetään yleisesti niin tapahtumien tekemien toimenpiteiden seuraamiseen kuin virheistä palautumiseen. [Gra1993]

Yhtäaikaisuuden hallinta ja ikilukkojen tunnistus vaativat hajautetuissa järjestelmissä joitakin erityistoimia. Kaksivaihelukituksen avulla saavutetaan sarjoitettavuus myös hajautetussa ympäristössä, mutta lukkojen vapauttamisen ajoitus on teknisesti vaikeampaa kuin yksittäisen resurssin tapauksessa. Resurssi saa vapauttaa lukkonsa

vasta, kun kaikki toiset tapahtumaan osallistuvat resurssit ovat saaneet kaikki tarvittavat lukkonsa. Lukituksen kontrollointiin voidaan käyttää erityistä protokollaa tai sitten lukot voidaan vapauttaa vasta vahvistuksen yhteydessä. Kaksivaihelukituksen mahdollisesti aiheuttamien ikilukkojen huomaaminen on myös hankalampaa hajautetussa ympäristössä. Odotusgraafeihin perustuva menetelmä on teknisesti vaikea ja raskas toteuttaa, sillä jokaisen tapahtumaan osallistuvan alijärjestelmän täytyy pitää yllä suurta määrää tietoa tapahtuman tilasta. Tästä syystä hajautetut järjestelmät suosivatkin yleensä suoritusaikaan perustuvaa ikilukkotunnistusta. [Bac1992]

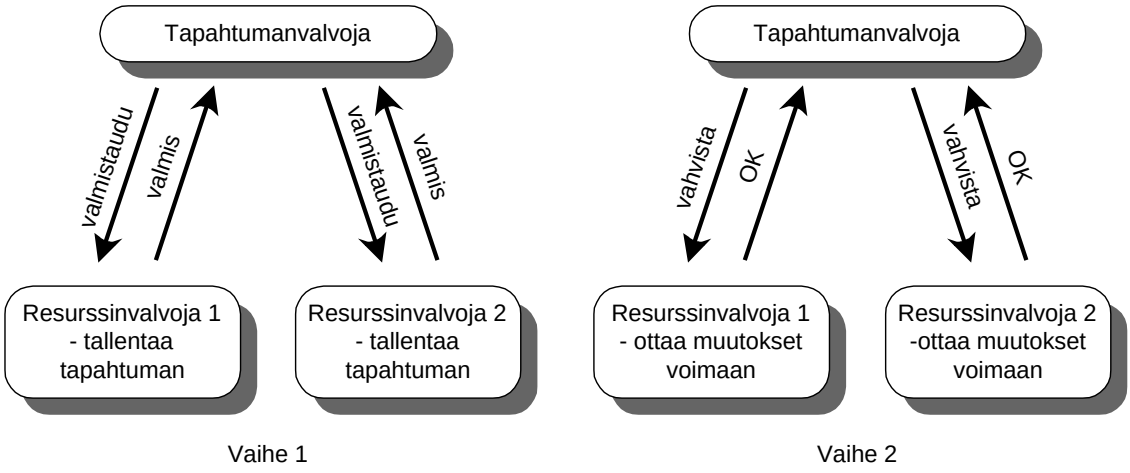
Hajautettua tapahtumaa, joka käsittelee tietoa useammassa kuin yhdessä hajautetun järjestelmän osassa, sitoo edelleen määrätyt ACID-vaatimukset. Tämä tarkoittaa käytännössä sitä, että joko kaikki tapahtumaan liittyvät alijärjestelmät suorittavat annetut tehtävät onnistuneesti tai sitten yhtään muutosta ei tehdä yhteenkään järjestelmän osaan. Tämä aiheuttaa ongelmia tapahtumankäsittelylle, sillä mikä tahansa tapahtumaan osallistuva hajautetun järjestelmän osa voi epäonnistua omassa tehtävässään. Varsinainen ongelma syntyy, jos osa alijärjestelmistä on jo ehtinyt vahvistaa oman osansa tapahtumasta, ennen kuin tapahtumaan kuuluva operaatio epäonnistuu jossain toisessa alijärjestelmässä. Tämän ongelman ratkaisuun on kehitetty kaksivaihevahvistumekanismi ja -protokolla (two-phase commit protocol, 2PC). [Ber1997] [Gra1993]

3.2 Kaksivaihevahvistus

Kaksivaihevahvistuksen (two-phase commit) ja kaksivaihevahvistusprotokollan tavoitteena on varmistaa luotettava tapahtumankäsittely hajautetussa ympäristössä, jossa tapahtuma käsittelee tietoa useissa eri alijärjestelmissä tai järjestelmäkomponenteissa. Kaksivaihevahvistuksen toimintaperiaate on yksinkertainen. Tapahtuman kulkua ohjaa tapahtumanvalvoja (transaction manager), joka on tehtävään erikoistunut järjestelmäkomponentti. Se pitää kirjaa käynnissä olevista tapahtumista, niiden tilasta ja resurssinvalvojista (resource manager), joita tapahtuman suorittaminen koskee. Resurssinvalvoja on komponentti, jonka kautta sen valvomaan resurssiin pääsee käsiksi.

Resurssi voi olla esimerkiksi tietokanta, jono, tiedosto, viesti tai joku muu mahdollisesti käsiteltävä objekti. Tärkeintä on, että resurssinvalvojan avulla käsiteltävä resurssi on kyvykäs osallistumaan tapahtumaan. Käytännössä tämä tarkoittaa sitä, että resurssi toteuttaa itsenäisesti ACID-periaatteet. [Ber1997]

Kaikki tapahtumat rekisteröidään aluksi tapahtumanvalvojalle, joka antaa niille ainutkertaisen tapahtumatunnuksen (transaction identifier). Jonkun tapahtumaan kuuluvan osapuolen pitää lisäksi ilmoittaa tapahtumanvalvojalle aina, kun tapahtuman käsittelyyn liittyy uusi resurssinvalvoja. Tapahtumanvalvojan täytyy tietää kaikki tapahtumaan osallistuvat resurssinvalvojat, sillä kun tapahtumankäsittelysovellus tahtoo operaatioiden suorittamisen jälkeen vahvistaa tapahtuman, niin tapahtumanvalvoja lähettää tiedon tästä kaikille tapahtumaan osallistuville resurssinvalvojille. Tämä on kaksivaihevahvistuksen ensimmäinen vaihe ja tässä vaiheessa resurssinvalvojien tulee tallentaa niitä koskeva osa tapahtumasta johonkin pysyvään muistiin, esimerkiksi kovalevylle. Kun tapahtuma on tallennettu resurssinvalvoja vastaa tapahtumanvalvojalle, että se on valmis vahvistamaan oman osansa tapahtumasta. Tapahtumanvalvoja odottaa että kaikki resurssinvalvojat ovat valmiita ja käskee niitä sitten vahvistamaan tapahtuman. Tämä on kaksivaihevahvistuksen toinen vaihe. Kuva 12 esittää kaksivaihevahvistuksen vaiheet ja niihin liittyvät viestit ja toimenpiteet.



Kuva 12. Kaksivaihevahvistusprotokollan toiminta [Ber1997].

Pysyvään muistiin tallentamisen tarkoitus on se, että vaikka joku resurssinvalvoja vikaantuisi tai kaatuisi ennen kuin se ehtii vahvistamaan tapahtuman, niin tapahtuman tiedot eivät pääse katoamaan. Resurssinvalvojan tulee vahvistaa tapahtuma kun se toipuu virheestä ja pääsee taas aktiiviseksi. Peruutuksen tapauksessa asia hoituu samankaltaisesti, tosin hieman yksinkertaisemmin. Tapahtumanvalvoja lähettää tiedon peruutuksesta kaikille tapahtumaan osallistuville resurssinvalvojille ja ne peruvat kaikki omat muutoksensa.

Kaksivaihevahvistus ja siihen kuuluva koordinointityö on varsin raskasta ja väärinkäytettynä sillä voidaan hukata hajauttamalla saavutettu tehokkuushyöty. Suorituskykyinen hajautettu järjestelmä tulisikin suunnitella siten, että sen alijärjestelmät olisivat mahdollisimman itsenäisiä ja järjestelmässä suoritettaisiin niin vähän hajautettuja tapahtumia kuin mahdollista.

3.3 Hajautetut tapahtumankäsittelystandardit ja -arkkitehtuurit

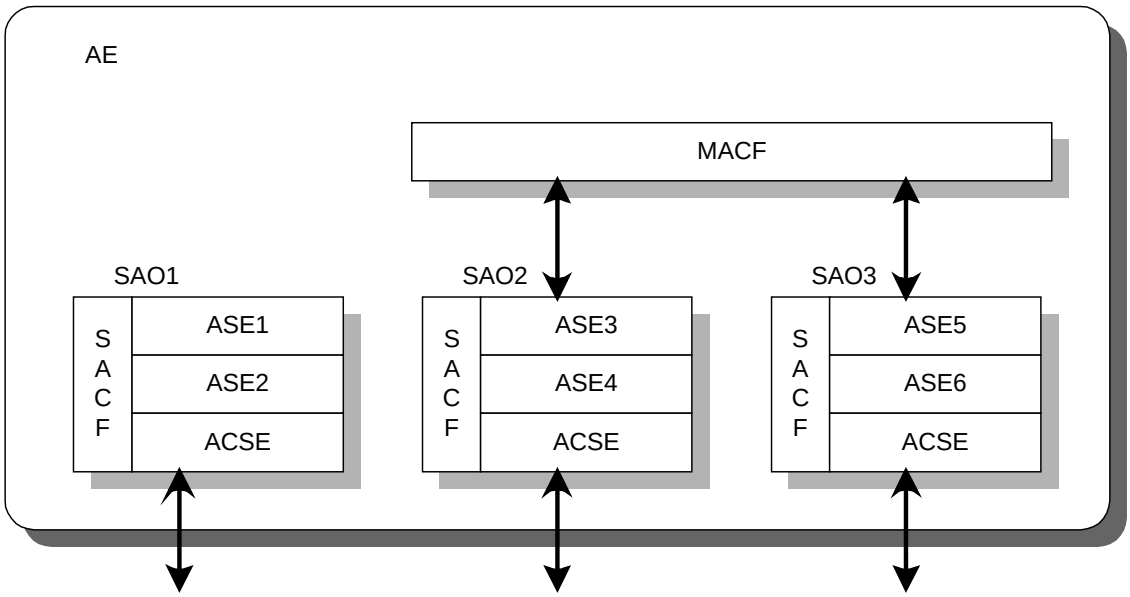
Tapahtumankäsittelyn standardointi on pääosin kulkenut toteutuksien viitoittamalla tiellä. Monet standardoidut komponentit tai mekanismit on siis lainattu suoraan tai lähes suoraan valmiista toteutuksista. Hajautettujen tapahtumien standardointi on keskittynyt avointen rajapintojen, kommunikointimenetelmien ja kaksivaihevahvistusmekanismien määrittämiseen.

3.3.1 OSI TP

OSI TP -standardin (Open Systems Interconnection) kehitys on aloitettu vuonna 1986. Standardia alettiin kehittää, sillä tarve kytkeä useiden eri valmistajien tapahtumanhallintaohjelmistoja (TP monitor) toisiinsa laajempien järjestelmien muodostamiseksi oli suuri. IBM, joka tuolloinkin oli suuri ja merkittävä tekijä markkinoilla, oli ratkaissut ongelman omien tuotteidensa tapauksessa SNA-protokollapinoon (Systems Network Architecture) kuuluvalla LU6.2-protokollalla. Osa toisistakin valmistajista tuki IBM:n LU6.2-protokollaa, mutta usein valmistajat käyttivät

omia ratkaisujaan. Tämän takia OSI TP -standardia lähdettiin kehittämään ja koska IBM oli yksi johtavista standardin kehittäjistä, muistuttaa OSI TP monessa suhteessa hyvin paljon IBM:n omaa LU6.2:ta. OSI TP ei siis määritä mitään ohjelmointirajapintoja vaan ainoastaan viestirajapinnan, jonka avulla tapahtumiin osallistuvat komponentit voivat kommunikoida keskenään. [Har1991]

OSI TP kuuluu OSI-kerrosmallin seitsemännelle kerrokselle, eli sovelluskerrokselle. OSI-mallissa sovelluskerrokset eivät keskustele keskenään, eikä sovelluskerrokselle ole mitään määrätty palvelua tai toimintoa. Sen sijaan kommunikointi käydään sovelluskerroksella olevien ASE-elementtien (Application Service Element) välillä. ASE siis sisältää määritykset yhdelle palvelulle tai ainakin osalle siitä ja sovelluskerros AE (Application layer Entity) koostuu useista ASE-elementeistä. SAO (Single Association Object) on taas yhden palvelun vaatima yhdistelmä ASE-elementtejä. SAO sisältää aina yhden tai useamman ASE-elementin ja lisäksi SACF-elementin (Single Association Control Function), jonka tehtävänä on määrittää ASE-elementtien välistä yhteistyötä. Lisäksi yhden SAO:n sisällä aina yksi ASE-elementti toimii ACSE-elementtinä (Association Control Service Element), eli se ohjaa SAO:oon kuuluvien ASE-elementtien toimintaa. Tämän lisäksi sovelluskerrokseen voi kuulua vielä MACF-elementti (Multiple Association Control Function) ja sen tehtävänä on ohjata eri SAO:jen välistä yhteistyötä ja toimintaa. Kuva 13 esittää OSI-kerrosmallin sovelluskerrosta ja sen sisältämiä komponentteja. [Har1991]



Kuva 13. OSI-kerrosmallin sovelluskerros.

Tapahtumankäsittelyä hyödyntävät sovellukset tarvitsevat jokaiseen SAO:oon ainakin ACSE:n, tapahtumankäsittely ASE:n eli TPASE:n ja vähintään yhden käyttäjä-ASE:n. Lisäksi SAO:ssa voi olla myös vahvistuksesta, yhtäaikaisuudesta ja virheistä palautumisesta vastaava ASE, joka on määritetty OSI CCR -standardissa (Commitment, Concurrency and Recovery). Tapahtumankäsittelyjärjestelmä käyttää TPASE-elementtiä luodakseen tapahtumien hallintaan liittyvät viestit. Tällaisia viestejä ovat esimerkiksi dialogiin tai yhteyksien muodostamiseen liittyvät viestit. Käyttäjä-ASE:n tehtävänä on varsinaisten tietoa sisältävien PDU:iden luominen. [Har1991]

Yhdessä OSI TP ja OSI CCR määrittävät järjestelmänlaajuiset tapahtumatunnisteet sekä edellä kuvatun kaltaisen kaksivaihevahvistusmekanismin järjestelmien välille. OSI TP -standardia voidaankin pitää kaksivaihevahvistusprotokollan määrittävän standardina. OSI TP -standardi ei kuitenkaan ole koskaan saanut merkittävää jalansijaa, eikä siihen perustuvia toteutuksia ole juurikaan olemassa. [Ber1997]

3.3.2 Open Group ja X/Open DTP

Open Group on kansainvälinen toimittaja- ja teknologiariippumaton yhtymä, jonka tavoitteena on helpottaa uuden tekniikan käyttöönottoa kattavien määritysten, standardien ja laadunvarmistus prosessien avulla. Open Group syntyi X/Open:in ja OSF:n (Open Software Foundation) yhdistyttyä yhdeksi organisaatioksi. [Ope2003]

Open Group:in hajautettua tapahtumankäsittelyä koskevat standardit ovat periytyneet X/Open yhtymän standardeista, jotka puolestaan ovat lainanneet ideoita niin IBM:n jo vuonna 1968 esittelemästä CICS-järjestelmästä (Customer Information and Control System) kuin BEA Systems:in Tuxedo:sta. CICS oli aikoinaan ensimmäinen järjestelmä, jossa oli erillinen tapahtumanvalvojakomponentti koordinoimassa tapahtumien suoritusta. Tässä työssä Open Group:in hajautettua tapahtumankäsittelyä koskevista standardeista käytetään nimitystä X/Open DTP (Distributed Transaction Processing), sillä tämä nimi on organisaatiomuutoksista huolimatta vakiintunut ja yleisesti käytössä. X/Open DTP -standardin uusin versio 3 on julkaistu jo vuonna 1996. Kyse on siis jo vanhasta ja kypsästä tekniikasta, joka ei ole kokenut muutoksia useaan vuoteen. Tekniikka on kuitenkin edelleen laajalti käytössä ja monet nykyisistä järjestelmistä tarjoavat X/Open DTP -standardien mukaiset rajapinnat ja toiminnallisuudet.

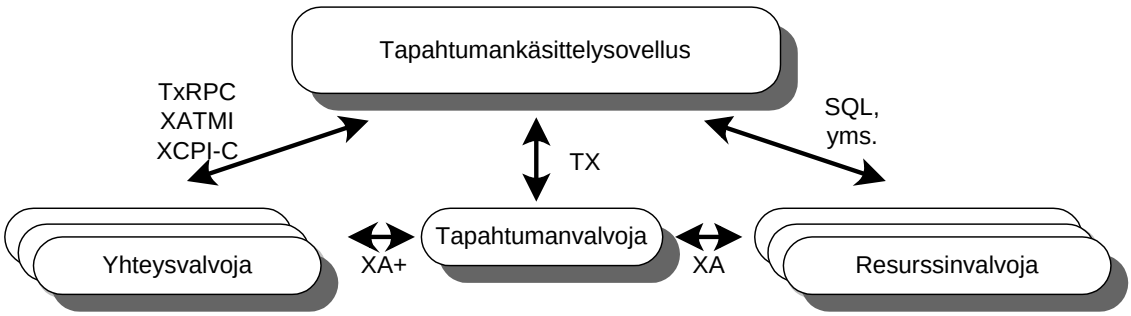
X/Open DTP -standardien tarkoituksena on määrittää tapahtumankäsittelyyn tarvittavat komponentit sekä rajapinnat niiden välille. Alkuperäisenä ideana on ollut, että järjestelmät voivat sisäisesti käyttää mitä tahansa protokollaa vahvistuksen toteuttamiseen mutta järjestelmien välinen kommunikointi toteutettaisiin OSI TP -protokollan avulla. X/Open DTP:n tärkeimmät standardit ovat:

- TX-standardi määrittää ohjelmointirajapinnan, jonka avulla tapahtumankäsittelysovellus voi kommunikoida tapahtumanvalvojan kanssa. Standardi määrittää siis tapahtumanhallintaan käytetyt aloita, vahvista ja peru operaatiot.

- TxRPC-standardi määrittää rajapinnan tapahtumankäsittelysovelluksen RPC kutsuja käyttävän yhteysvalvojan välille.
- XA-standardi määrittää tapahtumanvalvojan ja paikallisen resurssinvalvojan välisen rajapinnan. Tämä on yleisin ja valmistajien osalta kaikkein tuetuin X/Open DTP -standardi.
- XA+-standardi laajentaa tavallista XA-standardia määrittämällä rajapinnan tapahtumanvalvojan ja yhteysvalvojan (communications resource manager) välille.
- XATMI-standardi määrittää rajapinnan tapahtumankäsittelysovelluksen ja asiakas/palvelin mallia käyttävän yhteysvalvojan välille.
- XCPI-P-standardi määrittää rajapinnan tapahtumankäsittelysovelluksen ja IBM:n CICS-järjestelmän ohjelmointirajapintaa käyttävän yhteysvalvojan välille.
- STDL-standardi tapahtumien järjestelmäriippumattomalle kuvauskielelle.

