LAPPEENRANNAN TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto

UUDET OLIOTEKNOLOGIAT ÄLYVERKKOTOTEUTUKSESSA

Diplomityön aihe on hyväksytty Lappeenrannan teknillisen korkeakoulun tietotekniikan osaston osastoneuvostossa 7.10.1998.

Työn tarkastaja: dosentti, FT Olli Martikainen
Työn ohjaaja: DI Panu Puro

Pasi Nummisalo
Vanha Nurmijärventie 139
01700 Vantaa
p. 0400-300 778

TIIVISTELMÄ

Tekijä: Nummisalo, Pasi Antero

Tutkielman nimi: Uudet olioteknologiat älyverkkototeutuksessa

Osasto: Tietotekniikan osasto

Vuosi: 1999

Paikka: Lappeenranta

Diplomityö. Lappeenrannan teknillinen korkeakoulu.

61 sivua, 15 kuvaa, 2 liitettä.

Tarkastajana dosentti, FT Olli Martikainen.

Avainsanat: CORBA, IN, Java, komponentti, sovelluskehys, TOVE

Keywords: CORBA, IN, Java, component, framework, TOVE

Olioteknologiat ja -menetelmät ovat tuoneet merkittävää helpotusta alati kasvavien tietoliikenneohjelmistojen kehitykseen. Ohjelmistojen tiukkenevat vaatimukset, monitasoinen rakenne, laadunvarmistus ja aikataulujen kiristyminen ovat vauhdittaneet ohjelmakehityksen siirtymistä kohti olioteknologioiden käyttöä.

Työn tavoitteena on esitellä älyverkkototeutukseen sopivia olioteknologioita ja niistä syntyviä käytännön etuja. Työn teoriaosassa käydään läpi älyverkkoarkkitehtuuria ja älyverkkototeutuksiin soveltuvia uusia olioteknologioita. Näistä merkittävimmiksi nousevat komponentit, sovelluskehykset, CORBA-standardi (Common Object Request Broker) ja Java-kieli.

Käytännön osuus koostuu teoriaosassa esiteltyä olioteknologiaa käyttävästä älyverkon palvelunohjauspisteestä, palvelunhallintapisteestä ja niihin liittyvien ratkaisujen esittelystä. Tehty toteutus ei ole täydellinen esimerkki älyverkon palvelunohjaus- ja hallintapisteen oikeaoppisesta toteutuksesta, vaan pikemminkin esimerkki uuden olioteknologian tuomista mahdollisuuksista ja eduista. Suurimpia etuja ovat komponenttien tuoma ongelma-alueiden ja koodin kapselointi, komponenttituotantoa ohjaavien sovelluskehysten avustava vaikutus, Javan mahdollistama koodin todellinen siirrettävyys ja CORBA:n tarjoama olioiden helppo hajautus.

ABSTRACT

Author: Nummisalo, Pasi Antero

Title: New object technologies in intelligent network implementation

Department: Department of Computer Science

Year: 1999

Place: Lappeenranta

Master of Science Thesis. Lappeenranta University of Technology.

61 pages, 15 pictures, and 2 appendices.

Supervisor: docent, PhD Olli Martikainen

Keywords: CORBA, IN, Java, component, framework, TOVE

Object technologies and object-oriented methods have improved the implementation of large-scale telecommunication software. Higher user requirements, multilayer architecture, quality control and shorter time to market requirements have driven software development to use object technologies.

The aim of this work is to present object technologies suitable for intelligent network solutions and to explain the object advantage in a practical solution. The theory part of this work describes architecture of the intelligent network and new object technologies applicable to the intelligent network solutions. The most remarkable ones of these new object technologies are components, the CORBA (Common Object Request Broker) architecture and the Java language.

The practical part introduces an intelligent network service control and management point using the new object technology. The practical part is not a full-featured example of building the service control point and the service management point, but an example of using object technology in a new context. The biggest advantages of object technologies in this context are problem and code capsulation with components, frameworks helping component production, platform neutrality of the Java language and an easy object distribution with the CORBA technology.

ALKUSANAT

Diplomityö on tehty Teknillisessä korkeakoulussa tietoliikenneohjelmistojen ja multimedian laboratoriossa. Työ liittyi laajempaan TOVE-projektiin.

Esitän parhaat kiitokset professori Olli Martikaiselle tietoliikennetekniikan innostavasta opetuksesta ja huimista visioista. Lisäksi haluan kiittää kaikkia työn arviointiin osallistuneita henkilöitä.

Suurimmat kiitokset työ valmistumisesta kuuluvat perheelleni. Ilman heidän uhrautuvaa ja pitkäjänteistä tukea en olisi nyt tässä. Suuri halaus myös rakkaimmalleni, Virpille, kiitos että jaksat.