Hieman yleistettynä X/Open DTP -standardia voidaan pitää tapahtumankäsittelyarkkitehtuurina OSI TP -standardissa määritetylle hajautetulle kaksivaihevahvistusprotokollalle. [Ber1997] [Ope2002]

Hajautettujen tapahtumien käsittelyyn osallistuu usein useampi kuin yksi sovellus. Tämän takia X/Open DTP -arkkitehtuuriin kuuluu lisäksi erillinen yhteysvalvoja, jonka tehtävän on tarjota kommunikointikanava ja standardoidut operaatiot sovellusten väliseen kommunikointiin. Kuva 14 esittää X/Open DTP -arkkitehtuurin ja siihen liittyvät tärkeimmät komponentit ja niiden väliset rajapinnat.



Kuva 14. X/Open hajautettu tapahtumankäsittelyarkkitehtuuri. [Ber1997]

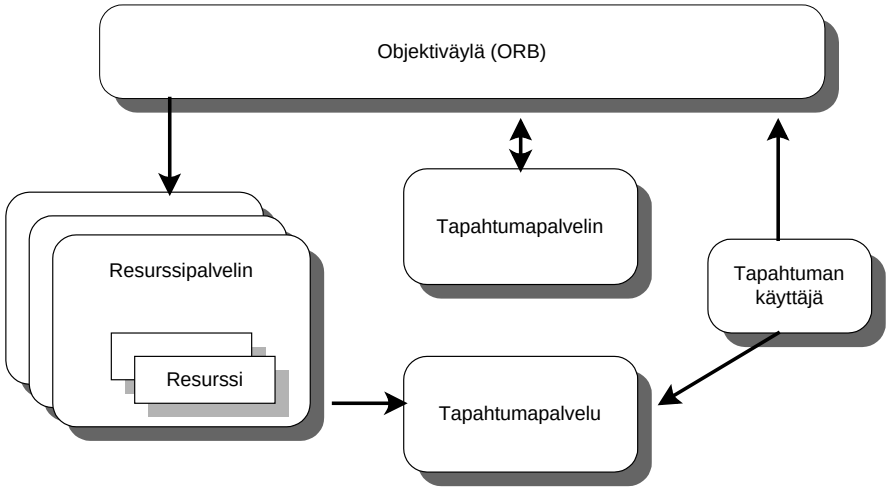
3.3.3 CORBA-tapahtumapalvelu

OMG:n (Object Management Group) CORBA (Common Object Request Broker Architecture) on eräs tunnetuimmista ja yleisimmin käytössä olevista hajautetuista arkkitehtuureista. Se on olio-pohjainen arkkitehtuuri, joka tarjoaa sovelluksille läpinäkyvän hajautuksen ja laajan valikoiman valmiita palveluita. OTS (Object Transaction Service) on eräs CORBA arkkitehtuurin keskeisiä palveluita ja sen tarkoituksena on mahdollistaa tapahtumankäsittelyn toteuttaminen hajautettuihin ja olio-pohjaisiin sovelluksiin. OTS on myös mahdollista yhdistää perinteisiin tapahtumankäsittelyjärjestelmiin, jolloin olio-pohjaisten ja perinteisten järjestelmien toisiinsa liittäminen helpottuu merkittävästi. Käytännössä liityntä perinteisiin järjestelmiin näkyy muun muassa seuraavin tavoin:

- Yksittäinen tapahtuma voi hyödyntää sekä ORB:n (Object Request Broker) päällä suoritettavia että perinteisiä sovelluksia ja resursseja.
- OTS:n toteutuksen tulee olla yhteensopiva X/Open-arkkitehtuurin kanssa.
- Käytössä olevia perinteisiä ohjelmia ja resurssinvalvojia tulee voida hyödyntää olioiden kautta.
- Olioita ja niiden resursseja tulee voida hyödyntää olemassa olevien ohjelmistojen ja resurssinvalvojien avulla.

[Sie2000][By1995]

CORBA version 3 mukaisen OTS:n tulee tukea ainakin yksitasoista tapahtumamallia. Ketjutettua mallia ei tueta, mutta tuki sisäkkäiselle mallille on valinnainen. Tästä johtuen sisäkkäisten tapahtumien tuki on valmistajakohtainen ja sen löytyminen pitää varmistaa tapauskohtaisesti. OTS toteutuksia löytyy useilta eri valmistajilta, mutta kaikkien tulee täyttää CORBA määrityksen mukaiset vaatimukset.



Kuva 15. Corba-arkkitehtuurin tapahtumapalvelu.

3.3.4 J2EE-tapahtumapalvelu

Sun Microsystems on määritellyt kolme erillistä Java-ympäristöä sovelluspalvelimilla (application server) suoritettavia Java-ohjelmia varten. Nämä ympäristöt ovat:

- J2ME (The Java 2 Platform, Micro Edition),
- J2SE (The Java 2 Platform, Standard Edition) ja
- J2EE (The Java 2 Platform, Enterprise Edition).

Jokainen ympäristö sisältää itseään pienempien ympäristöjen ominaisuudet, joten J2EE ympäristö on näistä laajin ja sisältää myös kaikki J2SE:n ja J2ME:n sisältämät määritykset. J2EE sisältää JTA- (Java Transaction API) ja JTS-teknologiat (Java Transaction Service), joten se on tämän työn kannalta kiinnostavin Java-ympäristö. [Rom1999]

Java-tapahtumapalvelu on toteutettu jakamalla CORBA-arkkitehtuurista tuttu OTS kahteen erilliseen osaan. JTS on tarkoitettu järjestelmätoimittajille ja se määrittää rajapinnat ja siirrettävät oliot, joita tapahtuman- ja resurssinvalvojat käyttävät kommunikointiin. JTS määrittää siis kuinka tapahtumanvalvoja toteutetaan ja sen avulla sovelluspalvelinten kehittäjät voivat tehdä järjestelmistä yhteensopivia, eli tapahtuma

voi käsitellä resursseja useilla eri valmistajien sovelluspalvelimilla. Hajautettua tapahtumaa suorittava JTS-standardin mukainen tapahtumanvalvoja käyttää standardeja OTS-rajapintoja ja IIOP-protokollaa (Internet Inter ORB Protocol) kommunikoidessaan toisten tapahtumanvalvojien kanssa. JTS siis toteuttaa sisäisesti myös CORBA-standardin mukaisen OTS:n. [Rom1999] [Sun2002]

Sovelluskehittäjien ei tarvitse tietää kuinka JTS toimii, sillä heitä varten on tehty erillinen rajapinta, JTA. JTA koostuu kahdesta erillisestä rajapinnasta, joista toinen on X/Open XA –standardia noudattaville resurssinvalvojille ja toinen J2EE:n oma rajapinta, joka mahdollistaa tapahtumien vaatiman logiikan ohjelmoimisen sovelluksen lähdekoodiin käyttäen rajapinnan tarjoamia tapahtumanhallintamenetelmiä. J2EE-standardi tukee ainoastaan yksitasoisia tapahtumia. [Rom1999] [Sun2002]

3.3.5 Web Services -tapahtumapalvelu

WWW:tä käytetään kasvavassa määrin myös sovellusten väliseen kommunikointiin ja järjestelmien yhdistämiseen. Sovellusten WWW:n kautta käytettävistä rajapinnoista ja niiden tarjoamista palveluista käytetään nimitystä Web Services. Web Services –tekniikoiden kehittämistä ohjaa W3C (World Wide Web Consortium) ja sen määrittämät työryhmät, jotka ovat:

- Web Services –arkkitehtuurista vastaava työryhmä,
- XML-protokollista vastaava työryhmä,
- Web Services –kuvauksista vastaava työryhmä ja
- Web Services –koreografiasta vastaava työryhmä.

Työryhmät koostuvat ympäri maailma olevien yritysten ja organisaatioiden edustajista. Lähes kaikilla suuremmilla tietotekniikkataloilla on edustusta Web Services -työryhmissä. [W3C2003] Työryhmien lisäksi myös valmistajat kehittävät Web Services –tekniikoita joko yksin tai yhteistyössä toisten valmistajien kanssa. Nykyiset Web Services –standardit, kuten WSDL (Web Services Description Language) ja SOAP

(Simple Object Access Protocol) määrittävät protokollia Web Services –tekniikoiden yhteistoimintaa varten. IBM, Bea Systems ja Microsoft ovat määritelleet standardia Web Services –ympäristössä tapahtuvalle tapahtumankäsittelylle, mutta standardi on edelleen keskeneräinen eikä sillä ole vielä virallista hyväksyntää. [Web2002b]

WS-C (Web Services Coordination) ja WS-TX (Web Services Transaction) ovat IBM:n, Bea System:in ja Microsoft:in kehitteillä olevia standardeja tapahtumankäsittelyn toteuttamiseksi hajautetussa Web Services –ympäristössä. WS-C määrittää koordinaatiopalvelun, jonka avulla sovellusten välistä yhteistoimintaa voidaan hallita. Koordinaatiopalveluun kuuluu seuraavat osat:

- Aktivointipalvelu, jonka avulla sovellus voi luoda koordinointi-instanssin.
- Rekisteröintipalvelu, jonka avulla sovellukset voivat rekisteröityä koordinointi-instanssiin.
- Koordinaatiotyypin mukaiset protokollat.

[Web2002a]

WS-TX-standardi määrittää kaksi koordinaatiotyyppiä, atomäärisen tapahtuman (atomic transaction) ja liiketoiminta aktiviteetin (business activity) sekä niiden tarvitsemat protokollat. Atomäärinen tapahtuma on lyhytkestoinen ja kaikki tai ei mitään -periaatetta noudattava toiminto. Standardi määrittää protokollat, joiden avulla nykyiset tapahtumankäsittelyjärjestelmät voivat piilottaa omat valmistajakohtaiset protokollansa. Tavoitteena on hajautettu järjestelmä, jossa eri valmistajien tapahtumankäsittelyjärjestelmät voivat suorittaa Web Services –palveluiden käyttämiä hajautettuja tapahtumia. [Web2002b]

4 TAPAHTUMANHALLINTAOHJELMISTOT

Tapahtumanhallintaohjelmistot ovat perinteinen tapa tapahtumankäsittelystä aiheutuvien ongelmien ratkaisuun. Vanhimmat ohjelmistot ovat syntyneet jo 60-luvulla, mutta ne ovat säilyttäneet asemansa näihin päiviin saakka. Tapahtumanhallintaohjelmistot ovat laajalti käytössä liikemaailmassa. Lippujen varaukset, pankkien rahaliikenne ja tilausten käsittely ovat hyviä esimerkkejä sovellusalueista, joissa tapahtumanhallintaohjelmistoilla on vankka jalansija. Internetin myötä myös erilaiset julkisessa verkossa tapahtuvat, yleensä sähköiseen kaupankäyntiin liittyvät palvelut ovat lisänneet tapahtumanhallintaohjelmistojen tarvetta ja toisaalta aiheuttaneet niille kehityspaineita uusien tekniikoiden muodossa. Tämän johdosta tapahtumanhallintaohjelmistot ovat, pitkästä iästään huolimatta, jatkuvasti kehittyneet ja omaksuneet uusia tekniikoita kuten Java tai XML (eXtended Markup Language). [Cmu1997] [Ber1997]

Tapahtumanhallintaohjelmiston tehtävänä on luoda, suorittaa ja hallita tapahtumankäsittelysovelluksia sekä auttaa omalta osaltaan ACID-vaatimusten toteuttamisessa. Tapahtumanhallintaohjelmisto tarjoaa ympäristön, joka mahdollistaa yksittäisten tapahtumien yhtäaikaisen ja tehokkaan suorittamisen niin pienissä kuin suurissakin järjestelmissä. Tapahtumanhallintaohjelmisto ei varsinaisesti suorita tapahtumia itse, vaan se jakaa tehtävät toisille järjestelmän komponenteille ja siten ohjaa tapahtumien kulkua. X/Open DTP –mallissa tapahtumanhallintaohjelmistot toimivat lähinnä tapahtumanvalvojan roolissa. Tapahtumanhallintaohjelmistot sisältävät usein kuitenkin paljon enemmän ominaisuuksia, kuin pelkkä tapahtumanvalvojana toimiminen edellyttäisi. [Ber1997] [Orf1996]

Tapahtumanhallintaohjelmiston ehkä tärkein ominaisuus on sen kyky sitoa erilliset tapahtumien käsittelyyn osallistuvat osapuolet yhdeksi kokonaisuudeksi ja etenkin yhden rajapinnan taakse. Tämän ansiosta muun muassa kuormanjako ja mahdollinen hajautus voidaan peittää järjestelmää käyttävältä sovellukselta. Komponenttien ja yksityiskohtien piilottaminen on toisaalta johtanut siihen, että tapahtumanhallintaohjelmistot ovat yleensä riippuvaisia järjestelmästä jossa niitä

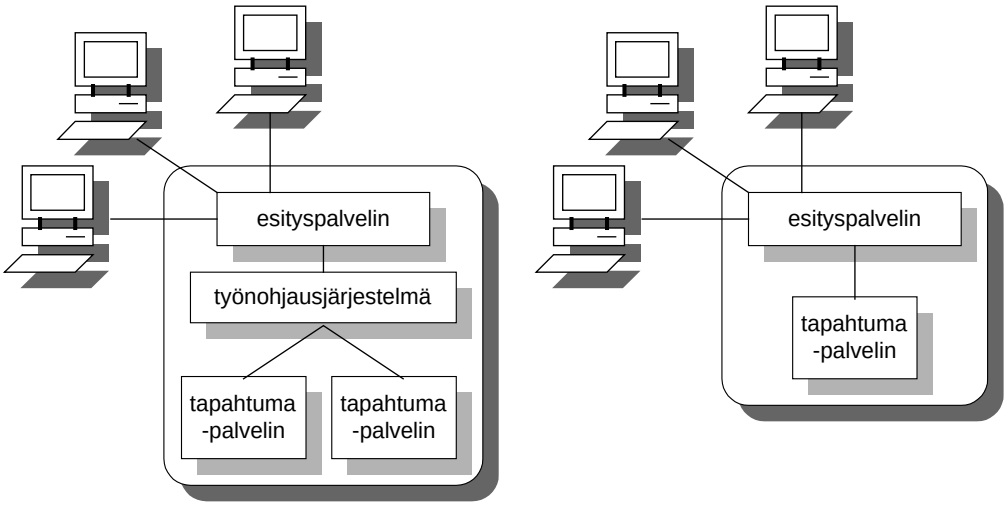
käytetään. Tästä johtuen tapahtumanhallintaohjelmistot ovat usein laite- tai käyttöjärjestelmävalmistajien toimittamia. Suurinta mahdollista suorituskykyä tavoiteltaessa ohjelmistot on usein suunniteltu ja optimoitu vain ja ainoastaan kyseisen valmistajan tuotteita silmälläpitäen. [Ber1997]

4.1 Tapahtumanhallintaohjelmiston rakenne ja toiminta

Tapahtumanhallintaohjelmistot ovat rakenteeltaan kerrosmaisia. Jokaiselle kerrokselle on määritetty tietyt tehtävät ja vastuualueet. Käyttäjän terminaalin kanssa kommunikoiva komponentti on nimeltään esityspalvelin (presentation server). Sen tehtävänä on muodostaa järjestelmään suuntautuvia pyyntöjä käyttäjältä saamansa syötteen perusteella sekä myöhemmin esittää tapahtuman tulokset käyttäjälle. Seuraavan kerroksen eli työnohjausjärjestelmän (workflow controller) tehtävänä on toimia tapahtumia ohjaavana ja hallitsevana osapuolena. Tyypillisesti tämän kerrokseen tehtäviin kuuluu tapahtumien käynnistäminen ja vahvistaminen sekä mahdollinen reitititys. Kolmannen kerroksen muodostaa tapahtumapalvelin (transaction server), joka on työnohjausjärjestelmän komennossa toimiva ja tapahtumia suorittava ohjelma. Tapahtumapalvelimena voi toimia esimerkiksi jokin tietokantapalvelin (DBMS), tai sitten tapahtumanhallintaohjelmisto voi sisältää omia tapahtumapalvelimiaan. Yhteensopivuuden varmistamiseksi tapahtumapalvelimen olisi hyvä kommunikoida resurssin kanssa jollain yleisesti käytössä olevalla datankäsittelykielellä. Ylivoimaisesti yleisin resurssityyppi on tietokanta, jolloin yleisesti hyväksytty kommunikointitapa on esimerkiksi SQL (Structured Query Language).

Tapahtumanhallintaohjelmisto joka sisältää esityspalvelimen, työnohjausjärjestelmän ja tapahtumapalvelimen on rakenteeltaan kolmikerroksinen. Toinen vaihtoehto on kaksikerrosmalli, josta työnohjausjärjestelmä on jätetty pois. Tämä malli sopii hyvin pienille ja yksinkertaisille järjestelmille, joissa on tyypillisesti vähän käyttäjiä ja vain yksi tapahtumapalvelin. Tällöin on usein järkevää toteuttaa suora yhteys esityspalvelimelta tapahtumapalvelimelle. Kaksikerroksinen

tapahtumanhallintaohjelmisto ei kuitenkaan skaalaudu kovinkaan hyvin. Mikäli käyttäjien ja datan määrä kasvaa ja tarvitaan useampia esitys- ja tapahtumapalvelimia, niin näiden välisten kommunikointiyhteyksien määrä kasvaa eksponentiaalisesti. Tähän ongelmaan työnohjausjärjestelmät tarjoavat hyvän ratkaisun. Kuva 16 esittää kolme- ja kaksikerroksisten tapahtumanhallintaohjelmistojen rakenteen.



Kuva 16. Kolme- ja kaksikerroksiset tapahtumanhallintaohjelmistot.

Suurissa ja hajautetuissa tapahtumankäsittelyjärjestelmissä työnohjausjärjestelmien ja esityspalvelinten sijoittelu vastaa tehokkuusvaatimukset huomioiden usein maantieteellistä sijoittelua. Tyypillisesti yhden alueen esityspalvelimet ovat yhteydessä samaan, oman alueensa työnohjausjärjestelmään. Tällä pyritään pääasiassa vähentämään kommunikoinnin aiheuttamaa turhaa viivettä.

Järjestelmään tuleva pyyntö voi aiheuttaa useita erillisiä tapahtumia tai tapahtuman, joka koskee useampaa tapahtumapalvelinta. Tällaisissa tapauksissa on tärkeä tietää mille tapahtumapalvelimille tapahtuma annetaan suoritettavaksi ja missä järjestyksessä tämä tehdään, mikäli tapahtuma koskee useampaa palvelinta. Tämä hallintaan käytetään työnkulkuohjaajan sovelluskohtaista osaa, jonka tarkoituksena on ohjata tapahtuman kulkua järjestelmässä. Jotkut tapahtumanhallintaohjelmistot käyttävät erillistä ohjelmointikieltä tapahtumien kulun kuvaamiseen. Tällaisia kieliä ovat muun muassa TDL (Task Definition Language) ja STDL (Structured Transaction Definition

Language). Kaikilla tapahtuman kuvaamiseen käytetyillä kielillä on omat ominaisuutensa, mutta yksi niitä selvästi erottava tekijä on rinnakkaisen suorituksen tukeminen. Monet tapahtumat ovat luonteeltaan sellaisia, että niitä voitaisiin suorittaa yhtäaikaisesti useammalla tapahtumapalvelimella. Tästä huolimatta vain harvat tapahtumankuvauskielet tukevat sitä. Muita tapoja tapahtumien kulun ohjaamiseen ovat esimerkiksi dynaamisten kirjastofunktioiden käyttö tai erilliset asetustiedostot (configuration file). [Ber1997]

Tapahtumanhallintaohjelmistoja verrataan usein tietokantapalvelimiin ja niillä suoritettaviin, tietokantaproseduureilla toteutettuihin tapahtumiin, Vertaus osuu hyvin lähelle oikeaa, sillä molemmat toteuttavat pääpiirteittäin samoja asioita. Molemmissa asiakas kutsuu järjestelmään kuvattuja tapahtumia, joita järjestelmä suorittaa. Perinteinen tietokantapalvelin ja sitä käyttävä sovellus on kuitenkin selvästi kaksikerroksinen malli ja siihen pätee siis aiemmin käsitellyt kaksikerroksisen mallin ongelmat. Nämä ongelmat voidaan ratkaista tietokantapalvelimissa käytettävän työnohjausjärjestelmän avulla, mutta sekään ei ratkaise kaikkia ongelmia. Eräs vakavimmista ongelmista koskee hajautettuja tapahtumia. Tietokantapalvelimet tukevat yleensä tiedon hajauttamista useampaan saman valmistajan tietokantaan, mutta entä jos järjestelmässä onkin useita eri valmistajien tietokantoja? Tai entäpä jos osa tapahtumassa käsiteltävistä resursseista onkin jotain muita resursseja kuten jonoja tai tiedostoja? Tapahtumanhallintaohjelmistojen etu onkin mahdollisuus hyödyntää useita erilaisia ja eri valmistajien resursseja. [Ber1997]

4.2 Kommunikointimallit

Tapahtumanhallintaohjelmiston eri komponenttien täytyy luonnollisesti kommunikoida keskenään. Kommunikoinnin tarve ja merkitys kasvaa hajautetuissa järjestelmissä, mutta usein yhdessäkin laitteistossa suoritettava ohjelmisto on järkevää pilkkoa pienempiin ja itsenäisiin osiin. Eri prosessien välisen kommunikoinnin toteuttamiseen on kolme eri tapaa:

1. RPC-kutsut (Remote Procedure Call), jotka simuloivat yhden prosessien sisäisten aliohjelmakutsujen toimintaa.
2. Viestipohjainen lähestymistapa, eli prosessit vastaanottavat ja lähettävät toisilleen viestejä.
3. Viestijonot, eli molemmat prosessit kommunikoivat erillisen viestijonon kanssa. Toinen prosessi laittaa viestinsä jonoon, josta toinen voi sen taas lukea.

Näistä malleista kahdesta ensimmäisestä käytetään nimitystä aktiivinen tapahtumankäsittely, eli asiakas ja pyynnön toteuttava palvelin ovat suorassa yhteydessä toisiinsa ja palvelin toteuttaa asiakkaan komentoja heti käskyn saatuaan. Viestijonot poikkeavat edellisistä, sillä niissä viestit kulkevat erillisen viestijonon kautta, jolloin palvelimen ei tarvitse reagoida viestiin välittömästi, sillä viesti pysyy tallessa viestijonossa. Kuten myöhemmin todetaan, tästä ominaisuudesta on hyötyä useissa tilanteissa.

4.2.1 RPC-kommunikointi

RPC-kutsujen avulla hajautetun ohjelmiston toteutus voi tehokkaasti mallintaa yhden prosessin sisäistä, aliohjelmakutsuihin perustuvaa kommunikointia. Käytännössä prosessi voi kutsua toisessa prosessissa suoritettavaa aliohjelmaa saaden tuloksen kun kutsu palaa. RPC-kommunikointimalli on normaalisti synkroninen, eli RPC-kutsun tehnyt prosessi odottaa kunnes toinen prosessi on suorittanut tehtävän ja palauttanut tuloksen kutsujalle. RPC-kommunikoinnista on olemassa myös asynkronisia versioita, mutta ne muistuttavat rakenteeltaan viestipohjaista lähestymistapaa ja ovatkin yleensä yhdistelmiä molemmista malleista.

RPC:stä on olemassa useita eri versioita. Tapahtumanhallintaohjelmistojen tapauksessa yleisimmin käytössä oleva RPC-mekanismi perustuu OSF DCE –malliin (Open Software Foundation’s Distributed Computing Environment). Muista yleisesti käytetyistä malleista voisi mainita esimerkiksi Sun Microsystems:in ONC RPC -mallin (Open Network Computing). Myös OMG:n CORBA-arkkitehtuuri sisältää RPC-malliin perustuvan viestinnän. [Ber1997] [Blo1992]

4.2.2 Viestipohjainen kommunikointi

Viestipohjaisessa mallissa kommunikoivat osapuolet lähettelevät toisilleen viestejä. Molemmat osapuolet voivat itsenäisesti lähettää ja vastaanottaa viestejä, eli kyse on oikeastaan P2P-viestinnästä (peer-to-peer messaging). Tapahtumankäsittelyn yhteydessä käytettyyn P2P-viestintään on määritelty useita protokollia, joista yleisimmin käytetty on todennäköisesti IBM:n SNA-protokollapinoon kuuluva LU6.2-protokolla. [Ber1997]

P2P-viestintään osallistuvat osapuolet voivat kommunikoida joko vuoro- (half duplex) tai kaksisuuntaisesti (full duplex). Vuorosuuntaisessa kommunikoinnissa viestien lähetys ja vastaanotto tehdään vuorotellen, eli toinen osapuoli on lähettävässä ja toinen vastaanottavassa tilassa. Roolit voivat toki vaihtua tilanteen mukaan. Kaksisuuntaisessa kommunikoinnissa ei ole vastaavanlaisia tiloja, vaan molemmat osapuolet voivat lähettää ja vastaanottaa viestejä samanaikaisesti. OSI TP -protokolla on esimerkki kaksisuuntaisesta kommunikoinnista, kun taas IBM:n LU6.2-protokolla käyttää vuorosuuntaista mallia.

4.2.3 Viestijonot

Viestijonoilla pyritään ratkaisemaan yhteyksien katkeamisesta tai prosessien vikaantumisesta aiheutuvia kommunikointiongelmia. Jos esimerkiksi järjestelmän käyttäjän ja esityspalvelimen välinen yhteys menee poikki tai palvelin vikaantuu, niin käyttäjä joutuu pyynnön lähettämiseksi odottamaan yhteyden palautumista. Tai vaihtoehtoisesti jos yhteys, asiakas tai palvelin vikaantuu vastausta lähetettäessä, niin pahimmassa tapauksessa koko palvelimen lähettämä vastausviesti katoaa. Viestijonojen avulla ongelma on ratkaistavissa, sillä viestien lähettämisen ja vastaanottamisen välillä ei ole suoraa riippuvuutta. Viestijonoista voi olla suurta apua myös kuormanjaon (load balancing) toteuttamiseen, sillä usea eri palvelin voi itsenäisesti poimia asiakkaiden jättämiä pyyntöjä viestijonosta. [Ber1997]

Viestijono, jonka kautta viestit kulkevat, on osa tapahtumankäsittelyprosessia ja siten sen kautta kulkevat operaatiot täytyy saada pysyviksi tai peruutetuiksi riippuen tapahtuman lopputuloksesta. Viestijono on usein toteutukseltaan sellainen, että jonossa olevat viestit tallennetaan tai voidaan tallentaa johonkin pysyvään muistiin, esimerkiksi kiintolevylle. Tällä pyritään välttämään viestien katoamista järjestelmän vikatilanteissa. Monet viestijonot mahdollistavat lisäksi viestikohtaiset määritykset siitä, että tallennetaanko ne siirron aikana pysyvään muistiin vai työmuistiin. Viestijono-ohjelmistoja on saatavana kaupallisina tuotteina useilta eri valmistajilta. Esimerkkeinä voidaan mainita IBM:n MQSeries, Sonic Software:n SonicMQ, Bea Systems:in MessageQ ja MSMQ (Microsoft Message Queuing).

4.3 Kaupalliset tapahtumanhallintaohjelmistot

Markkinoilla on useita tapahtumanhallintaohjelmistoja useilta eri valmistajilta. Ohjelmistot voivat olla yleiskäyttöisiä tai sitten ne on suunniteltu ainoastaan tiettyä tarvetta ja arkkitehtuuria varten. Oikean tuotteen valinta perustuukin lähinnä käyttötarpeen ja ympäristön sanelemien vaatimusten mukaan. Seuraavissa kappaleissa esitellään joitakin tapahtumien käsittelyyn tarkoitettuja ohjelmistoja ja niiden ominaisuuksia.

4.3.1 IBM:n tuotteet

IBM on jo kauan aikaa ollut johtava tapahtumanhallintaohjelmistojen ja –järjestelmien toimittaja. IBM:ltä löytyy useita erilaisia tuotteita, jotka on tarkoitettu eri tyyppisten ja kokoisten ongelmien ratkaisuun.

IBM:n CICS tuli markkinoille jo vuonna 1968. Se oli tulleessaan eräs ensimmäisistä tapahtumankäsittelypalveluita tarjoavista järjestelmistä ja se on edelleen yksi yleisimmin käytetyistä. CICS:n kehityksen taustalla oli tietojärjestelmien monimutkaistuminen ja sitä kautta tarve käsitellä interaktiivisia tapahtumia luotettavasti ja tehokkaasti.

CICS edusti alunperin suuren laskentatehon omaavilla palvelinkoneilla, eli keskustietokoneilla (mainframe), suoritettavaa tapahtumanhallintaohjelmistoa, mutta nykyisin kehitys ja erilaiset vaatimukset ovat jakaneet CICS-järjestelmän useisiin osiin. IBM:n palvelinkoneille tarkoitettu järjestelmä suoritetaan yhdessä käyttöjärjestelmän prosessissa ja se toteuttaa itse omat sisäiset palvelunsa käyttäen muun muassa käyttöjärjestelmän säikeitä (thread) niiden toteutukseen. Huolimatta siitä, että CICS-järjestelmä on suunnittelunsa puolesta helposti siirrettävä (portable), on sen järeisiin palvelinkoneisiin tarkoitettu versio saatavilla vain IBM:n omille laitteistoille ja käyttöjärjestelmille. Tätä dokumenttia kirjoitettaessa uusin CICS:n versio on saatavilla IBM:n z/OS, OS/390 ja VSE/ESA käyttöjärjestelmille. Muihin CICS:n versioihin palataan myöhemmin TXSeries-järjestelmän yhteydessä. [Ibm2003a] [Gra1993]

Keskustietokoneille tarkoitettua CICS-järjestelmää käytetään usein hajautetussa tilassa, jolloin yksi CICS-prosessi hoitaa tietyn osan järjestelmän resursseista ja käyttäjistä. Hajautus voidaan toteuttaa joko yhden tietokoneen sisällä jakamalla järjestelmä useaksi prosessiksi tai hajauttamalla prosessit useille eri tietokoneille. Yhden koneen sisällä CICS-järjestelmät voivat keskustella käyttäen käyttöjärjestelmän jaettuun muistiin perustuvaa viestinvälitystä. Tämä on huomattavasti lähi- tai alueverkkoja tehokkaampi ratkaisu. Mikäli hajautus kuitenkin toteutetaan usean tietokoneen välille, käytetään järjestelmän osien väliseen kommunikointiin IBM:n omaa LU6.2-protokollaa.

IBM:n IMS (Information Management System) on yhdistelmä tietokantapalvelinta ja tapahtumanhallintaohjelmistoa. Myös IMS:n juuret ulottuvat kauas historiaan sillä sen kehittäminen on saanut alkunsa jo 60-luvulla. Myös IMS on perinteisesti ollut keskustietokoneilla suoritettava järjestelmä, jossa ”suuren laskentakapasiteetin keskustietokone palvelee jopa tuhansia sovelluksia ympäri maailman”. [Gra1993] IMS on eräs yleisimmistä tapahtumankäsittelyjärjestelmistä.

IMS perustuu viestijonojen avulla toteutettuun kommunikointiin ja se on tiukasti integroitu IBM:n omiin käyttöjärjestelmiin ja siten myös laitteisiin. IMS, toisin kuin CICS, käyttää hyväkseen käyttöjärjestelmän tarjoamia resursseja ja palveluita, eikä

korvaa niitä omilla ratkaisuillaan kuten CICS tekee. Tästä johtuen IMS järjestelmä on tiukemmin sidoksissa käyttöjärjestelmään, joten sen siirtäminen toiselle käyttöjärjestelmälle on vaikeaa. Vaikka IMS sisältää oman tietokantapalvelimensa, tarjoaa se mahdollisuuden myös ulkoisten tietokantojen ja resurssien käyttöön. [Ibm2003b]

IBM Encina on alunperin kehitetty Carnegie Mellon:in yliopistossa kokeelliseksi tapahtumanhallintaohjelmistoksi. Sen kehitystä sponsoroi IBM, jonka tytäryhtiö Transarc aloitti järjestelmän markkinoimisen vuonna 1991. Nykyisin Encina:aa markkinoidaan kuitenkin osana IBM:n TXSeries tuotetta. Encina:n rakenne ja toiminta perustuu OSF DCE –malliin, joten Encina käyttääkin kommunikointiin DCE –mallissa määriteltyjä RPC-mekanismeja. [Ber1997]

IBM TXSeries edustaa IBM:n uudempaa tuotesukupolvea. Se on IBM:n aluevaltaus omien käyttöjärjestelmiensä ulkopuolisille markkinoille ja se sisältää itse asiassa kaksi erillistä tapahtumanhallintaohjelmistoa. TXSeries-ohjelmistoon on sisällytetty mukautetut versiot sekä CICS- että Encina-tapahtumanhallintaohjelmistoista. Lisäksi TXSeries sisältää lukuisan määrän muita IBM:n myös erikseen myytäviä komponentteja, kuten MQSeries viestijono-ohjelmiston. TXSeries-järjestelmässä on tuki useille eri käyttöjärjestelmille ja tuettujen järjestelmien listalla ovat muun muassa:

- Microsoft Windows NT ,
- Microsoft Windows 2000,
- Sun Solaris sekä
- HP-UX.

TXSeries on yhdistettävissä perinteisiin CICS- ja Encina-järjestelmiin, joten yhteensopivuus olemassa olevien ratkaisujen kanssa on taattu. [Ibm2003c] [Ric1997]

IBM TPF (Transaction Processing Facility) on tapahtumankäsittelyyn kehitetty käyttöjärjestelmä, joka on tarkoitettu äärimmäisen raskaiden tapahtumankäsittely-ympäristöjen käyttöön. Käyttöjärjestelmä toimii ainoastaan IBM:n omilla

palvelinkoneilla ja prosessoreilla ja suuri osa järjestelmästä on kirjoitettu konekielellä suurimman mahdollisen suorituskyvyn saavuttamiseksi.

4.3.2 BEA Tuxedo

BEA Systems:in Tuxedo on tapahtumanhallintaohjelmisto, joka on alunpitäen suunniteltu toimimaan hajautetusti useilla itsenäisillä UNIX-koneilla. Tuxedon, kuten myös UNIX-käyttöjärjestelmän, juuret ovat AT&T:n laboratorioissa ja nykyään sen kehittämisestä ja markkinoinnista vastaa BEA Systems. Ensimmäinen kaupallinen Tuxedo:n versio esiteltiin vuonna 1983. [Gra1993]

Tuxedo on IBM:n IMS:in kaltaisesti jakautunut kahteen eri osaan. System/T sisältää hallintarajapinnan ja tapahtumien ja yhteyksien hallintaan osallistuvat komponentit. System/D on puolestaan SQL-tietokantajärjestelmä, joka peruskokoonpanossa toimii System/T:n resurssina. Järjestelmät ovat kuitenkin itsenäisiä ja System/T:tä voidaan käyttää myös toisten valmistajien tietokantaratkaisujen kanssa. Vastaavasti System/D toimii minkä tahansa X/Open XA -protokollaa tukevan tapahtumavalvojan kanssa. [Ber1997]

4.4 Yhteenveto

Järjestelmien hajauttaminen pienempiin osiin ja toisaalta erilaisten järjestelmien toisiinsa yhdistäminen ovat nostaneet esiin vaatimuksen avoimista, hajautettua tapahtumankäsittelyä käsittelevistä, standardeista. Tämän taustalla ovat ongelmat, joita aiheutuu kun tapahtumat ulottuvat useisiin eri järjestelmiin. Tapahtumien tulee hajautetussakin järjestelmässä noudattaa ACID-vaatimuksia. Niiden saavuttamiseksi joudutaan käyttämään joitain erityismenetelmiä kuten kaksivaihevahvistusta.

Hajautettu tapahtumankäsittely on laaja aihealue ja se sisältää lukuisan määrän erilaisia standardeja ja lähes standardeiksi muodostuneita tuotteita ja tekniikoita. IBM:n tuotteiden merkitystä ei voi vähätellä tapahtumankäsittelyjärjestelmien kehityksestä

puhuttaessa ja nämä tuotteet ovat osaltaan olleet vaikuttamassa myös avointen standardien kehitykseen. OSI TP ja X/Open DTP –standardit määrittävät hajautetussa tapahtumankäsittelyssä välttämättömän kaksivaihevahvistusmekanismin ja siihen tarvittavat komponentit ja rajapinnat. OSI TP –standardi on kuitenkin jäänyt teorian tasolle, eikä siihen perustuvia toteutuksia ole merkittävässä määrin olemassa. Käytännössä IBM:n LU6.2-protokolla on saanut lähes standardin aseman tapahtumanhallintaohjelmistojen välisessä kommunikoinnissa. X/Open DTP –standardiperheeseen kuuluvaa XA-standardia voidaan pitää tämän työn kannalta merkityksellisimpänä avoimena hajautettua tapahtumankäsittelyä koskevana standardina, sillä sen avulla resurssinvalvoja ja tapahtumanvalvoja voivat kommunikoida keskenään kaksivaihevahvistuksen saavuttamiseksi.

Uudemmat oliopohjaiset arkkitehtuurit, kuten CORBA ja J2EE, sisältävät kattavat määritykset myös hajautettuun tapahtumankäsittelyyn. J2EE-standardin tapahtumankäsittely on periytynyt suoraan CORBA:n OTS-standardista, joten ne ovat pohjimmiltaan yhteensopivia keskenään. Hajautettujen tapahtumaohjelmien kehittäminen CORBA- ja J2EE-ympäristöihin onkin suhteellisen yksinkertaista, sillä sovelluspalvelimet tarjoavat standardien ansiosta valmiit mekanismit ja luokat tapahtumien kontrollointiin. Web Services –tapahtumankäsittelystandardit ovat tätä työtä kirjoitettaessa vielä keskeneräisiä, joten niiden toimivuudesta ja toteutuksien määrästä on vaikea sanoa vielä mitään.