SISÄLLYSLUETTELO

1 JOHDANTO 6
2 ÄLYVERKKO 8
2.1 Käsitteellinen malli 8
2.2 Palvelutaso 9
2.3 Yleinen toiminnallinen taso 9
2.3.1 Palveluriippumaton rakennuspala (SIB) 10
2.3.2 Yleinen palvelulogiikka (GSL) 11
2.3.3 Peruspuheluprosessi (BCP) 11
2.4 Hajautettu toiminnallinen taso 11
2.4.1 Puhelumalli 13
2.4.2 Havaintopisteiden käsittely 13
2.4.3 CCF/SSF/SCF -toiminta 14
2.5 Fyysinen taso 16
3 MERKINANTOVERKKO 19
3.1 Sanomansiirto-osa 19
3.2 Merkinantoyhteyden ohjausosa 20
3.3 Tapahtumankäsittelyosa 20
3.4 INAP 21
4 UUDET OLIOTEKNOLOGIAT 23
4.1 Olio-ohjelmointi 23
4.2 Suunnittelumallit 23
4.3 Sovelluskehykset 24
4.4 Komponentit 25
4.5 CORBA 26
4.5.1 Rajapinnan kuvauskieli 27
4.5.2 ORB 27
4.5.3 Oliopalvelut 28
4.5.4 Alakohtaiset rajapinnat ja yleiset palvelut 28

4.5.5 CORBA 3.0 28
5 JAVA 30
5.1 Java-kieli 30
5.2 Java-alusta 32
5.3 JavaBeans 34
5.4 RMI 36
5.5 Enterprise JavaBeans 36
5.5.1 Roolit 38
5.5.2 Toiminta 40
6 ÄLYVERKKOTOTEUTUS 41
6.1 TOVE-projekti 41
6.2 Tavoite ja menetelmät 41
6.3 B-SSP 42
6.4 DCF-sovelluskehys 43
6.4.1 DCF-ympäristö 43
6.4.2 Komponentit ja viesti 44
6.4.3 Käyttöliittymä 44
6.5 Palvelunhallinta- ja palvelunohjauspiste 45
6.5.1 Palveluesimerkki 46
6.5.2 INAP-rajapinta 48
6.5.3 IDL ja INAP 50
6.6 Mahdollisuudet ja jatkokehitys 50
7 TULOKSET 52
7.1 Kokemukset ja suositukset 52
8 JOHTOPÄÄTÖKSET 54

LYHENTEET

AC	Application Context
API	Application Programming Interface
ASE	Application Service Element
ASN.1	Abstract Syntax Notation no. 1
AWT	Abstract Windowing Toolkit
BCP	Basic Call Process
BCSM	Basic Call State Model
BISUP	Broadband ISDN User Part
CCAF	Call Control Agent Function
CCF	Call Control Function
CID	Call Instance Data
CMIP	Common Management Interface Protocol
CORBA	Common Object Request Broker Architecture
CS	Capability Set
DCF	Distributed Component Framework
DCOM	Distributed Component Object Model
DFP	Distributed Functional Plane
DII	Dynamic Invocation Interface
DP	Detection Point
DPC	Destination Point Code
EDP	Event Detection Point
EJB	Enterprise JavaBeans
FE	Functional Entity
FEAM	Functional Entity Manager
FSM	Finite State Machine
GFP	Global Functional Plane
GSL	Global Service Logic
GT	Global Title

HLSIB	Higher Level SIB
IDL	Interface Definition Language
IF	Information Flow
IIOP	Internet Inter-ORB Protocol
IN	Intelligent Network
INCM	IN Conceptual Model
IP	Intelligent Peripheral
ITU-T	International Telecommunications Union
JDK	Java Development Kit
JFC	Java Foundation Classes
JIT	Just In Time
JTA	Java Transaction API
JNDI	Java Naming and Directory Interface
MACF	Multiple Association Control Element
MTP	Message Transfer Part
MVC	Model View Control
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnection
OVOPS	Object-oriented Virtual Operations System
PIC	Point In Call
POI	Point Of Initiation
POR	Point of Return
RMI	Remote Method Interface
ROSE	Remote Operation Service Element
RPC	Remote Procedure Call
SACF	Single Association Control Function
SCCP	Signaling Connection Control Part
SCEF	Service Creation Environment Function
SCEP	Service Creation Environment Point

SCF	Service Control Function
SCME	SCF Management Entity
SCP	Service Control Point
SDF	Service Data Function
SDP	Service Data Point
SF	Service Feature
SIB	Service Independent Building Block
SLP	Service Logic Program
SLPI	Service Logic Program Instance
SLPL	Service Logic Program Library
SMAF	Service Management Access Function
SMF	Service Management Function
SMP	Service Management Point
SN	Service Node
SRF	Specialized Resource Function
SS7	Signaling System no. 7
SSCP	Service Switching and Control Point
SSD	Service Support Data
SSF	Service Switching Function
SSME	SSF Management Entity
SSN	Subsystem Number
SSP	Service Switching Point
TCAP	Transaction Capabilities Application Part
TDP	Trigger Detection Point
TOVE	Transparent Object oriented Virtual Exchange, Toimialaverkot
UML	Unified Modeling Language
UNI	User Network Interface

1 JOHDANTO

Tietoliikenteen alalla tapahtuu juuri nyt suuria muutoksia. Tekninen kehitys, niin tietotekniikassa kuin tietoliikennetekniikassakin, on ottanut suuria harppauksia ja vauhti näyttää vain kiihtyvän. Ohjelmistojen osuus kasvaa kiihtyvällä tahdilla ja uusia ohjelmistometodeja etsitään ja otetaan käyttöön. Myös tietotekniikan ja tietoliikennetekniikan konvergenssi näyttää päivä päivältä todennäköisemmältä.