Seuraavaksi työssä tutustutaan Intellitel ONE –järjestelmään ja varsinkin sen tietokanta-arkkitehtuuriin, sekä pohditaan kuinka siihen voitaisiin lisätä tuki paikallisia ja hajautettuja tapahtumia käyttäville sovelluksille. Ratkaisun olisi hyvä olla mahdollisimman yhteensopiva olemassa olevien standardien ja tuotteiden kanssa, sillä yhtenä tavoitteena on helpottaa Intellitel ONE –järjestelmän yhdistämistä toisten valmistajien järjestelmiin ja mahdollistaa niissä suoritettavat hajautetut tapahtumat.

5 INTELLITEL ONE

Intellitel ONE perustuu NIM-malliin (Network Intelligent Middleware) ja se toteuttaa mallissa määritetyt toiminnallisuudet ja rajapinnat. Intellitel ONE tarjoaa ympäristön ja työkalut sovellusten kehittämiseen, prosessien hallintaan ja laskutukseen. Intellitel ONE järjestelmä voi toimia joko yksittäisessä (single) tai kahdennetussa (double) tilassa. Kahdennus toimii aina siten, että toinen puoli järjestelmästä on aktiivisessa (active) ja toinen passiivisessa (passive) tilassa. Puolenvaihto tapahtuu automaattisesti, mikäli aktiivisen puolen järjestelmä vikaantuu riittävän vakavasti esimerkiksi rautavian vuoksi. Tällöin passiivinen puoli muutetaan aktiiviseksi ja se ottaa liikenteen hoitaakseen. Puolenvaihto voidaan myös käynnistää manuaalisesti prosessienhallintajärjestelmän kautta esimerkiksi aktiivisen puolen huoltotoimenpiteiden vuoksi. Intellitel ONE -ohjelmisto ei toistaiseksi tue kahdennusta, jossa molemmat puolet olisivat yhtäaikaisesti aktiivisia (active-active). [Int1998b]

5.1 CVOPS

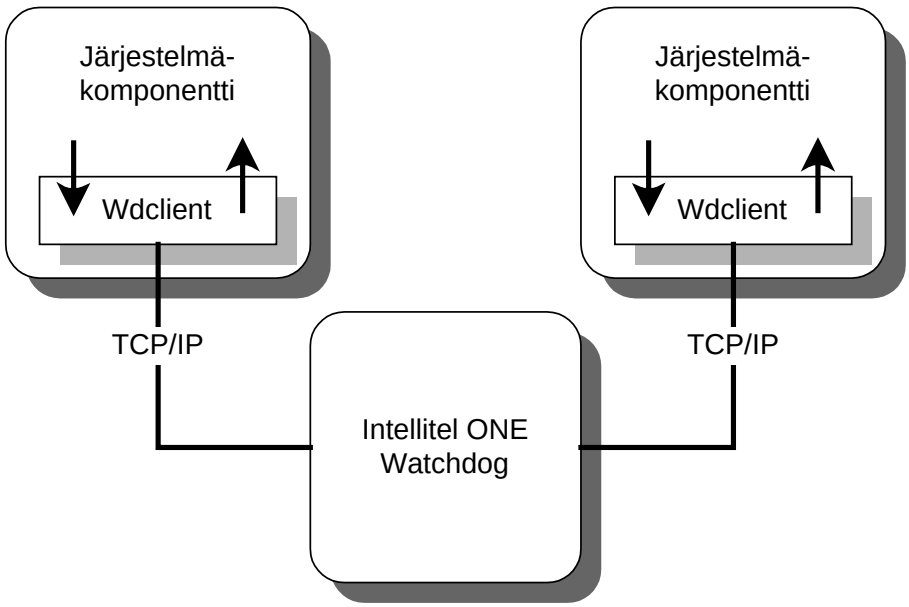
Intellitel ONE perustuu alunperin VTT:n kehittämään ja nykyisin Celtius Oy:n ylläpitämään tietoliikenneohjelmistojen suunnittelu- ja toteutusympäristöön CVOPS:iin. CVOPS on C tai C++ kielinen kehitysympäristö ja se perustuu OSI-kerrosmalliin. CVOPS tarjoaa työkaluja ja funktiokirjastoja protokollien suunnitteluun, toteutukseen, testaamiseen ja simulointiin. [Cel2003b]

CVOPS-ohjelma koostuu virtuaalitehtävistä, eli VTASK:eista (Virtual Task) ja niitä on yhdessä ohjelmassa aina vähintään yksi. VTASK:it edustavat OSI-kerrosmallin kerroksia ja ne kommunikoivat toistensa kanssa CVOPS-rajapintojen avulla. VTASK sisältää muun muassa tila-automaatin, rajapinnat toisiin VTASK:eihin ja mahdollisesti yhteys-VTASK:eja (Connection Virtual Task), joita käytetään jonkun itsenäisen kokonaisuuden kuvaamisen varsinaisen VTASK:in sisällä. Yhteys-VTASK:eja voidaan käyttää esimerkiksi usean yhtäaikaisen verkkoyhteyden kuvaamiseen, jolloin yhdellä yhteys-VTASK:illa kuvataan yhden verkkoyhteyden tilaa. [Cel2003a] [VTT1999]

5.2 Intellitel ONE prosessienhallinta

Intellitel ONE sisältää prosessienhallintajärjestelmän, jonka tehtävänä on järjestelmäkomponenttien käynnistäminen, sulkeminen sekä niiden toiminnan seuraaminen. Prosessienhallintajärjestelmä koostuu kahdesta komponentista. Intellitel ONE Watchdog on varsinainen järjestelmänvalvojakomponentti, joka tarkkailee prosessien tilaa Wdclient-komponentin avulla. Wdclient on komponentti, jonka kautta muut Intellitel ONE -prosessit ovat yhteydessä varsinaiseen järjestelmänvalvojaan.

Järjestelmää käynnistettäessä Watchdog on vastuussa muiden järjestelmäkomponenttien käynnistämisestä. Käynnistettävät prosessit se lukee järjestelmän asetustiedostosta. Käynnistymisen jälkeen järjestelmäkomponentit muodostavat yhteyden Watchdog:iin Wdclient-komponentin avulla. Wdclient muodostaa TCP/IP-yhteyden (Transmission Control Protocol / Internet Protocol) itsensä ja Watchdog-komponentin välille. Järjestelmäkomponentti kommunikoi Wdclient:in kanssa sisäistä CVOPS-rajapintaa käyttäen. Kuva 17 esittää prosessienhallintaan osallistuvat komponentit.



Kuva 17. Intellitel ONE prosessienhallinta.

Järjestelmän toimiessa Watchdog:in tehtävänä on tarkkailla komponenttien toimintaa ja tilaa ja tarpeen vaatiessa käynnistää jumiutuneet tai kaatuneet prosessit uudelleen. Prosessien tilan tarkkailuun on pääasiassa kaksi eri menetelmää:

- Watchdog:in ja Wdclient:in välisen TCP/IP-yhteyden tarkkailu.
- Viestipohjainen mekanismi (heartbeat), eli Wdclient lähettää Watchdog:ille tasaisin väliajoin viestiä merkiksi siitä, että prosessi on toimivassa tilassa.

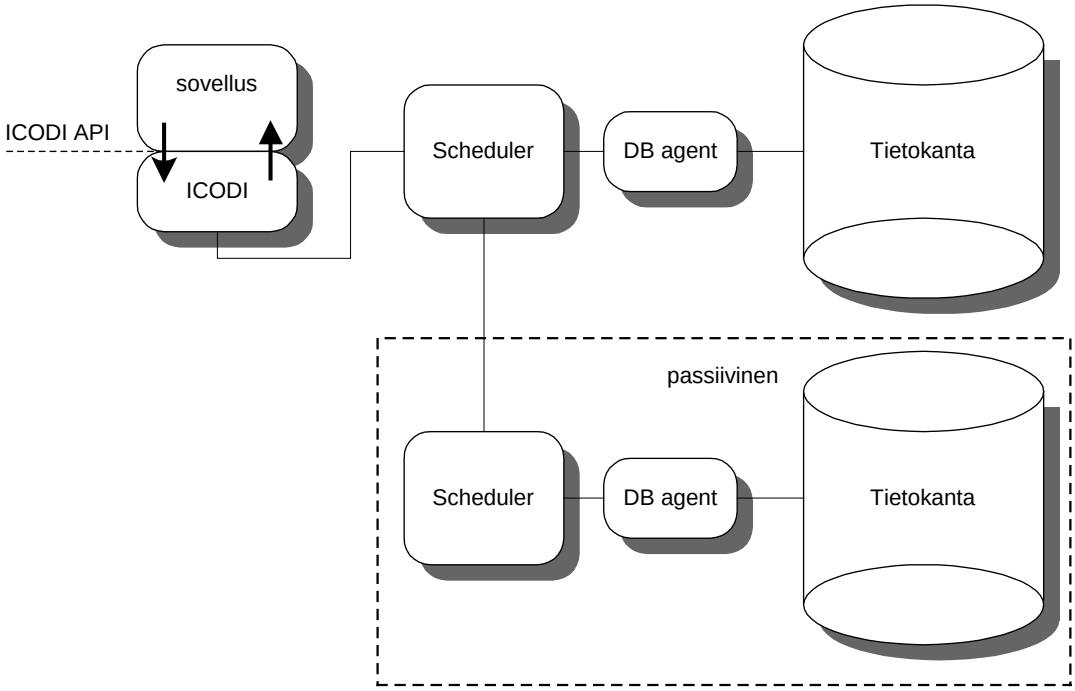
Erilaisilla tarkkailumenetelmillä pyritään huomaamaan erilaiset vikatilanteet. Mikäli TCP/IP-yhteys katkeaa, niin on selvää, ettei viestipohjaista tarkkailuakaan voida enää suorittaa. Tällöin Watchdog olettaa että prosessi on kaatunut ja pyrkii käynnistämään sen uudelleen. Pelkkä TCP/IP yhteyden tarkkailu ei kuitenkaan riitä, sillä prosessi voi ajautua ikisilmukkaan tai muuhun ikuiseen lukkotilanteeseen vaikka yhteys olisikin kunnossa. Tällöin se ei luonnollisestikaan pysty lähettämään tarvittavia viestejä Watchdog:ille, joten Watchdog olettaa prosessin jumiutuneen ja käynnistää sen uudelleen.

5.3 Intellitel ONE tietokanta-arkkitehtuuri

Intellitel ONE sisältää tietokantariippumattoman ja luotettavan tietokanta-arkkitehtuurin, jonka kautta yhteydet tietokantaresursseihin hoidetaan. Arkkitehtuurin etuna on tietokannasta riippumaton ja yhtenäinen ohjelmointirajapinta, jonka ansiosta järjestelmään kehitetyt sovellukset voivat toimia millä tahansa Intellitel ONE:n tukemalla tietokantaratkaisulla ilman erillisiä muutos- tai sovitustöitä.

Intellitel ONE:n tukema aktiivinen/passiivinen kahdennus näkyy tietokanta-arkkitehtuurissa siten, että sovelluksen käsittelemä data tallennetaan automaattisesti myös passiivisen puolen tietokantaan. Käytännössä datavirran kahdennus on toteutettu Scheduler-komponentissa. Aktiivisen puolen Scheduler lähettää saamansa datan myös passiivisen puolen Schedulerille. Jos aktiivinen puoli vikaantuu, passiivinen voi ottaa ohjat käsiinsä ja sen tietokanta on valmiiksi yhtenäisessä tilassa. Kuva 18 esittää

Intellitel ONE:n kahdennetun tietokanta-arkkitehtuurin yksinkertaisimmassa muodossaan. Sovellus on yhteydessä aktiivisen puolen Scheduler-komponenttiin, joka kahdentaa datavirran järjestelmän passiiviselle puolelle. Järjestelmän passiivinen puoli on erotettu kuvassa katkoviivalla.

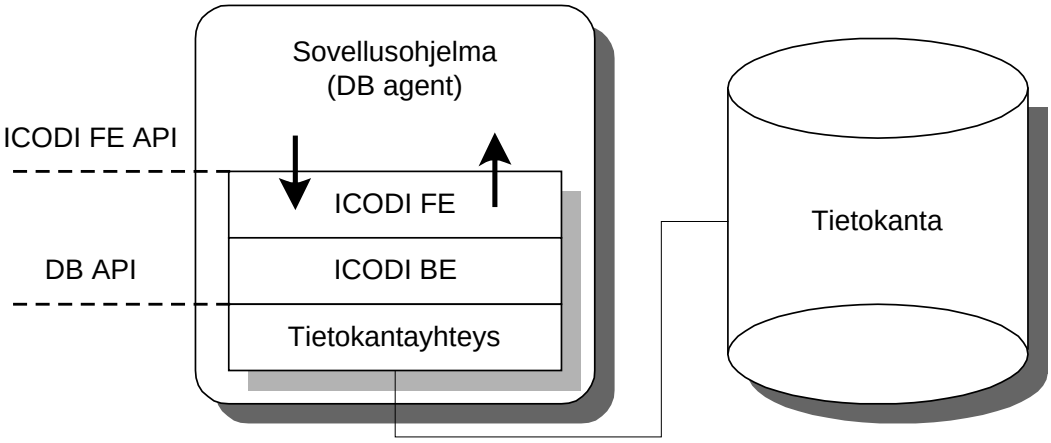


Kuva 18. Intellitel ONE kahdennettu tietokanta-arkkitehtuuri.

5.3.1 ICODI

ICODI (Intellitel Communications Database Interface) on kokoelma tietokantayhteyksiin käytettyjä komponentteja ja se sisältää ICODI FE:n (frontend), ICODI BE:n (backend), ohjelmointirajapinnan eli ICODI API:n (Application Programming Interface), kodekit ja ICODI-sanomien siirtoon tarvittavat protokollat. Niiden suunnittelussa ja toteutuksessa on pyritty yksinkertaisuuteen, tehokkuuteen ja joustavuuteen. ICODI on suunniteltu käytettäväksi relaatiotietokannan kanssa ja relaatiomalli näkyy ICODI:n kautta myös sitä käyttävälle sovellukselle. ICODI siis kuvaa relaatiomallisen tiedon ja operaatiot omissa operaatioissaan.

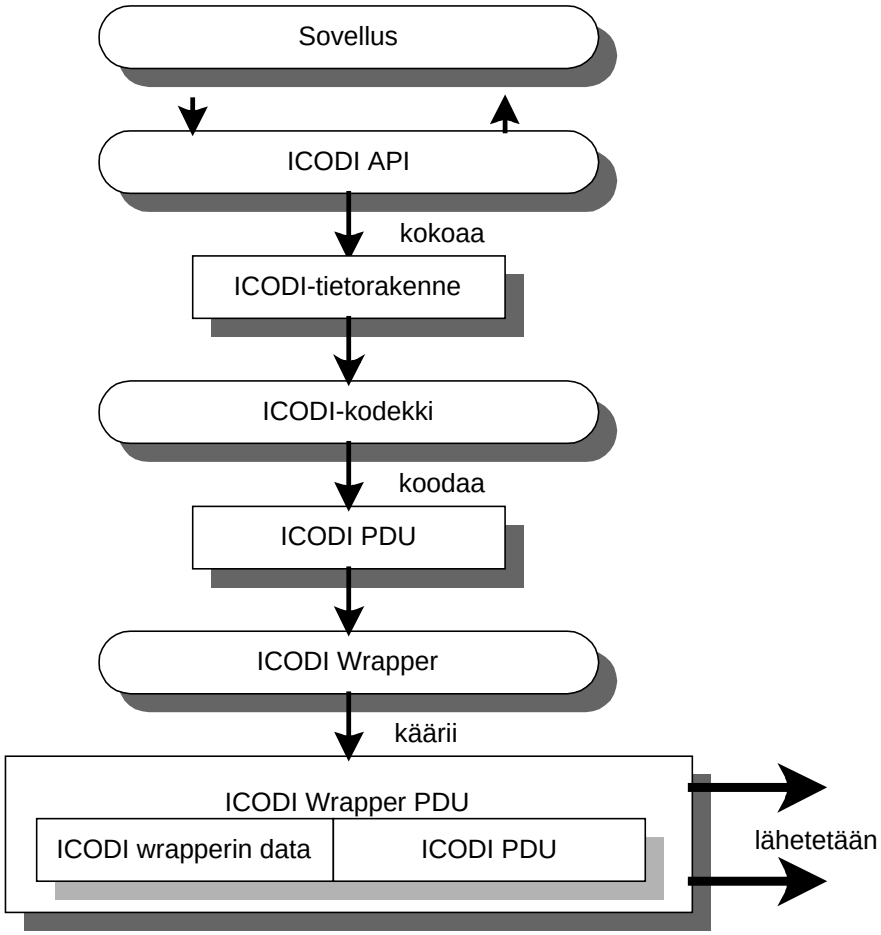
ICODI BE on ICODI-arkkitehtuurin tietokantariippuvainen osa. Se on C-kielellä toteutettu funktiokirjasto, jonka tarkoituksena on piilottaa tietokannan oma rajapinta ICODI FE:ltä. Tietokantarajapinta voi olla esimerkiksi Oraclen OCI (Oracle Call Interface). ICODI FE on siis tietokantaa käyttävälle sovellukselle näkyvä osa. ICODI FE:n kautta suoritettaville tehdään automaattinen vahvistus, eli mikäli operaation suoritus onnistui, vahvistaa ICODI FE operaation automaattisesti. Kuva 19 esittää ICODI FE ja BE -komponenttien sijainnin sovelluksessa, joka käyttää suoraa tietokantayhteyttä ICODI-kerroksen läpi.



Kuva 19. ICODI FE ja BE komponentit.

Varsinaiset palvelusovellukset eivät käytä suoraan ICODI FE:n API:a. Se on tarkoitettu järjestelmän sisäisten komponenttien käyttöön. Palvelusovellukset käyttävät Intellitel ONE tietokantaa ICODI API:n kautta. ICODI API on C-kielinen funktiokirjasto, joka tarjoaa valmiit rutiinit ICODI-operaatioiden muodostamiseen, lähettämiseen ja vastausten käsittelyyn. ICODI API muodostaa käyttäjäsovellukselta saamansa syötteen perusteella ICODI-tietorakenteen, josta muodostetaan ICODI-kodekin avulla varsinainen ICODI-sanoma eli ICODI PDU. ICODI-protokollan tarkoituksena on kommunikoida suoraan varsinaisen, esimerkiksi tietokannassa sijaitsevan, tietorakenteen kanssa. ICODI PDU ei siis sisällä siirtoon tai reititykseen liittyviä tietoja. [Int1998a]

ICODU PDU:n siirtoon käytetään erillistä ICODI Wrapper kuljetusprotokollaa. Se käärii ICODI PDU:n sisäänsä ja lisää lähetettävään dataan omat tietonsa, joita voidaan käyttää esimerkiksi reititykseen. Kuva 20 esittää ICODI-sanoman muodostuksessa ja lähetyksessä tarvittavat vaiheet ja tekijät.



Kuva 20. ICODI-sanoman muodostus ja lähetys.

5.3.2 Tietokanta-agentti

Tietokanta-agentti (DB agent) on komponentti, joka hoitaa yhteyden ja operaatiot varsinaisen tietokannan kanssa. Sen rakenne on kolmikerroksinen, varsinainen logiikka on toteutettu CVOPS työkalun avulla, mutta tietokantariippumattomuuden toteuttamiseksi se käyttää ICODI FE ja ICODI BE -kerroksia oman logiikkansa ja varsinaisen tietokanta-ajurin välissä.

Tietokanta-agentin toimintaperiaate on hyvin yksinkertainen. Se vastaanottaa ICODI-protokollalla kuvattuja ja ICODI Wrapper -protokollan avulla siirrettyjä operaatioita, suorittaa ne tietokantaan ja lähettää vastauksen. Tietokantaoperaatiot suoritetaan synkronisesti, eli tietokanta-agentti jää odottamaan tietokantaoperaation suoritusta. Tietokanta-agentti suorittaa operaatiot siis yksi kerrallaan, se on valmis suorittamaan uuden operaation vasta kun se on lähettänyt vastauksen edelliseen.

Synkronisesta operaatioiden suorituksesta aiheutuu kuitenkin ongelmia Intellitel ONE prosessienhallinnalle. Viestipohjaiseen prosessien tilan tarkkailuun kuuluu, että prosessi lähettää määräajoin viestin Watchdog-komponentille. Jos prosessi on jumissa odottaen tietokantakutsun palaamista, ei se luonnollisestikaan voi lähettää tarvittavaa viestiä. Tästä johtuen suuret operaatiot, joiden suoritus kestää kauan, ovat ongelma järjestelmän prosessienhallinnalle.

Ongelman ratkaisuun on kehitetty erillinen prosessienhallinnallinen viesti, jonka avulla tietokanta-agentti voi kertoa Watchdog-komponentille suorittavansa synkronista tietokantaoperaatiota, eikä siten voi lähettää tarkkailuviestiä. Viestissä prosessi kertoo arvion siitä, kuinka kauan operaation suoritus kestää.

5.3.3 Intellitel ONE scheduler

Paremman suorituskyvyn saavuttamiseksi järjestelmässä voi olla useampia tietokanta-agentteja, jotka suorittavan tietokantaoperaatioita rinnakkain. Usean tietokanta-agentin käyttö kuitenkin vaatii tahon, joka jakaa tehtävät tietokanta-agenttien kesken. Intellitel ONE järjestelmässä tämä komponentti on nimeltään Scheduler. ICODI operaatioiden jakamisen lisäksi sen tehtäviin kuuluu operaatiojonon hoitaminen, erillisen päivityslokin (update log) ylläpitäminen sekä mahdollinen datavirran kahdennus passiivisen puolen Schedulerin kanssa.

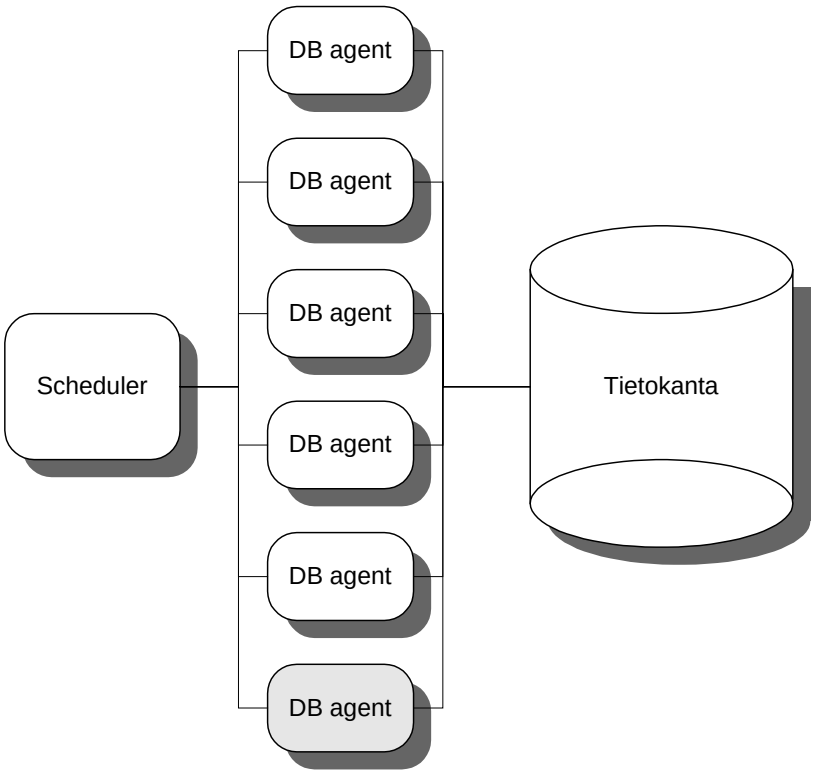
Päivitysloki on pysyvään muistiin tallennettu lokitiedosto tietokantaan tehdyistä muutoksista. Sen syntaksi on tietokantariippumaton ja sen avulla tietokannan sisältö

voidaan palauttaa haluttuun tilaan. Scheduler kirjoittaa kaikki tietokantaa muuttavat operaatiot ja niiden parametrit päivityslokiin. Käytännössä tällaisia operaatioita ovat:

- uuden rivin lisääminen tietokantaan,
- olemassa olevan rivin tietojen päivitys,
- rivin poistaminen tietokannasta ja
- käyttäjän toimesta tietokantaa päivittäväksi merkityt tietokantaproseduurikutsut.

Scheduler erottelee tietokanta-agentit päivittäviin ja lukeviin agentteihin ja ohjaa operaatiot oikeantyyppisille agenteille. Mikäli vapaata ja tehtävään sopivaa agenttia ei ole vapaana, niin Scheduler laittaa saapuneen operaation jonoon myöhempää käsittelyä varten. Päivityslokin ylläpidon ja mahdollisen kahdennuksen takia järjestelmässä ei käytännössä voi olla kuin yksi tietokantaa päivittävä tietokanta-agentti. Useita päivittäviä tietokanta-agentteja käytettäessä olisi vaarana, että päivitysloki ja oikeat tietokantatapahtumat tai järjestelmän eri puolten tietokannat eivät täsmää keskenään. Ongelma syntyisi, mikäli Scheduler kirjoittaisi päivityslokiin samaa riviä muokkaavien operaatioiden O1 ja O2 suoritusjärjestykseksi O1O2 mutta verkkoviiveiden ja/tai tietokanta-agenteissa tapahtuvien viiveiden takia oikea suoritusjärjestys olisikin O2O1. Ongelma vaikeutuisi entisestään, mikäli järjestelmää käytettäisiin kahdennetussa tilassa. Tällöin ongelmana ei olisi enää pelkästään päivitysloki vaan olisi mahdollista että operaatiot suoritettaisiin eri järjestyksessä järjestelmän passiivisella kuin aktiivisella puolella.

Edellä mainituista syistä johtuen järjestelmässä on normaalisti useita lukevia tietokanta-agentteja ja ainoastaan yksi päivittävä tietokanta-agentti. Kuva 21 esittää tietokanta-agenttien 5+1 peruskokoonpanon. Tämä tarkoittaa sitä, että järjestelmässä on 5 lukevaa ja 1 päivittävä tietokanta-agentti. Päivittävä tietokanta-agentti on merkitty kuvaan harmaalla taustavärillä.



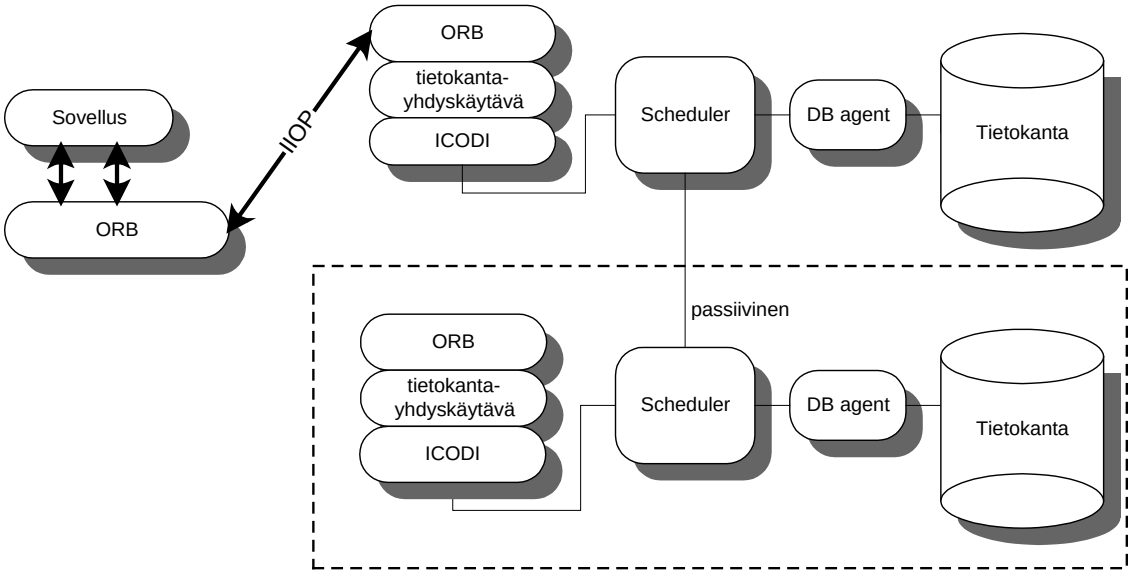
Kuva 21. Tietokanta-agenttien peruskokoonpano.

Scheduler komponentti käsittelee tietokantaoperaatioita ainoastaan ICODI-Wrapper tasolla. Se purkaa saapuvista viesteistä ICODI Wrapper:in tiedot ja tekee reitityspäätöksen näiden tietojen perusteella. Scheduler ei koske varsinaiseen ICODI PDU:hun, eikä sillä ole siihen tarvettakaan.

5.3.4 Tietokantayhdyskäytävä

Tietokantayhdyskäytävä on komponentti, jonka avulla ulkopuolinen järjestelmä voi hyödyntää Intellitel ONE –tietokantaa. Tietokantayhdyskäytävä tarjoaa IIOP-protokollan ja IDL-rajapinnan (Interface Definition Language), joiden avulla esimerkiksi CORBA-sovellus voi käyttää Intellitel ONE –tietokantaa. Tietokantayhdyskäytävä on teknisesti varsin yksinkertainen komponentti. Sen tehtävänä on kääntää IIOP-protokollan avulla saamansa operaatiot ICODI-operaatioiksi, jotka se lähettää Scheduler-komponentille käsiteltäväksi ja vastaavasti lähettää ICODI-vastaukset takaisin IIOP-protokollan avulla. Liiketoiminnan kannalta

tietokantayhdyskäytävän merkitys Intellitel ONE –järjestelmälle on suuri, sillä sen avulla Intellitel ONE voi toimia osana suurempaa järjestelmää. Kuva 22 esittää Intellitel ONE –järjestelmän tietokanta-arkkitehtuurisen yhdistämisen IIOP-protokollan avulla kolmannen osapuolen järjestelmään.



Kuva 22. Intellitel ONE -järjestelmän tietokantayhdyskäytävä.

6 TAPAHTUMANKÄSITTELY INTELLITEL ONE YMPÄRISTÖSSÄ

Intellitel ONE -järjestelmän tietokanta-arkkitehtuuri ei sisällä tapahtumien käsittelyyn tarvittavia komponentteja tai tekniikoita. Tapahtumia tarvitsevat ohjelmat on perinteisesti toteutettu tietokantaproseduurien avulla, eli ICODI-rajapinnan ja -protokollan läpi on kutsuttu tietokantaan tallennettua proseduuria ja välitetty tarvittavat parametrit proseduurille. Ratkaisu on kuitenkin ristiriidassa tietokantariippumattoman arkkitehtuurin kanssa, sillä tietokantaproseduurit ovat tietokantakohtaisia. Näin ollen tietokannan vaihtuminen tarkoittaisi sitä, että nämä proseduurit täytyisi kirjoittaa uudelleen uuden tietokannan tukemalla ohjelmointikielellä. Tietokantaproseduurien avulla ei voida myöskään toteuttaa tapahtumia, joiden pitäisi käsitellä dataa useammassa eri valmistajan tietokannassa.

6.1 Vaatimukset

Intellitel ONE -järjestelmän tulisi sisältää ominaisuudet, joiden avulla järjestelmässä voitaisiin suorittaa tapahtumia tehokkaasti ja luotettavasti. Ratkaisun tulisi toteuttaa ACID-vaatimukset ja sen tulisi mahdollistaa tietokantariippumaton tapahtumankäsittely. Kaiken taustalla on vaatimus helpottaa tapahtumankäsittelyä tarvitsevien sovellusten kehittämistä.

Ensisijaisena vaatimuksena suunnitelmalle on se, että tapahtumia voidaan suorittaa Intellitel ONE -järjestelmän sisällä, eli Intellitel ONE -järjestelmässä suoritettavat sovellukset voisivat hyödyntää järjestelmän tapahtumanhallintapalveluita omiin tarpeisiinsa. Toinen vaatimus kohdistuu Intellitel ONE –järjestelmän, ja tietokantayhdyskäytävän avulla siihen liitetyn ulkopuolisen järjestelmän muodostamaan laajempaan hajautettuun järjestelmään. Tämän tietokantayhteyden tehokas hyödyntäminen vaatii monissa tapauksissa sen, että Intellitel ONE voisi osallistua myös järjestelmässä suoritettaviin hajautettuihin tapahtumiin. Tällöin olisi mahdollista käyttää Intellitel ONE –tietokantaa osana hajautettua liiketoimintasovellusta ja Intellitel ONE –

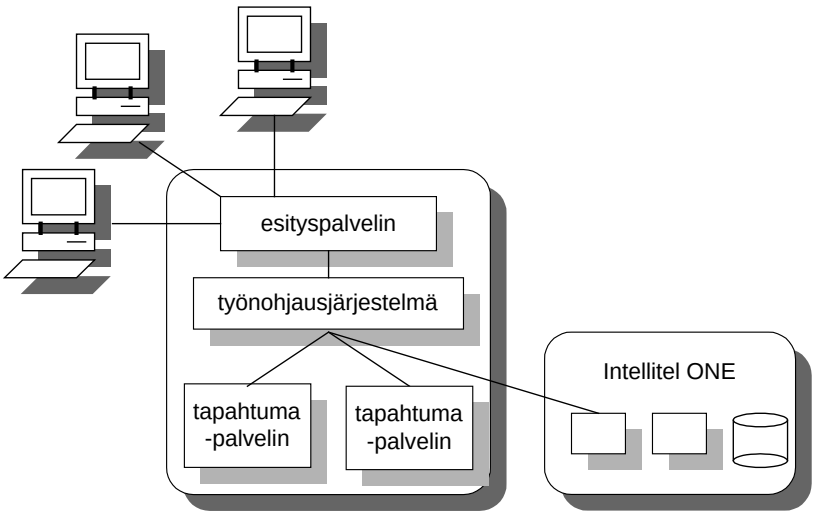
tietokanta toimisi yhtenä järjestelmän resurssina osallistuen kaksivaihevahvistukseen resurssinvalvojan ominaisuudessa.

Lisäksi suunnittelussa olisi hyvä ottaa huomioon olemassa olevien sovellusten ja komponenttien toiminta uudessa, tapahtumien käsittelyyn kykeneväisessä, ympäristössä. Yhteensopivuus vanhojen ratkaisujen kanssa helpottaa aina uuden tekniikan käyttöönottoa ja pienentää siihen kohdistuvia riskejä.

Intellitel ONE –järjestelmän tapahtumankäsittely asettaa joitakin vaatimuksia myös käytettäville resursseille. Käytettyjen resurssien tulee itsessään tukea tapahtumien suoritusta, eli niiden tulee täyttää ACID-vaatimukset. ICODI-komponenttien on tarkoitus toteuttaa ainoastaan tietokantariippumattomuus, joten ICODI ainoastaan siirtää resurssin ominaisuudet ja palvelut sitä hyödyntävän sovelluksen käyttöön. Intellitel ONE –tietokantajärjestelmään kuuluvien komponenttien tarkoituksena ei ole itse toteuttaa tapahtumakelpoisia resursseja vaan ainoastaan tarjota valmiiksi tapahtumia tukevien resurssien palvelut sovelluksien käyttöön tietokantariippumattoman rajapinnan avulla.

6.2 Yleisiä suunnitteluperiaatteita ja huomioita

Tapahtumien käsittelyyn vaadittujen ominaisuuksien toteutukseen on olemassa useita erilaisia vaihtoehtoja. Suunnittelun lähtökohtana voidaan pitää sitä, ettei Intellitel ONE –järjestelmän ole tarkoitus toteuttaa kaikkea tapahtumanhallintaohjelmiston ominaisuuksia ja siten kilpailla niitä vastaan. Tärkeämpää on saada järjestelmään resurssikohtainen tapahtumankäsittelytuki ja siten korostaa Intellitel ONE –tietokannan asemaa resurssinvalvojana, joka voisi mahdollisesti toimia joko oman tai ulkoisen tapahtumanvalvojan alaisuudessa. Ratkaisun olisi hyvä mahdollistaa Intellitel ONE –järjestelmän käyttö jonkun tapahtumanhallintaohjelmiston alaisuudessa. Käytännössä tämä on mahdollista esimerkiksi X/Open XA –rajapinnan ja IIOP-protokollan avulla. Kuva 23 esitää tapauksen, jossa Intellitel ONE –järjestelmää käytetään yhtenä tapahtumanhallintaohjelmiston tapahtumapalvelimena.



Kuva 23. Intellitel ONE osana tapahtumanhallintaohjelmistoa.

Yksi tärkeimmistä toteutuksen osa-alue on tapahtumien suorittamisen vaatiman logiikan toteutus. Logiikan toteuttamiseksi on erilaisia vaihtoehtoja ja niiden soveltuvuus riippuu pitkälti käyttötarkoituksesta. Suunnittelussa tulee ottaa huomioon se, että Intellitel ONE:a voitaisiin tietyissä tilanteissa käyttää osana suurempaa hajautettua järjestelmää, joten kaiken tapahtumien vaatiman logiikan toteuttaminen sisäisesti voi olla kokonaisuuden kannalta huono ratkaisu. Tällöin yhteistoiminta muiden järjestelmien kanssa voi vaikeutua huomattavasti.

Toteutuksen kannalta erilaisten vaihtoehtojen välillä voi olla hyvinkin suuria vaativuuseroja. Tämä täytyy ottaa huomioon myös suunnitteluvaiheessa, sillä teknisesti hienoin ja monimutkaisin suunnitelma ei ole tarkoituksenmukainen, mikäli sen toteuttaminen järkevässä ajassa ja saatavissa olevin resurssein ei ole mahdollista.