Kansallisten puhelinmonopolien purkaminen ja langattoman tiedonsiirron huima lisääntyminen on muuttanut tietoliikennekenttää ympäri maailmaa avaten markkinoita todelliselle kilpailulle. Markkinoiden laajentumisen ja avautumisen seurauksena on ilmestynyt uusia avoimia standardeja ajavia yrityksiä. Kilpailun kiristyessä kukaan ei voi enää linnoittautua omien epäyhteensopivien tekniikoiden ja palveluiden taakse. Markkinat yhdentyvät globaalissa mielessä, eikä maiden rajat enää rajoita esim. palveluiden tarjoajien markkinoita. /3/

Myös asiakkaiden vaatimukset ovat kasvaneet. He eivät enää tyydy tarjottuihin puhelinpalveluihin vaan vaativat aktiivisesti heidän omia tarpeitaan vastaavia palveluita. Tässä muuttuvassa tilanteessa yhdeksi tärkeimmistä valteista on noussut uusien palveluiden nopea saattaminen tuotantokäyttöön. Tässä kohdassa astuu kuvaan älyverkkoarkkitehtuuri ja älyverkkototeutuksissa käytettävät uudet olioteknologiat. Näiden pitäisi olla osaltaan vastaus edellä esitettyihin kasvaviin vaatimuksiin.

Tämän työn tavoitteena on tutkia uusien olioteknologioiden soveltuvuutta älyverkkototeutuksiin. Työ valottaa oliolähestymistavan huonoja ja hyviä puolia tarjoten uusia näkökantoja liittyen palvelujen luontiin ja ajamiseen. Teoriaosan ensimmäinen puolisko esittelee älyverkkoa olemassa olevien

standardien pohjalta. Toinen puolisko esittelee merkittävimmät uudet oliomenetelmät ja -teknologiat, joihin kuuluvat komponentit, sovelluskehykset, CORBA-standardi ja Java-kieli.

Käytännönosuudessa kuvataan Javalla tehty komponenttipohjainen sovelluskehys ja sen avulla toteutetut älyverkon palvelunhallintapiste ja palvelunohjauspiste. Toteutukset perustuvat soveltuvilta osin älyverkkostandardointiin, ja niissä on käytetty hyväksi teoriaosassa esiteltyjä olioteknologioita ja niihin liittyviä standardeja.

2 ÄLYVERKKO

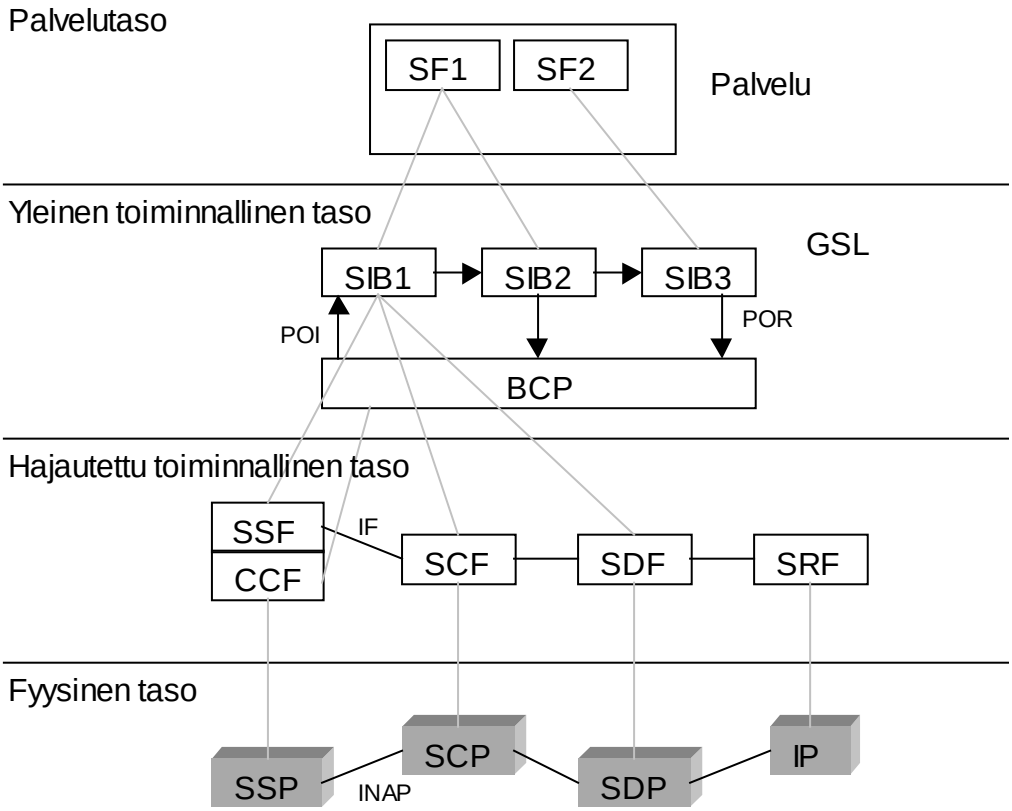
Älyverkko (IN, Intelligent Network) on tietoliikenneverkon palveluarkkitehtuurin malli. Sen tarkoituksena on tarjota avoin, hajautettu ja palveluriippumaton alusta lisäarvopalveluiden tarjoamiseen puhelinverkossa. Älyverkkomalliin kuuluu myös palvelun luontiin ja hallintaan liittyvät asiat. /4/

Älyverkon suunnittelulähtökohtia ovat olleet mahdollisuus erilaisten palveluiden tarjoamiseen fyysisen tason arkkitehtuurista riippumatta ja palveluiden toteuttaminen käyttäen palveluriippumattomia ja yleisiä komponentteja. Älyverkkomallin kuvaaman alustan tulisi pystyä palvelemaan suuria käyttäjämääriä ja tarjoamaan monipuolisia palveluita eri palvelutarjoajien toteuttamina.