Edellä mainituista seikoista johtuen esimerkiksi tietokannoista tai tapahtumanhallintaohjelmistoista tuttu tulkattavalla kiellellä kuvattuihin tapahtumiin perustuva tapahtumankäsittely ei ole paras ratkaisu Intellitel ONE –järjestelmän tarpeisiin. Yhden resurssin suuntaan tulkattavalla kielellä olisi mahdollista toteuttaa toimiva tapahtumankäsittelylogiikka esimerkiksi Scheduler-komponentissa, mutta toteutus olisi työlästä ja suorituskyky olisi vahvasti riippuvainen tulkin tehokkuudesta.

Lisäksi tulee huomioida, että Intellitel ONE ei sisällä, eikä sen ole tarkoituskaan sisältää samanlaista työnohjausjärjestelmää kuin varsinaiset tapahtumanhallintaohjelmistot. Tästä johtuen Intellitel ONE -järjestelmän sisäisten hajautettujen tapahtumien toteuttaminen tulkattavalla kielellä olisi vaikeaa.

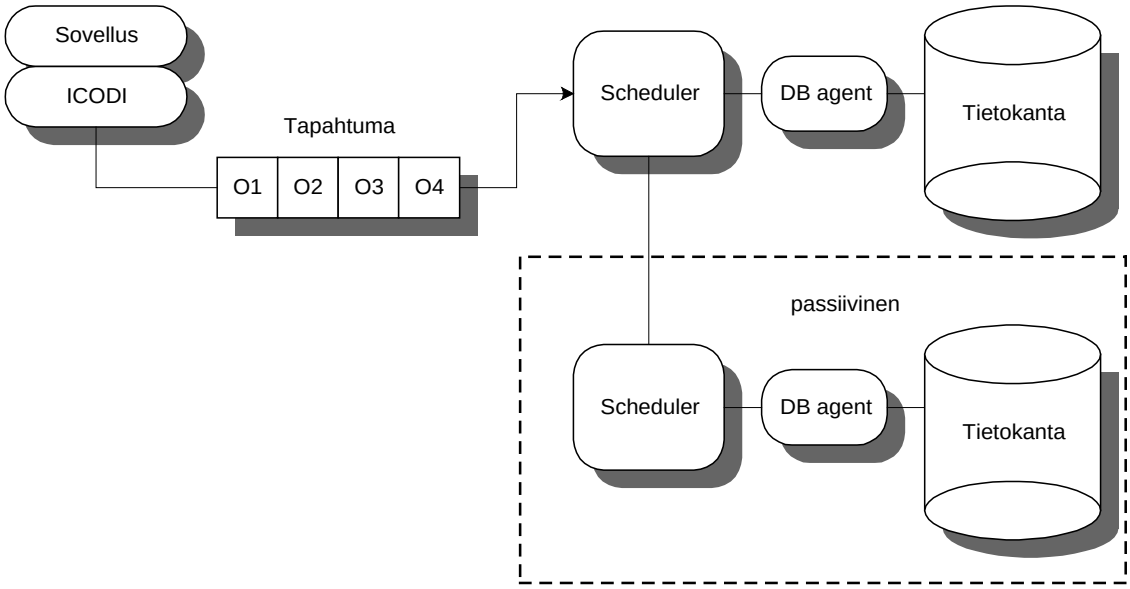
Esillä olleista syistä johtuen on järkevintä lähestyä ongelmaa resurssin näkökulmasta ja ajatella Intellitel ONE –tietokantajärjestelmää vain tietokanta- tai resurssiriippumattomana välikerroksena varsinaisen resurssin ja sitä käyttävän sovelluksen välillä. Intellitel ONE –tietokantajärjestelmän tulisi siis tarjota sellaiset tapahtumahallinnalliset työkalut ja rajapinnat, että niitä voisi hyödyntää niin sisäisessä ohjelmistokehityksessä kuin ulkoisten järjestelmien tapauksessakin.

Intellitel ONE –tietokantajärjestelmään ei ole järkevää ainakaan alkuvaiheessa toteuttaa omaa tapahtumanvalvojakomponenttia, koska sellaisen toteuttaminen veisi resursseja eikä sen tarpeellisuudesta voida vielä tehdä varmoja johtopäätöksiä. Järjestelmän sisäiset ja hajautettuja tapahtumia suorittavat sovellukset voivat itse toimia tapahtumanvalvojina omille tapahtumilleen. Tämä tietenkin vaatii sen, että sovellus suorittaa kaksivaihevahvistuksen resurssinvalvojina toimivien Scheduler-komponenttien kanssa. Intellitel ONE:een yhdistetyn ulkoisen järjestelmän sovellukset taas useimmiten käyttävät oman järjestelmänsä tapahtumanvalvojaa hajautettujen tapahtumien hallitsemiseen, joten tästäkin syystä erillisen tapahtumanvalvojakomponentin toteutusta voidaan helposti lykätä tulevaisuuteen.

Seuraavissa kappaleissa esitellään kolme vaihtoehtoista ratkaisumallia tapahtumankäsittelyn toteuttamiseksi Intellitel ONE -tietokantajärjestelmään, sekä kuvataan prototyyppitoteutus, jonka avulla yksinkertaisen ratkaisuvaihtoehdon soveltuvuutta voidaan arvioida. Ratkaisumallit esitetään ominaisuuksien kannalta kumulatiivisesti, eli esittely aloitetaan yksinkertaisesta mallista, jota myöhemmät mallit täydentävät uusilla ominaisuuksilla. Ratkaisumallit ovat hyvin erilaisia niin ominaisuuksiltaan kuin toteutuksen vaativuudeltaankin. Oikean ratkaisumallin valinta riippuu sekä käyttötarkoituksen sanelemista vaatimuksista että mallin jatkokehitykseen ja toteutukseen käytettävissä olevista resursseista.

6.3 Paikallinen resurssia päivittävä operaatiosarja

Yksinkertaisin malli on sellainen, jossa sovellus suorittaa tapahtuman yhtenä määrättynä joukkona resurssia päivittäviä operaatioita. Sovellus siis itse kokoaa operaatiot yhdeksi kokonaisuudeksi ja lähettää operaatiojoukon, eli tapahtuman, Scheduler-komponentin kautta tietokanta-agentille käsiteltäväksi. Tällöin sovelluksella ei ole varsinaista kontrollia tapahtuman suoritukseen. Tapahtuma alkaa silloin, kun tietokanta-agentti ryhtyy käsittelemään ensimmäistä tapahtumaan kuuluvaa operaatiota ja se loppuu siihen, kun viimeinenkin operaatio on käsitelty tai yksikin operaatioista on epäonnistunut. Tietokanta-agentti on vastuussa tapahtuman hoitamisesta resurssin eli tavallisimmin tietokannan suuntaan. Se aloittaa operaation ja tapauksesta riippuen joko vahvistaa tai peruu sen. Kuva 24 esittää yksinkertaisen tapahtumamallin, jossa tapahtuma koostuu joukosta operaatioita.



Kuva 24. Tapahtuman toteutus sarjana operaatioita.

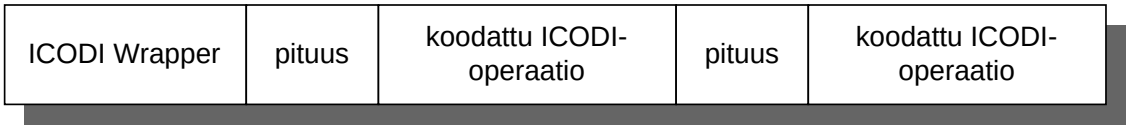
6.3.1 Prototyypin toteutus

Mallista päätettiin toteuttaa prototyyppi, jonka avulla niin mallin soveltumista reaalimaailman tarpeisiin kuin ongelmiakin voidaan paremmin arvioida. Prototyyppi

toteutus vaatii muutoksia lähes kaikkiin ICODI-komponentteihin sekä tietokanta-agentin logiikkaan. Scheduler-komponentti selviää vähillä muutoksilla, sillä sen ei tarvitse muuta kuin ohjata tapahtuman sisältävä ICODI Wrapper PDU resurssia päivittävälle tietokanta-agentille.

ICODI API:iin tehtiin muutoksia tapahtuman kokoamista ja vastauksen käsittelyä varten. Aiemmin ICODI API:lla on voinut koota kerrallaan vain yhden operaation ja sen tarvitseman datan, jotka on tallennettu globaaliin tietorakenteeseen. Sitten tämä tietorakenne on siirretty erillisellä funktiokutsulla CVOPS-rajapinnan kautta ICODI VTASK:lle käsiteltäväksi. Rakennetta muutettiin siten, että ICODI API:n avulla on mahdollista käsitellä useampaa operaatiota ennen viestin lähettämistä. Globaalista tietorakenteesta muodostuu siten lista, joka lähetyksen yhteydessä siirretään vanhaan tapaan ICODI VTASK:lle. Liitteessä 1 on kuvattu operaatiolistaa varten tarvitut muutokset ICODI-tietorakenteissa.

ICODI-kodekki perustuu XDR-kirjastoon (eXternal Data Representation), joka tarjoaa tavan esittää tietorakenteet laitteistoriippumattomalla tavalla ja on yleisesti käytössä esimerkiksi RPC-kutsuihin perustuvien tekniikoiden yhteydessä. ICODI-kodekki sisältää toimivaksi havaitut rutiinit yksittäisen ICODI-operaation koodaamiseen, joten prototyypin yhteydessä katsottiin parhaaksi käyttää mahdollisimman pitkälle vanhoja hyväksi havaittuja funktioita ja mekanismeja. ICODI-kodekkeihin toteutettiin tapahtumien koodausta varten funktio, joka koodaa jokaisen tapahtumaan kuuluvan operaation erikseen ja kokoaa yksittäisistä tavujonoista yhden kokonaisuuden. Jokaisen yksittäisen operaation edessä on sen viemä tila neljällä tavulla esitettynä. ICODI Wrapper:iin lisättiin parametri, joka kertoo tapahtumaan kuuluvien operaatioiden kokonaismäärän. Lisäksi ICODI:iin lisättiin uusi operaatiotyyppi tapahtumaa varten. ICODI Wrapper sisältää kentän, jossa operaatiotyyppi kerrotaan. Kuva 25 esittää prototyypissä käytetyn ICODI Wrapper PDU:n rakenteen.



Kuva 25. Prototyyppitoteutuksen ICODI Wrapper PDU.

Tietokanta-agentin käyttämää ICODI FE –rajapintaa muutettiin siten, että yhden operaation suorittamista ei seuraa automaattinen vahvistus. Muutos oli pakollinen, sillä tapahtumia käsiteltäessä on vahingollista vahvistaa yksittäiset operaatiot automaattisesti. Tämän takia rajapintaan lisättiin erilliset funktiot vahvista ja peruuta operaatioille. Tietokanta-agentin logiikka muutettiin vastaavasti, eli saadessa tapahtuman sisältämän ICODI Wrapper –paketin, tietokanta-agentti aloittaa uuden tapahtuman, suorittaa operaatiot yksi kerrallaan ja vasta kaikkien operaatioiden onnistuneen suorituksen jälkeen vahvistaa tapahtuman. Mikäli yksikin yksittäinen operaatio epäonnistuu, niin tietokanta-agentti peruu koko tapahtuman. Tietokanta-agentti lähettää tapahtuman suorittamisesta vastauksen sovellusohjelmalle. Vastaus sisältää koko tapahtuman tuloksen, sekä yksittäiset tulokset jokaisesta operaatiosta. Sovellus voi täten virheen tapauksessa saada selville operaation, joka aiheutti tapahtuman epäonnistumisen.

6.3.2 Havainnot ja ongelmat

Prototyypin avulla voitiin testata yksinkertaista tapahtumankäsittelymallia ja tehdä havaintoja sen perusteella. Prototyyppi itsessään osottautui toimivaksi ja se toteutti sille ennalta asetetut odotukset. Resurssia päivittäviin operaatioketjuihin saatiin prototyypin avulla tuki ACID-vaatimuksille.

Mallin etuihin kuuluvat yksinkertaisuus ja toteutuksen helppous. Tapahtuma voi Scheduler-komponentin kannalta katsottuna olla samanarvoinen operaatio kuin mikä tahansa muu tietokantaa päivittävä operaatio, eli Scheduler ainoastaan ohjaa tapahtuman tietokantaa päivittävällä tietokanta-agentille. Myös olemassa olevan kahdennuksen toteutus on varsin helppoa, sillä aktiivinen Scheduler voi ohjata tapahtumat passiiviselle

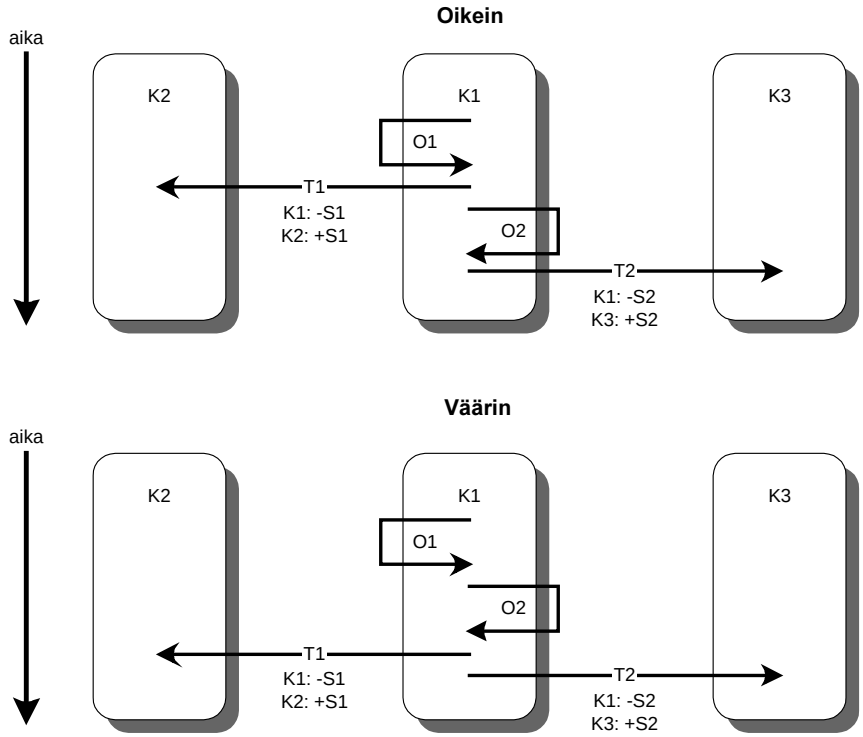
Schedulerille samaan tapaan, kuin se tekee yksittäisille resurssia päivittäville operaatioillekin.

Malli toteuttaa myös kaikki ACID-vaatimukset. Tapahtuman suoritus on yksi atomäärinen operaatio, sillä tietokanta-agentti huolehtii siitä, että mikäli yksikin operaatio epäonnistuu, koko tapahtuma perutaan. Yhtenäisyysvaatimuksen toteuttamiseksi tarvitaan yhteistyötä niin resurssin, kuin tapahtumaohjelmankin suunnasta. Monissa tapauksissa yhtenäisyysvaatimus voidaan toteuttaa resurssin tasolla estämällä eheyteen vaikuttavien seikkojen tuhoaminen tai muuttaminen. Eristyneisyysvaatimus toteutuu, sillä järjestelmässä on edelleen ainoastaan yksi päivittävä tietokanta-agentti ja se voi suorittaa kerralla ainoastaan yhtä tapahtumaa. Tästä syystä ei ole mahdollista, että tapahtuman suoritus vaikuttaisi millään tavalla toisiin tapahtumiin. Tapahtuman suorituksen voidaan olettaa täyttävän myös kestävyysvaatimuksen, sillä kun tietokanta-agentti on vahvistanut tapahtuman, siirtyy vastuu tapahtuman kestävyydestä varsinaiselle resurssille.

Prototyypin avulla havaittiin myös se, että mallissa on monia puutteita ja ongelmia. Mikäli osa tapahtuman syötteestä, tai pahimmassa tapauksessa koko syöte, koostuu resurssista saadusta datasta, soveltuu edellä esitetty yksinkertainen malli tapaukseen huonosti. Tilanne voidaan kyllä periaatteessa toteuttaa siten, että ensin luetaan resurssia yksittäisillä operaatioilla syötteen saamiseksi ja sitten koostetaan resurssia päivittävä tapahtuma luetun datan perusteella. Ongelmaksi muodostuu se, että tapahtumaan olennaisesti kuuluvaa datan lukemista ei suoriteta tapahtuman sisällä. Tällöin taas esimerkiksi tapahtumien välisten suoritusjärjestysten määrääminen vaikeutuu tai pahimmassa tapauksessa muuttuu mahdottomaksi.

Käytetään esimerkkinä rahan siirtämistä pankkitililtä toiselle. Muodostetaan kaksi tapahtumaa T1 ja T2. Tapahtuman T1 tarkoituksena on siirtää summa S1 käyttäjän K1 tililtä käyttäjän K2 tilille. Tapahtuma T2 taas siirtää summan S2 käyttäjän K1 tililtä käyttäjän K3 tilille. Käyttäjän K1 tililtä siirretään siis rahaa kahden muun käyttäjän tileille. Sillä kumpi tapahtumista suoritetaan ensin ja kumpi jälkimmäisenä ei ole

merkitystä. Molemmat tapahtumat koostuvat kahdesta operaatiosta. Ensin vähennetään käyttäjän K1 tiliä joko summan S1 tai summan S2 verran ja sitten lisätään vastaava summa joko käyttäjän K2 tai K3 tilille. Ensin summan vähentämistä pitää kuitenkin varmistaa, että käyttäjän K1 tilillä on riittävästi katetta. Koska käytössä oleva tapahtumien toteutukseen käytetty malli ei mahdollista lukevia operaatioita, joudutaan käyttäjän K1 tilin saldo lukemaan kahdella erillisellä operaatiolla O1 ja O2. O1 palauttaa saldon tapahtumaa T1 ja O2 tapahtumaa T2 varten. On selvää että lukuoperaatiot O1 ja O2 pitää suorittaa ennen vastaavia tapahtumia T1 ja T2, mutta tämä ei yksin riitä. Tarkasteltava malli ei sisällä tapahtumien ja operaatioiden välisen suoritusjärjestyksen määräämistä. Lukuoperaatioita O1 ja O2 ei kuitenkaan suoriteta samalla tietokanta-agentilla kuin tapahtumia T1 ja T2, joten on mahdollista, että operaatiot O1 ja O2 suoritetaan ennen kummankaan tapahtuman suorittamista ja ne molemmat palauttavat saman summan. Ensimmäisenä suoritettavaa tapahtumaa varten tehtävän lukuoperaation kannalta tämä ei ole ongelma, mutta jälkimmäinen palauttaa väärän summan, sillä oikea summa olisi se, joka käyttäjän K1 tilillä on kun ensimmäinen tapahtuma on jo suoritettu. Oikea suoritusjärjestys olisi siis joko O1T1O2T2 tai O2T2O1T1. Operaatioiden suoritusjärjestyksen merkitys on esitetty kuvassa 26. Jos lukuoperaatiot voitaisiin suorittaa tapahtumien sisällä, ei ongelmaa esiintyisi, sillä järjestelmä kykenee suorittamaan vain yhden tapahtuman kerrallaan. Järjestelmän kyky suorittaa vain yhtä tapahtumaa kerrallaan on kuitenkin sinälläänkin jo varsin vakava ongelma järjestelmän suorituskyvylle.