Yksi eniten älyverkon standardointiin vaikuttavista standardointielimistä on ITU-T (International Telecommunications Union). Älyverkon standardointityö on aloitettu 1989, ja se tapahtuu iteratiivisesti aikajaksoissa tuottaen yhteensopivia määrittelyjoukkoja (CS, Capability Set). Jokainen määrittelyjoukko määrittelee tuettavat palvelut ja palveluominaisuudet. Suositukset on koottu Q.1200-sarjaksi, joissa viimeinen numero määrittelee suosituksen aiheen. Suosituksen kolmannesta numerosta voidaan päätellä suosituksen määrittelyjoukko . /5/

2.1 Käsitteellinen malli

Älyverkon käsitteellinen malli (INCM, IN Conceptual Model) on viitekehys, jolla kuvataan älyverkon rakennetta. Käsitteellinen malli on jaettu neljään eri tasoon, joista jokainen käsittelee älyverkkoa erilaisesta käsitteellisestä näkökulmasta. Kaksi alinta kerrosta määrittelevät arkkitehtuurin ja ylimmät kerrokset liittyvät arkkitehtuurista riippumattomiin palveluihin. /3/



Kuva 1. Älyverkon käsitteellinen malli.

2.2 Palvelutaso

Palvelutaso (service plane) määrittelee sen, miltä palvelut (service) näyttävät niiden käyttäjien kannalta. Palvelut on kuvattu palveluominaisuuksien (SF, Service Feature) avulla. Palvelukuvaus on sanallinen kuvaus siitä, miten palvelu toimii tietyssä tilanteessa. Palveluominaisuuksien avulla palvelun tilaajalle voidaan esittää halutun palvelun eri kombinaatioita ja mahdollisuuksia. /3/

2.3 Yleinen toiminnallinen taso

Yleinen toiminnallinen taso (GFP, Global Functional Plane) kuvaa palvelut niiden suunnittelijoiden kannalta käsitellen älyverkkoa yhtenä kokonaisuutena. Peruspuheluprosessi (BCP, Basic Call Process) kuvaa puhelun tapahtumia ja

niistä aiheutuvia viestejä (POI, Point Of Initiation), sekä puhelunkäsittelyn kulkuun vaikuttavia viestejä (POR, Point of Return). Palvelutason palveluominaisuudet muodostuvat palveluriippumattomista rakennuspalasista (SIB, Service Independent Building Block). Yleinen palvelulogiikka (GSL, Global Service Logic) määrittelee sen, miten palveluominaisuuksissa tarvittavat SIB:it on ketjutettu yhteen. /3/

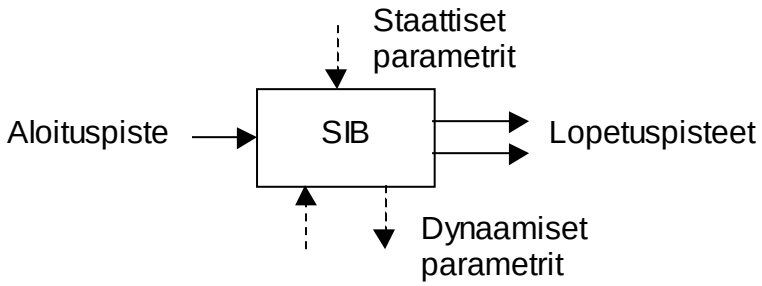
2.3.1 Palveluriippumaton rakennuspala (SIB)

Palveluominaisuudet kuvataan yhdellä tai useammalla SIB-primitiivillä.
SIB:in ominaispiirteitä ovat:

- SIB on uudelleenkäytettävä rakenneosa, joka kuvaa yhden täydellisen toiminnon.
- Sillä on tarkasti määritelty rajapinta, johon kuuluu yksi aloituspiste ja yksi tai useampia lopetuspisteitä.
- SIB on riippumaton palvelusta ja fyysisestä verkkoarkkitehtuurista.
- SIB:it on kuvattu standardilla tavalla.

SIB:in parametrit:

- Dynaamiset parametrit (CID, Call Instance Data), jotka vaihtuvat puhelusta toiseen, esim. soittajan puhelinnumero (A-numero).
- Staattiset parametrit (SSD, Service Support Data) riippuvat palveluominaisuudesta, johon SIB on liitetty.



Kuva 2. SIB:in rajapinnat.