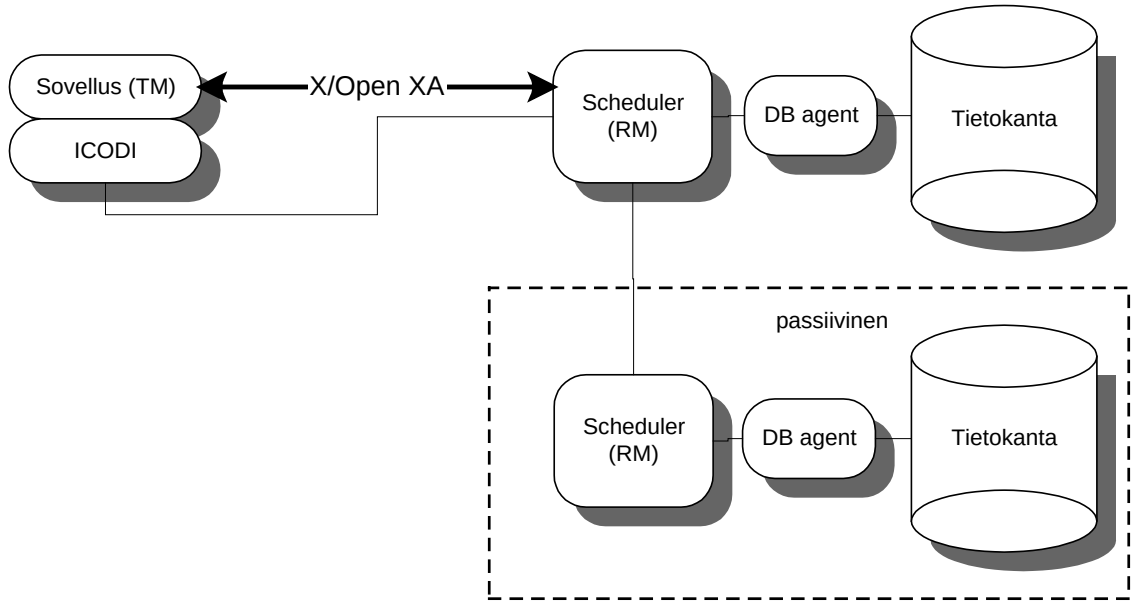
Kuva 26. Operaatioiden ja tapahtumien suoritusjärjestyksen merkitys.

Ratkaisumalli ei tue myöskään hajautettujen tapahtumien suoritusta, sillä tapahtuman kokoavalla sovelluksella ei ole kontrollia tapahtuman vahvistamiseen ja tapahtumaa varsinaisesti suorittava tietokanta-agentti ei sisällä kuin kaikki tai ei mitään -logiikan. Kaksivaihevahvistuksen toteuttaminen ei siis ole mahdollista tässä esitetyn ratkaisumallin yhteydessä.

Edellä esitetty malli sopii yksinkertaisiin ja paikallisiin tilanteisiin, joissa on tärkeää varmistaa vain se, että joukko resurssia päivittäviä operaatioita tulee suoritettua kaikki tai ei mitään -periaatetta noudattaen. Suoraviivaiset operaatiot kuten käyttäjän luominen järjestelmään tai tuotteen tietojen päivitys ovat esimerkkejä tapahtumista, joihin edellä kuvattu ratkaisumalli sopii hyvin.

6.4 Hajautuksen salliva ja kontrolloitava ratkaisumalli

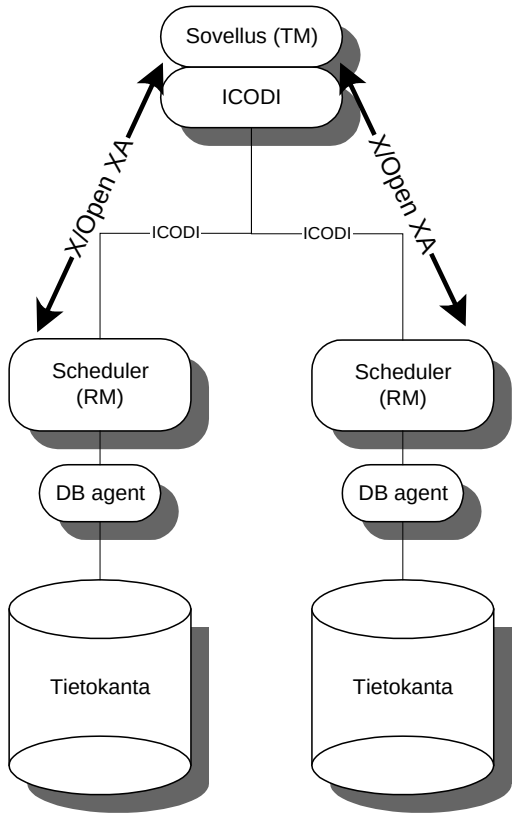
Toisessa ratkaisumallissa on lähdetty parantamaan ensimmäisen vaihtoehdon ongelmaa hajautettujen tapahtumien kanssa. Ongelmaksi edellisessä mallissa muodostui se, ettei tapahtumaa suorittavalla sovelluksella ollut varsinaista kontrollointimahdollisuutta tapahtuman kulkuun. Tällöin hajautettujen tapahtumien ja kaksivaihevahvistuksen toteutus käy mahdottomaksi. Tässä ratkaisuvaihtoehdossa tapahtumaa käyttävä sovellus kontrolloi tapahtuman suorittamista erillisen hallintarajapinnan kautta. Käytännössä tämä rajapinta on järkevintä toteuttaa X/Open XA –standardin mukaiseksi, sillä silloin taataan yhteensopivuus ulkoisten tapahtumanvalvojien ja tapahtumanhallintaohjelmistojen kanssa. Kuva 27 esittää järjestelmän, jossa sovellus ohjaa tapahtuman kulkua X/Open XA –rajapinnan avulla Intellitel ONE –järjestelmässä. Sovellus toimii kuvassa tapahtumanvalvojan roolissa ja Scheduler-komponentti vastaa resurssista, eli sitä voidaan pitää resurssinvalvojana.



Kuva 27. Sovelluksen kontrolloima tapahtuma Intellitel ONE -järjestelmässä.

Ratkaisu mahdollistaa hajautettujen tapahtumien suorittamisen joko yhden tai useamman eri sovelluksen kontrolloimana. Intellitel ONE –järjestelmässä suoritettavien sovellusten tapauksessa on järkevää ainakin aluksi rajoittaa tapahtumaa käyttävien

sovellusten määrä yhteen sillä muuten tarvittaisiin erillinen tapahtumanvalvoja komponentti, joka hallitsisi tapahtumien suoritusta. Yhden tapahtumaa kontrolloivan sovelluksen tapauksessa sovellus voi itse toimia tapahtumanvalvojana. Kuva 28 esittää tapauksen, jossa Intellitel ONE –järjestelmää käyttävä sovellus suorittaa hajautetun tapahtuman kahteen eri tietokantaan. Kuvan yksinkertaistamiseksi siinä ei ole esitetty kahdennettujen järjestelmien passiivisia puolia.



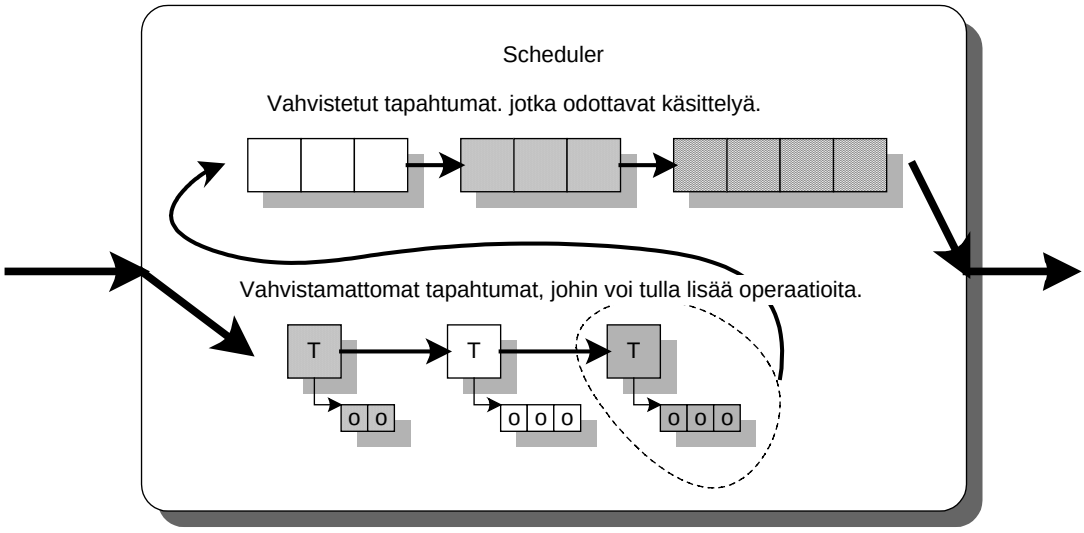
Kuva 28. Hajautettu tapahtuma Intellitel ONE -järjestelmässä.

Ulkopuolista tapahtumanhallintaohjelmistoa käytettäessä voidaan luonnollisesti käyttää sen tarjoamaa tapahtumanvalvojaa, jolloin Intellitel ONE –tietokantajärjestelmän ja etenkin Scheduler-komponentin rooliksi jää toimia yhtenä järjestelmän resurssinvalvojista.

6.4.1 Komponenttien vaatimat muutokset

Toinen ratkaisumalli aiheuttaa paljon enemmän muutoksia järjestelmän komponentteihin kuin ensimmäinen. Tässä mallissa on lähdetty siitä periaatteesta, että tietokanta-agentin toiminta on samanlainen kuin ensimmäisen ratkaisuvaihtoehdon tapauksessa. Se siis suorittaa vain kokonaisia tapahtumaketjuja kaikki tai ei mitään -periaatetta noudattaen. Sovelluksen lähettämät yksittäiset ICODI-operaatiot kootaan Scheduler-komponentissa tapahtumiksi ja valmiit tapahtumat lähetetään tietokanta-agentille käsiteltäväksi sitten, kun sovellus on vahvistanut tapahtuman. Scheduler-komponentin tulee lisäksi käsitellä tapahtumanvalvojana toimivalta sovellukselta tuleva operaatio, joka määrää resurssinvalvojan valmistautumaan tapahtuman vahvistamiseen. Viimeistään siinä vaiheessa tapahtuma tulee kaksivaihevahvistuksen sääntöjä noudattaen tallettaa pysyvään muistiin.

Scheduler-komponenttiin pitäisi siis nykyisen viestijonon lisäksi toteuttaa kaksi uutta jonoa. Toisessa jonossa olisi keskeneräiset tapahtumat ja niihin liittyvät operaatiot ja toisessa vahvistetut operaatiot, jotka odottaisivat tietokanta-agentille pääsyä. Molempien jonojen tulisi virheidenhallinnallisista syistä olla pysyviä, eli jonot tulisi tallentaa kiintolevylle tai johonkin muuhun pysyvään muistiin. Kuva 29 esittää Scheduler-komponenttiin tarvittavat viestijonot tapahtumien käsittelyä varten.



Kuva 29. Scheduler-komponentin tapahtumajonot.

6.4.2 Havainnot ja ongelmat

Ratkaisumalliin jää edelleen samoja ongelmia, joita käsiteltiin jo ensimmäisen ratkaisumallin yhteydessä. Tapahtumat voivat edelleen koostua ainoastaan resurssia päivittävistä operaatioista, koska tapahtumaan kuuluvista operaatioista ei ole mahdollista saada välituloksia ennen tapahtuman vahvistamista. Tästä syystä ratkaisumalli rajoittaa edelleen sovelluksen logiikkaa tilanteissa, joissa tapahtumaan tarvittaisiin syötettä resurssista käsin. Malli ei myöskään ratkaise järjestelmän suorituskykyongelmia, sillä järjestelmässä voi edelleenkin olla ainoastaan yksi tapahtumia ja päivittäviä operaatioita suorittava tietokanta-agentti.

6.5 Resurssin ominaisuuksien hyödyntäminen ICODI-kerroksen läpi

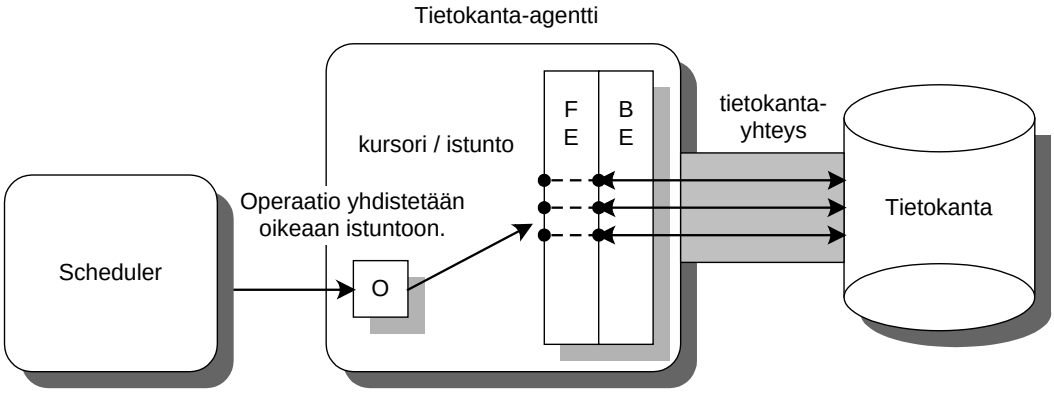
Kolmannessa ratkaisumallissa on tarkoituksena ratkaista toisen mallin jättämiä ongelmia sekä suorituskyvyn että tapahtumalogiikan rajoittuneisuuden suhteen. Tavoitteena tässä mallissa on viedä tapahtumissa suoritettavien operaatioiden käsittely ICODI-kerroksen läpi suoraan varsinaiselle resurssille ilman, että operaatioita niputetaan yhdeksi kokonaisuudeksi matkan varrella. Tällöin jokaisesta tapahtumaan kuuluvasta operaatiosta voidaan saada välitulos, jota voidaan hyödyntää jatkossa tapahtuman logiikassa. Ratkaisumalli sallii siis myös lukuoperaatioiden suorittamisen tapahtuman sisällä.

Ratkaisumalli mahdollistaa monimutkaisenkin logiikan toteuttamiseen tapahtumiin ja logiikka voidaan kuvata samalla ohjelmointikielellä, kuin millä varsinainen ohjelma toteutetaan. Sovellus voi tapahtuman aloittamisen jälkeen suorittaa samaan tapahtumaan liittyen niin monta operaatiota kuin se haluaa ja operaatiot voivat olla minkä tyyppisiä tahansa.

6.5.1 Komponenttien vaatimat muutokset

Tapahtumien sisältämät operaatiot suoritetaan varsinaisessa resurssissa sitä mukaa, kun sovellus suorittaa niitä ICODI-operaatioina. Tällöin jokaista ICODI-kerroksella olevaa tapahtumaa varten tietokanta-agentin ja resurssin välille tarvitaan resurssikohtainen tapahtuma, joita pitäisi voida suorittaa useampia yhtäaikaisesti. Esimerkiksi tietokannan tapauksessa tämä vaatisi yhden tietokantakursorin tai –istunnon avaamista jokaista ICODI-tapahtumaa varten.

Scheduler-komponentin vastuulle jäisi nykyisenkaltainen operaatiojonon ylläpito, eli tietokanta-agentti suorittaisi edelleenkin vain yhden operaation kerrallaan. Muutos olisi kuitenkin siinä, että tietokanta-agentti voisi suorittaa operaatioita useampaan eri tapahtumaan ennen kuin yhtäkään niistä vahvistetaan. Käytännössä tietokantakursoreiden tai –istuntojen määrä voi olla rajattu, joten Scheduler-komponentin vastuulle jäisi tämän rajoituksen noudattaminen. Scheduler-komponentin pitäisi tietää mitä tapahtumia tietokanta-agentilla on kesken ja montako uutta tapahtumaa se voi vielä avata ilman, että yhtäkään kesken olevista tapahtumista vahvistetaan. Mikäli Scheduler saa pyynnön aloittaa uusi tapahtuma, mutta tietokanta-agentti ei voi käsitellä useampaa yhtäaikaista tapahtumaa, niin Scheduler-komponentin täytyy laittaa tapahtumapyyntö jonoon ja käsitellä se kun tietokanta-agentin resursseja vapautuu. Tällöin sovellus ei saa kuittausta tapahtuman hyväksymisestä ennen kuin tapahtuma on tietokanta-agentin käsiteltävänä ja siihen voidaan suorittaa operaatioita. Kuva 30 esittää kuinka tietokanta-agentti yhdistää käsiteltäväksi tulevan operaation oikean tapahtuman käytössä olevaan tietokantakursoriin tai –istuntoon.



Kuva 30. Yhtäaikaisten tapahtumien käsittely tietokanta-agentissa.

6.5.2 Havainnot ja ongelmat

Lukuoperaatioiden suorittaminen tapahtuman sisällä ja siten myös tapahtumaa hoitavan tietokanta-agentin kautta lisää entisestään suorituskykyongelmia yhden resurssia päivittävän tietokanta-agentin tapauksessa. Tästä syystä olisi tärkeää parantaa järjestelmää siten, että se sallisi usean päivittävän tietokanta-agentin rinnakkaisen käytön. Tällöin voitaisiin luopua lukeva/päivittävä tietokanta-agentti jaosta ja kaikki tietokanta-agentit voisivat käsitellä kaikentyyppisiä operaatioita.

Ongelma rinnakkaisista päivittävistä tietokanta-agenteista on johtunut lähinnä päivityslokista eli on olemassa vaara siitä, että päivityslokissa operaatioiden järjestys on erilainen kuin niiden oikea suoritusjärjestys. Tämä ongelma voidaan ratkaista siten, että vaikka järjestelmässä suoritetaan yhtäaikaisesti useita tapahtumia eri tietokanta-agenteilla, käsitellään vahvistukset yksi kerrallaan. Mekanismi aiheuttaa muutoksia päivityslokin toteutukseen, jolloin päivityslokista tulee oikeastaan tapahtumaloki. Muutosten jälkeen vahvistusjärjestys on ainoa tekijä, jolla on merkitystä tapahtumalokin ja varsinaisen resurssin välisen koherenssin kannalta.

Kahdennetussa järjestelmässä edellä kuvattu malli ei ole yksistään riittävä. Operaatiot voidaan verkon viiveistä johtuen suorittaa eri järjestyksessä järjestelmän aktiivisella ja passiivisella puolella ja vaikka vahvistukset käsiteltäisiinkin yksi kerrallaan, niin voi olla, että järjestelmän passiivisella puolella tapahtuma ei esimerkiksi lukkotilanteiden

vuoksi ole voitu suorittaa vaikka se aktiivisella puolella olisi valmis vahvistettavaksi. Tämä ongelma voidaan ratkaista Scheduler-komponenttien välisen kaksivaihevahvistuksen avulla, eli joko molemmat puolet vahvistavat muutoksensa tai sitten muutokset perutaan molemmilta puolilta ja sovellukselle palautetaan virhe tapahtuneesta.