2.3.2 Yleinen palvelulogiikka (GSL)

SIB-primitiiveistä ketjuttamalla koottu yleinen palvelulogiikka määrittelee SIB-primitiivien järjestyksen. Tämän SIB-ketjun tulee olla liittyneenä BCP-SIB:iin. Ennen tätä palvelulogiikalla ei juurikaan ole käyttöä, koska se ei pysty toimimaan puhelumallin kanssa.

2.3.3 Peruspuheluprosessi (BCP)

Peruspuheluprosessi kuvaa abstraktilla tasolla puhelumallia ja siitä mahdollisesti aiheutuvia viestejä. Viestit kuvataan pisteillä, joihin haluttu palvelulogiikka voidaan liittää. Näin palvelulogiikalla pystytään vaikuttamaan meneillä olevaan puhelunmuodostukseen. Palvelulogiikan viimeinen SIB-primitiivi liitetään myöskin BCP-SIB:iin. Näin palvelulogiikalle annettu kontrolli palaa takaisin puhelumalliin, aiheuttaen siellä esim. haluttuun tilaan siirtymisen tai palvelulogiikan antamien tietojen käyttöönoton.

2.4 Hajautettu toiminnallinen taso

Hajautettu toiminnallinen taso (DFP, Distributed Functional Plane) jakautuu toiminnallisiin kokonaisuuksiin (FE, Functional Entity) ja niiden vuorovaikutussuhteisiin (IF, Information Flow). Taso on tarkoitettu kuvaamaan älyverkkoa verkon suunnittelijoiden ja verkkopalveluiden tarjoajien kannalta. Toiminnalliset kokonaisuudet voidaan jakaa palvelun suoritustoimintoihin sekä palvelun luonti- ja hallintatoimintoihin. Palvelun suoritustoimintoja ovat (kuva 5):

- Puhelunohjausagenttitoiminto (CCAF, Call Control Agent Function) mahdollistaa käyttäjien liittymisen verkkoon. CCAF toimii rajapintana käyttäjän ja verkon puhelunohjaustoiminnon välillä.
- Puhelunohjaustoiminto (CCF, Call Control Function) huolehtii puhelun käsittelystä ja ohjauksesta tarjoten mm. liipaisintoiminnallisuuden (trigger functionality) älyverkkopalveluiden käynnistämiseksi.

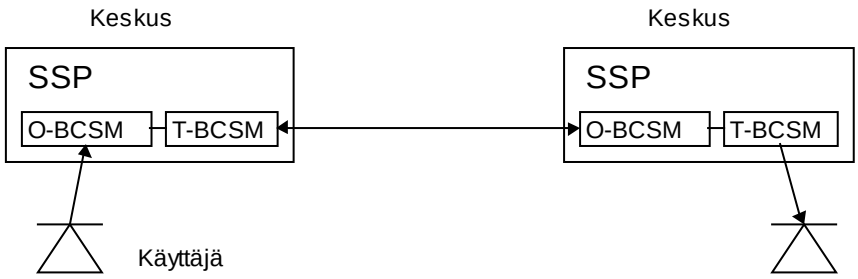
- Palvelunkytkentätoiminto (SSF, Service Switching Function) tarjoaa toiminnot puhelun- ja palvelunohjauksen väliseen yhteistoimintaan. SSF laajentaa CCF:n liipaisintoimintaa tarkastamalla mm. liipaisimien laukaisuehdot. SSF hallitsee myös CCF:n ja SCF:n välistä merkinantoa ja vaikuttaa CCF:n toimintaan SCF:ssä sijaitsevien palveluiden käskyjen mukaan.
- Palvelunohjaustoiminto (SCF, Service Control Function) sisältää älyverkon palvelulogiikan, jonka mukaan hallitaan älyverkkopuhelua. SSF:n lisäksi SCF voi olla yhteydessä myös muihin toiminnallisiin kokonaisuuksiin, jotka voivat sisältää tarvittavan lisälogiikan (SRF) tai palvelun tarvitseman tiedon (SDF).
- Palvelutietokantatoiminto (SDF, Service Data Function) sisältää reaaliaikaisen asiakas- ja palvelutietokannan.
- Erikoisresurssitoiminto (SRF, Specialized Resource Function) sisältää älyverkkopalvelun vaatimat käyttäjän vuorovaikutukseen tarvittavat toiminnot, kuten käyttäjän näppäilyiden tulkkaus, äänitiedotukset, puheentunnistus ja merkkiäänet. SRF on yhteydessä SCF:n ja SSF:n (ja CCF:n) kanssa.

Palvelun luonti- ja hallintatoiminnot (kuva 5):

- Palvelunhallintatoiminto (SMF, Service Management Function) ohjaa palvelun hankintaa, levittämistä ja hallintaa.
- Palvelunhallinnanpääsytoiminto (SMAF, Service Management Access Function) määrittelee rajapinnan SMF:ään.
- Palvelunluontiympäristötoiminto (SCEF, Service Creation Environment Function) mahdollistaa älyverkkopalveluiden määrittelyn, toteutuksen, testauksen ja siirron SMF:ään. Tuloksena saadaan palvelulogiikan kuvaus ja siihen liittyvät tietokuvaukset.

2.4.1 Puhelumalli

CCF:n peruspuhelumallia (BCSM, Basic Call State Model) kuvataan kahdella erillisellä mallilla, joita kutsutaan tulevaksi (originating) ja lähteväksi (terminating) puhelumalliksi. Mallit kuvaavat kahta eri tilakonetta (FSM, Finite



State Machine). Toinen on tuleville puheluille ja toinen lähteville puheluille, keskuksen kannalta katsottuna (kuva 3). Puhelumalli koostuu tiloista (PIC, Point In Call) ja havaintopisteistä (DP, Detection Point). Havaintopisteistä puhelumallin kontrolli voidaan siirtää palvelulogiikalle.

2.4.2 Havaintopisteiden käsittely

Jokainen havaintopiste voi olla aktiivinen tai passiivinen. Vain aktiiviset havaintopisteet voivat aiheuttaa sanoman lähetyksen palvelulogiikalle. Havaintopisteet voidaan asettaa aktiiviseksi staattisesti tai dynaamisesti. Staattinen aktivointi tapahtuu SMF:n toimesta, ja havaintopiste pysyy aktiivisena kunnes se passivoidaan jälleen SMF:n toimesta. Dynaaminen

Kuva 3. Puhelumallien väliset suhteet.

aktivointi tapahtuu taas SCF:n toimesta, ja se on voimassa kunnes aktivoitu piste on ohitettu tai SCF:n ja SSF:n välinen vuorovaikutussuhde loppuu. Staattisesti aktivoitu havaintopiste on nimeltään TDP (Trigger Detection Point) ja dynaamisesti aktivoitu piste EDP (Event Detection Point). Staattinen aktivointi suoritetaan uuden palvelun asennuksen yhteydessä. Tätä toimenpidettä ei ole standardoitu ITU-T:n nykyisissä standardeissa. /5/

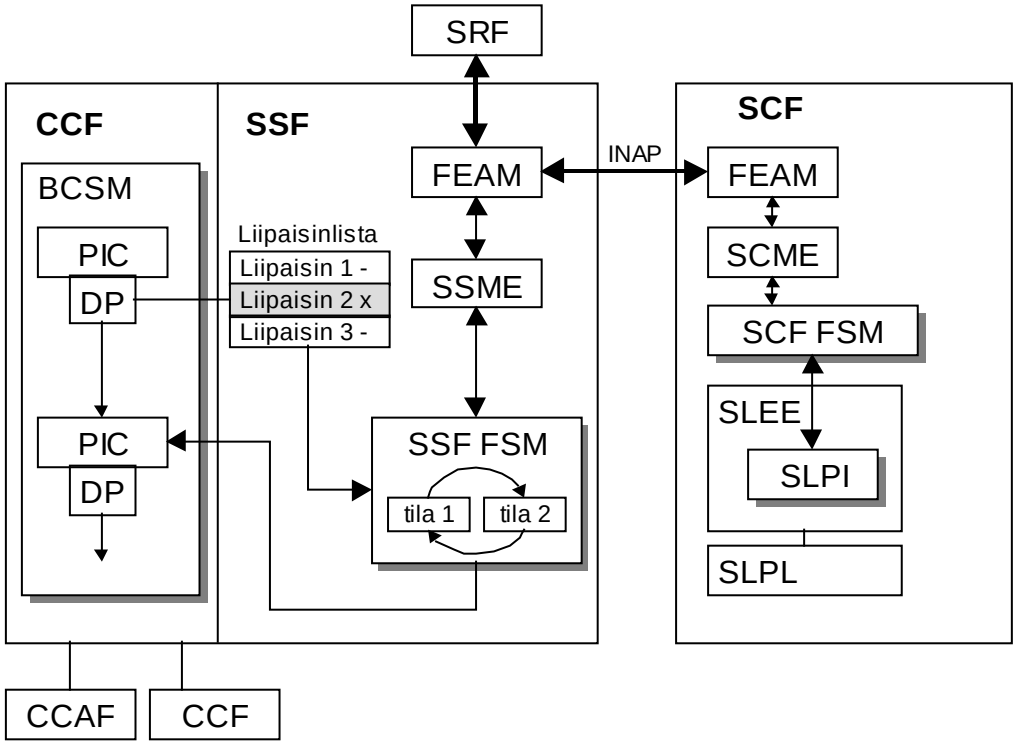
TDP-havaintopiste voi sisältää useita liipaisimia (trigger) ja niihin liittyviä ehtoja. Aktivoinnin lisäksi havaintopisteen liipaisimen tulee siis täyttää tietty ehto ennen SCF-SSF-vuorovaikutussuhteen aloittamista. Ehto voi olla esim. tietty B-tilaajan (vastaanottajan) numero tai sen alkuosa (esim. 0800, ilmaispuhelu). Jos puhelunkäsittelyyn halutaan lisäohjeita, asetetaan havainnointipisteen tyypiksi odottava pyyntö (R, Request). Tällöin puhelumalli jää odottamaan ohjeita palvelulogiikalta ja jatkaa toimintaansa vasta saatuaan tarvittavat ohjeet. Paluuviesti voi ohjata puhelumallin esim. toiseen tilaan tai viestissä voi olla pyyntö aktivoida lisää havaintopisteitä (dynaaminen aktivointi). Toinen havaintopisteen tyyppi on ilmoitus (N, notification). Tällöin puhelunmuodostusta ei pysäytetä vaan lähetetään ainoastaan viesti, johon ei odoteta vastausta. Näiden ehtojen mukaan havaintopisteet nimetään seuraavasti: TDP-N, TDP-R, EDP-N, EDP-R. /5/