Eristyneisyys vaatimuksen toteuttaminen on vaikeaa järjestelmässä, jossa voidaan suorittaa useita päivittäviä operaatioita yhtäaikaisesti. Intellitel ONE – tietokantajärjestelmän tapauksessa tilanne osoittautuu erityisen ongelmalliseksi, sillä järjestelmän tulee toteuttaa eristyneisyys useiden erilaisten resurssien tapauksessa. Tietokantapalvelimet toteuttavat omat eristyneisyysmekanisminsa, kuten esimerkiksi sarjoittamisen, mutta niiden toiminta ja toteutus vaihtelevat tuotteiden välillä. Tietokantapalvelimet tukevat yleensä SQL-standardeissa määriteltyjä erilaisia eristyneisyystasoja, jotka määräävät tapahtumien eristyneisyyden asteen. SQL92-standardi määrittää seuraavat eristyneisyysasteet:

- vahvistamattoman tiedon luku (read uncommitted),
- vahvistetun tiedon luku (read committed),
- toistettava luku (repeatable read) ja
- sarjoitettava.

[Ora2000]

Taulukko 1 esittää SQL92-standardin eristyneisyysasteiden ja mahdollisesti ongelmia aiheuttavien tapahtumien yhteisvaikutuksien riippuvuudet:

Taulukko 1. SQL92-standardin eristyneisyysasteet [Ora2000].

Eristyneisyysaste	Likainen luku (1)	Ei-toistettava luku (2)	Haamuluku (3)
vahvistamattoman tiedon luku	Mahdollinen	Mahdollinen	Mahdollinen
vahvistetun tiedon luku	Ei mahdollinen	Mahdollinen	Mahdollinen
toistettava luku	Ei mahdollinen	Ei mahdollinen	Mahdollinen
Sarjoitettava	Ei mahdollinen	Ei mahdollinen	Ei mahdollinen

1. Likainen luku (dirty read) tarkoittaa sitä, että tapahtuma voi lukea toisen tapahtuman muuttamaa vahvistamatonta dataa.
2. Ei-toistettava luku (non-repeatable read) tarkoittaa sitä, että tapahtuma lukee resurssin tiedon uudelleen ja näkee toisten tapahtumien sillä välin muuttaman vahvistetun tiedon.
3. Haamuluku (phantom read) tarkoittaa sitä, että jos tapahtuma suorittaa saman kyselyn uudelleen, voi se nähdä toisten tapahtumien sillä välin lisäämät uudet rivit.

Resurssien käyttämien erilaisten eristyneisyysmekanismien lisäksi myös pienemmissä toteutukseen liittyvissä yksityiskohdissa on eroja. Esimerkiksi Oracle-tietokantapalvelin ei käytä ollenkaan lukulukkoja. Tästä johtuen on mahdollista, että toinen tapahtuma pääsee muuttamaan tietoa resurssissa, jota toinen tapahtuma parhaillaan lukee. Tästä johtuen tapahtumat, jotka suorittavat resurssin eheyteen liittyviä toimenpiteitä sovellustasolla eivät voi luottaa edes tapahtumat sarjoittavaa eristyneisyysastetta käytettäessä siihen, ettei tapahtuman lukema data muutu tapahtuman aikana. Tätä voidaan kontrolloida sovellustasolla asettamalla resurssiin lukkoja sovelluksesta käsin. [Ora2000] Lukituksen kontrollointi olisi siis tärkeää saada ICODI-kerroksen läpi ICODI API:a käyttävien sovellusten hyödynnettäväksi. Tämä tarkoittaa käytännössä uusia ICODI-operaatioita, jotka voivat olla tapahtuman sisäisiä operaatioita siinä missä kaikki muutkin operaatiot.

7 JOHTOPÄÄTÖKSET

Tapahtumien luotettava ja tehokas käsittely on ehdottoman tärkeä tekniikka varsinkin liiketoimintasovelluksista puhuttaessa. Teknologian kehitys ja järjestelmille asetetut tehokkuus-, hallinta- ja tietoturvavaatimukset ovat viimeisten vuosien aikana lisänneet hajautettujen järjestelmien tarpeellisuutta merkittävästi. Hajautetut ympäristöt ovat yleistyessään aiheuttaneet monia ongelmia perinteisille tapahtumankäsittelymekanismeille. Näiden ongelmien ratkaisu on kuitenkin ensiarvoisen tärkeää, sillä tapahtumankäsittelyn merkitys ainoastaan korostuu hajautetuissa järjestelmissä. Tapahtumankäsittelyn voidaankin katsoa olevan tekniikka, jonka avulla hajautetun tietojenkäsittelyn luotettavuus voidaan varmistaa.

Hajautetuilla järjestelmillä pyritään perinteisiin järjestelmiin verrattuna saavuttamaan parempaa suorituskykyä, luotettavuutta, tietoturvaa ja järjestelmän hallintaa. Hajautettujen tapahtumien kontrollointi on kuitenkin varsin raskasta, joten hyvään suorituskykyyn pyrkivä järjestelmä tulisi suunnitella siten, että hajautettuja tapahtumia käytetään mahdollisimman vähän. Hajautetut tapahtumat ovat myös herkempiä virheille ja vaativat enemmän toimenpiteitä virheistä palautumiseen kuin paikalliset tapahtumat. Näistä ongelmista huolimatta on tilanteita, joissa hajautettujen tapahtumien käyttö on perusteltua ja järkevää. Usein se voi olla myös ainoa vaihtoehto tietyn ongelman ratkaisemiseksi.

Hajautettu tapahtumankäsittely on laaja alue, johon liittyy paljon erilaisia tekniikoita ja standardeja. Erilaisilla standardeilla on kuitenkin hyvin pitkälle yhteiset perinteet ja niitä kaikkia yhdistää se seikka, että ne ovat perineet hyvin paljon ideoita ja menetelmiä valmiista, olemassa olevista sovelluksista. Tähän on syynä se, että monet valmistajat ovat ehtineet kehittää omat tekniikkansa paljon ennen avointen standardien syntyä. Tämä on ehkä suurin syy siihen, että hajautettuun tapahtumankäsittelyyn ei ole olemassa avoimia ja yleisesti hyväksyttyjä eli niin kutsuttuja ”*de facto*” -standardeja. Lähimmäksi tätä asemaa on päässyt X/Open DTP –standardiperhe ja varsinkin siihen kuuluva XA-standardi, joka määrittelee rajapinnat ja komponentit tapahtumien hajautettuun kaksivaihevahvistukseen.

Uudemmissa oliopohjaisissa ympäristöissä kuten CORBA- ja J2EE-standardien mukaisissa arkkitehtuureissa asiat ovat tapahtumankäsittelyn suhteen huomattavasti paremmin. Molemmat standardit määrittävät arkkitehtuurin, jossa myös hajautettuun tapahtumankäsittelyyn on kiinnitetty huomiota. J2EE:n JTS-standardi on lisäksi periytynyt suoraan CORBA-arkkitehtuurin OTS-standardista, joten niiden väliseen yhteistoimintaan on myös kiinnitetty huomiota. Molemmat standardit tukevat myös X/Open XA –standardin mukaisia resurssinvalvojia.

Oliopohjaisten CORBA- ja J2EE-ympäristöjen yleistyminen ja niiden tarjoamat tapahtumanhallintapalvelut ovat vähentäneet erillisten tapahtumanhallintaohjelmistojen tarvetta ja tämä suuntaus tulee jatkumaan myös lähitulevaisuudessa. Myös tapahtumanhallintaohjelmistoja kehittävät yritykset ovat huomanneet tämän ja nykyiset kaupalliset J2EE-sovelluspalvelimet tarjoavat usein J2EE-tapahtumankäsittelyn lisäksi rajapinnat myös yrityksen omiin perinteisiin tapahtumanhallintaohjelmistoihin. Toisaalta myös tapahtumanhallintaohjelmistot ovat omaksuneet uusia ominaisuuksia ja tekniikoita ja lähentyneet siten uudempia oliopohjaisia järjestelmiä. Näillä toimenpiteillä pyritään takaamaan yhteensopivuus perinteisten ja uusien järjestelmien välillä. On kuitenkin selvää, että perinteiset tapahtumanhallintaohjelmistot tulevat jatkossa keskittymään entistä enemmän raskaiden ja hajautettujen sovellusten käyttöön, sillä kevyempään käyttöön tietokantapalvelimet ja J2EE- ja CORBA-ympäristöjen ominaisuudet riittävät mainiosti.

CORBA- ja J2EE-standardeihin perustuvat ympäristöt eivät kuitenkaan hyvistä ominaisuuksistaan huolimatta ainakaan lähitulevaisuudessa uhkaa tapahtumanhallintaohjelmistojen asemaa suurten yritysten raskaiden liiketoimintaohjelmistojen perustana. CORBA- ja J2EE-sovellukset ovat hitaita verrattuna laitteisto-optimoituihin tapahtumanhallintaohjelmistoihin, joten vastaavien käyttäjämäärien palveleminen ei nykyisillä CORBA- ja J2EE-sovelluksilla ole mahdollista. Perinteiset tapahtumanhallintaohjelmistot ovat olleet markkinoilla jo kauan ja ne ovat ajan mittaan saavuttaneet erittäin hyvän luotettavuuden ja suorituskyvyn, joten tästäkään syystä niiden korvaaminen uudemmilla tekniikoilla ei ole ajankohtaista.

Tästä johtuen voidaankin todeta, että raskaat tapahtumanhallintaohjelmistot ja CORBA- ja J2EE-ympäristöt kilpailevat pääosin erilaisilla markkinoilla.

Web Services on suhteellisen uusi ja nopeasti kehittyvä teknologia, jonka tulevaisuus näyttää varsin valoisalta. Web Services –tapahtumankäsittelyn standardointityö on vielä kesken mutta on varmaa, että tapahtumankäsittelyllä tulee olemaan keskeinen rooli, mikäli Web Services –tekniikoita tullaan hyödyntämään esimerkiksi eri valmistajien liiketoimintasovellusten yhdistämiseen.

Työn yhtenä tarkoituksena on ollut selvittää ja suunnitella, kuinka Intellitel ONE –järjestelmään saataisiin toteutettua tietokantariippumaton tuki tapahtumankäsittelylle. Ratkaisun olisi hyvä mahdollistaa sekä hajautetut tapahtumat että yhteistoiminta muiden järjestelmien ja niiden tapahtumankäsittelyratkaisujen kanssa. Suunnitelmassa tuli kuitenkin muistaa myös se seikka, että sen pitäisi olla myös toteutettavissa saatavissa olevin resurssein. Varsinaiset tapahtumanhallintaohjelmistot ovat kompleksisia järjestelmiä, eikä Intellitel ONE –järjestelmän kehittäminen niiden kaltaiseksi ole perusteltua. Suunnittelussa onkin otettu näkökulma, jossa korostetaan Intellitel ONE -tietokantajärjestelmän asemaa standardien mukaisena resurssina, jota voidaan hyödyntää niin sisäisissä kuin ulkoisten järjestelmien kontrolloimissa tapahtumissa.

Intellitel ONE –tietokantajärjestelmä perusajatus on ollut resurssiriippumattomuus. Tapahtumankäsittely Intellitel ONE –järjestelmässä on kuitenkin esitettyjen mallien valossa enemmän tai vähemmän riippuvainen resurssin kyvystä käsitellä tapahtumia. Resurssin tulee siis itsenäisesti toteuttaa tuki paikallisille tapahtumille ja tarjota keinot tapahtumien vahvistamiseen ja perumiseen. Tämän lisäksi kolmannessa mallissa esitelty tapa kuvata yhtäaikaiset tapahtumat resurssikohtaisilla kursoreilla tai istunnoilla vaatii tuen käytettävältä resurssilta.

Resurssin tapa hoitaa lukitukset ja tiedon eristäminen tapahtumien kesken vaikuttaa suoraan myös Intellitel ONE –tietokantajärjestelmän tapahtumankäsittelyyn. Intellitel ONE –tietokantajärjestelmä toteuttaa ainoastaan tietokantariippumattoman välikerroksen, joten kaikki resurssin tekniseen toimintaan liittyvät säädöt tulee tehdä

suoraan resurssin tarjoamin keinoin. Tämäkin seikka voi heikentää resurssiriippumatonta perusajatusta, sillä esimerkiksi kahdennetun järjestelmän tapauksessa on tärkeää, että järjestelmän molempien puolten tietokannat tai resurssit käyttävät samanlaista eristyneisyysmekanismia ja toimivat kaikin tavoin mahdollisimman identtisesti. Muuten on vaarana, että järjestelmän aktiivinen ja passiivinen puoli ajautuvat ajan kuluessa epäyhtenäiseen tilaan. Tämä on kuitenkin tiedostettu ongelma ja se voidaan ratkaista sopivilla resurssivalinnoilla ja suunnittelulla.

LÄHTEET

[Bac1992] Bacon, Jean. Concurrent systems: an integrated approach to operating systems, database and distributed systems. USA: Addison-Wesley Publishers Ltd, 1992. 602 s. ISBN 0-201-41677-8.

[Ber1997] Bernstein, Philip A, & Newcomer, Eric. Principles of Transaction Processing. San Francisco: Morgan Kaufmann Publishers, Inc. 1997, 358 s. ISBN 1-55860-415-4.

[Bjö1991] Björn, Kari. A Failure Model of A Distributed Transaction Processing System. Espoo, Otaniemi: TKK monistamo, 1991. Digitaalitekniikan laboratorion tutkimusraportti, 78 s. ISBN: 951-22-0901-2.

[Blo1992] Bloomer, John. Power Programming with RPC. USA: O'Reilly & Associates, Inc. 1992. 459 s. ISBN: 0-937175-77-3.

[By1995] De By, Rolf et al. A reference architecture for cooperative transaction processing systems. Espoo: Valtion teknillinen tutkimuskeskus VTT, 1995. VTT tiedotteita n:o 1694, 102 s. ISBN: 951-38-4849-3.

[Cel2003a] CVOPS overview [WWW-dokumentti], 2003 [viitattu 7.3.2003]. Celtius Oy. Saatavissa: http://www.celtius.com/cpfpage_704.htm

[Cel2003b] Background of CVOPS [WWW-dokumentti], 2003 [viitattu 7.3.2003]. Celtius Oy. Saatavissa: http://www.celtius.com/cpfpage_1092.htm

[Cmu1997] Carnegie Mellon University, Software Engineering Institute. Transaction Processing Monitor Technology [WWW-dokumentti], 1997 [viitattu 11.3.2003]. Saatavissa: http://www.sei.cmu.edu/str/descriptions/tpmt_body.html

[Gra1993] Gray, Jim & Reuter, Andreas. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers, Inc. 1993, 1002 s. ISBN 1-55860-190-2.

[Har1991] Harju, Jarmo et al. OSI Transaction Processing standard. An introduction and outlines of implementation. Espoo: Valtion teknillinen tutkimuskeskus VTT, 1991. VTT tiedotteita n:o 1250, 44 s. ISBN: 951-38-3882-X.

[Ibm2003a] CICS – Product overview [WWW-tuotedokumentti], 2003 [viitattu 10.3.2003]. IBM Software. Saatavissa: http://www-3.ibm.com/software/ts/cics/

[Ibm2003b] IMS Overview [WWW-tuotedokumentti], 2003 [viitattu 10.3.2003]. IBM Software. Saatavissa:http://www-3.ibm.com/software/data/ims/presentations/two/imsoverview/index.htm

[Ibm2003c] TXSeries – Product overview [WWW-tuotedokumentti], 2003 [viitattu 14.3.2003]. IBM Software. Saatavissa: http://www-3.ibm.com/software/ts/txseries/

[Int1998a] ICODI arkkitehtuurikuvaus, 1998. Intellitel Communications Oy.

[Int1998b] Intellitel ONE 3.0 User Manual, 2002. 365 s. Intellitel Communications Oy.

[Ope2002] Standards Information Base [WWW-dokumentti], 2002 [viitattu 12.2.2003]. The Open Group. Saatavissa: http://www.opengroup.org/sib.htm

[Ope2003] About The Open Group [WWW-dokumentti], 2003 [viitattu 11.2.2003]. The Open Group. Saatavissa: http://www.opengroup.org/overview/index.htm

[Ora2000] Oracle Corporation. Oracle8i/Application Developer's Guide – Fundamentals, 2000.

[Orf1996] Orfali, Robert et al. The Essential Distributed Objects Survival Guide. USA: John Wiley & Sons, Inc. 1996, 640 s. ISBN 0-471-12993-3.

[Ric1997] Ricciuti, Mike. IBM's single package untangles middleware [WWW-dokumentti], 1997 [viitattu 14.3.2003]. Saatavissa: http://news.com.com/2100-1001-204487.html?tag=bplst

[Rom1999] Roman, Ed. Mastering Enterprise JavaBeans and the Java2 Platform, Enterprise Edition.USA: John Wiley & Sons, Inc. 1999. 709s. ISBN: 0-471-33229-1.

[Sie2000] Siegel, Jon. Corba 3 Fundamentals and Programming second edition. USA: John Wiley & Sons, Inc. 2000. 928 s. ISBN 0-471-29518-3.

[Sun2002] Sun Educational Services. Developing J2EE Compliant Applications - Student Guide, 2002.

[VTT1999] Valtion teknillinen tutkimuskeskus VTT. CVOPS User s Guide for CVOPS, 6.3, 1999.

[W3C2003] Web Services Activity [WWW-dokumentti], 2003 [viitattu 19.3.2003]. World Wide Web Consortium. Saatavissa: http://www.w3c.org/2002/ws

[Web2002a] Web Services Coordination [WWW-dokumentti], 2002 [viitattu 19.3.2003]. Bea Systems, IBM, Microsoft. Saatavissa: http://www-106.ibm.com/developerworks/library/ws-coor/ http://dev2dev.bea.com/techtracks/ws-coordination.jsp http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-coordination.asp

[Web2002b] Web Services Transaction [WWW-dokumentti], 2002 [viitattu 19.3.2003]. Bea Systems, IBM, Microsoft. Saatavissa: http://www-106.ibm.com/developerworks/library/ws-transpec http://dev2dev.bea.com/techtracks/ws-transaction.jsp http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-transaction.asp

LIITE 1. ICODI-tietorakenteen muutokset prototyyppitoteutuksessa

Perinteinen ICODI-tietorakenne:

```
struct ICODI {
        int type;
        union {
                Response *response;
                GetRequest *getRequest;
                SetRequest *setRequest;
                CreateRequest *createRequest;
                DeleteRequest *deleteRequest;
                ActionRequest *actionRequest;
                FreeSQLRequest *freeSQLRequest;
        } ICODIStruct;
};
typedef struct ICODI ICODI;
```

ICODI-tietorakenne tapahtumien käsittelyä varten:

```
/* Transaction data structure */
typedef struct icodi_transaction
{
    ICODI *op;
    struct icodi_transaction *next;
}
ICODITransaction;
```