2.4.3 CCF/SSF/SCF -toiminta

CCF:n saadessa ilmoituksen tulevasta puhelusta luo se uuden instanssin tulevan puhelumallin tilakoneesta. Puhelunmuodostus alkaa juuri luodun tilakoneen alkutilasta ja etenee puhelumallin mukaan. Jos jokin havaintopisteistä on asetettu TDP-tyyppiseksi ja liipaisimeen asetettu ehto täyttyy, luodaan instanssi SSF-tilakoneesta (SSF-FSM). Jos kyseessä oli TDP-R -tyyppinen havaintopiste, SSF-tilakone siirtyy odotustilaan (waiting for instructions) ja lähettää palvelupyyntöviestin (initialDP) palvelunohjauspisteeseen. SSF sisältää yhden SSF-tilakoneen per älyverkkopuhelu. SSME/SCME (SSF/SCF Management Entity) ja FEAM (Functional Entity Manager) hoitavat SSF/SCF-tilakoneiden liikennöinnin toiminnallisiin kokonaisuuksiin.

Palvelupyyntöviestin saapuessa SCF:ään käynnistetään viestin parametrien mukainen palvelulogiikkaohjelma (SLP, Service Logic Program) palvelulogiikkaohjelmakirjastosta (SLPL, Service Logic Program Library). Palvelun valintaan käytetään yleensä palveluavainparametriä (service key).

Näin palvelupyyntöviesti synnyttää uuden palvelulogiikkaohjelmainstanssin (SLPI, Service Logic Program Instance). Palvelunajoympäristö (SLEE, Service Logic Execution Environment) voi sisältää useita palveluinstansseja ja jokaisella niistä on myös oma SCF-tilakone (SCF FSM).



Kuva 4. CCF/SSF/SCF -toiminta.

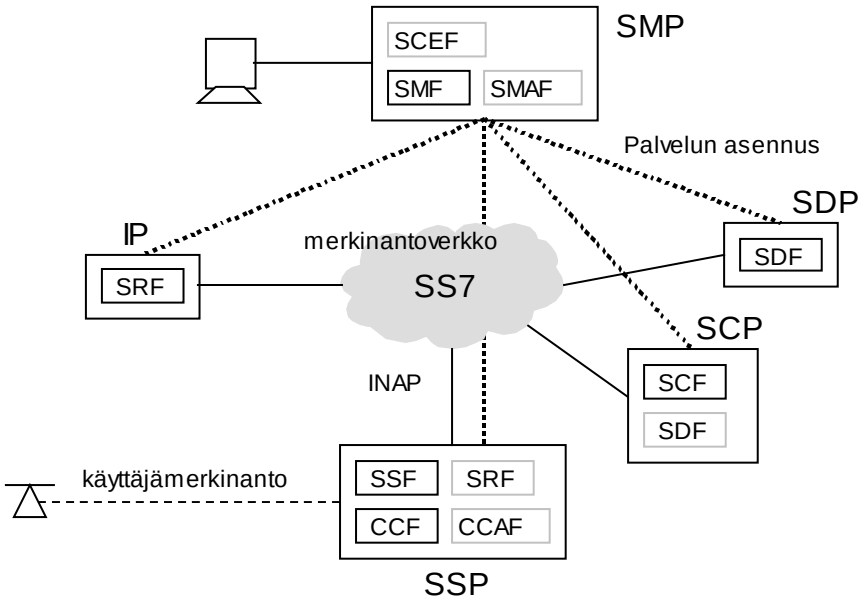
2.5 Fyysinen taso

Fyysinen taso (Physical Plane) määrittelee älyverkon fyysisen arkkitehtuurin. Taso kuvaa fyysisiä itsenäisiä kokonaisuuksia (Physical Entity), jotka sisältävät vähintään yhden toiminnallisen kokonaisuuden, sekä niiden välisiä protokollia. Tärkeimmät fyysiset kokonaisuudet ovat (kuva 5):

- Palvelunkytkentäpiste (SSP, Service Switching Point) on digitaalinen puhelinkeskus, jota käytetään älyverkkopalveluiden tarjoamiseen. Toiminnallisesti SSP sisältää CCF- ja SSF-toiminnot . Jos SSP on paikalliskeskus, se voi sisältää myös CCAF-toiminnon.
- Palvelunohjauspiste (SCP, Service Control Point) ohjaa älyverkkopalveluita. Se sisältää toiminnallisesti SCF:n ja mahdollisesti myös SDF:n. SCP voi olla esim. standardi Unix-työasema tai reaaliaikakäyttöjärjestelmällä varustettu moniprosessoritietotokone. SCP

voi sijaita myös SSP:n yhteydessä, jolloin toteutusta kutsutaan nimellä SSCP (Service Switching and Control Point).

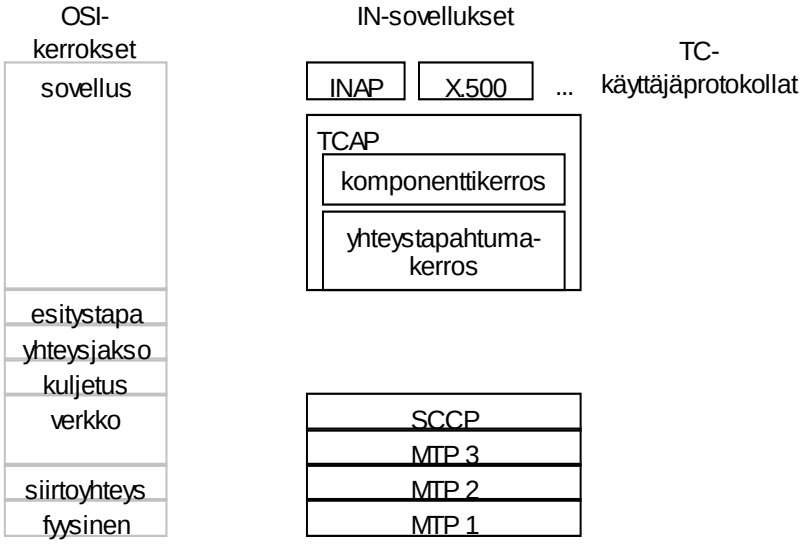
- Palvelutietopiste (SDP, Service Data Point) sisältää SDF-toiminnon ja on käytännössä tehokas tietokantapalvelin.
- Älykäs oheislaite (IP, Intelligent Peripheral) sisältää älyverkkopalveluiden tarjoamisessa tarvittavaa lisätietoa tai toiminnallisuutta. Palvelun aikana älykäs oheislaite toimii palvelunohjauspisteeltä saamiensa komentojen mukaan. Älykkäällä oheislaitteella on myös puheyhteys SSP:hen.
- Palvelusolmu (SN, Service Node) sisältää SCF:n, SDF:n, SRF:n ja SSF/CCF:n. Palvelusolmulla voidaan toteuttaa nopeasti haluttu palvelu ilman merkinantoverkkoa.
- Palveluiden hallintapiste (SMP, Service Management Point) sisältää SMF:n ja mahdollisesti myös SMAF:n ja SCEF:n. SMP voi olla kytketty kaikkiin fyysisiin kokonaisuuksiin, joskin ei merkinantoverkon kautta vaan esim. X.25-verkon kautta. Tätä rajapintaa ei ole määritelty ITU-T:n CS-1 määrityksessä, mutta protokollaksi soveltuu esim. CMIP-verkonhallintaprotokolla (Common Management Interface Protocol). SMP:tä voidaan käyttää tiedon hallitsemisen lisäksi myös tiedon keräämiseen eri fyysisistä kokonaisuuksista.
- Palvelunluontipiste (SCEP, Service Creation Environment Point) sisältää SCEF-toiminteen. SCEP on suoraan yhteydessä SMP:hen.
- Palvelunhallinnanliityntäpiste (SMAP, Service Management Agent Point) tarjoaa käyttäjille yhteyden yhteen tai useampaan SMP:hen.



Kuva 5. Älyverkon toiminnalliset ja fyysiset kokonaisuudet.

3 MERKINANTOVERKKO

SS7-merkinantoverkkoa (Signaling System no. 7) käytetään digitaalisen televerkon liikenteen ohjaukseen ja hallintaan (esim. puhelunohjaus, IN-palvelut). SS7-verkossa on pyritty luotettavaan ja nopeaan tiedonsiirtomekanismiin eri telekommunikaatiojärjestelmien komponenttien välillä erottaen puheyhteys merkinantoyhteydestä. SS7-verkko eroaa osittain avoimen viitemallin eli OSI-mallin (Open Systems Interconnection) kerrosrakenteesta, koska se on toteutettu ennen OSI-standardia. Kuvassa 6 on esitetty SS7-verkon IN-pinoon liittyvät protokollatasot ja niitä vastaavat OSI-



kerrokset. /17/

3.1 Sanomansiirto-osa

Sanomansiirto-osa (MTP, Message Transfer Part) toimii SS7-järjestelmän perustana tarjoten luotettavan, yhteydettömän siirtojärjestelmän merkinantopisteiden välillä. MTP on jaettu kolmeen toiminnalliseen tasoon ja hallintaosaan. Taso 1 määrittelee merkinantolinkin fysikaaliset, sähköiset ja toiminnalliset ominaisuudet. Merkinantolinkkinä käytetään Euroopassa 64

Kuva 6. SS7-merkinantoverkon tasot.

kbit/s digitaalista yhteyttä, joka on 2 Mbit/s PCM-johdon (Pulse Code Modulation) yksi aikaväli. Taso 2 (merkinantokanava) huolehtii kehysten oikeellisuudesta, järjestyksestä ja uudelleenlähetyksistä. Taso 3 (merkinantoverkko) tarjoaa MTP-käyttäjäosalle luotettavan ja yhteydettömän siirron yksikäsitteisillä pistekoodiosoitteilla (PC, Point Code). Hallintatoimilla pyritään takaamaan sanomien siirto verkossa esiintyvistä ylikuormitus- ja vikatilanteista huolimatta. /17/

3.2 Merkinantoyhteyden ohjausosa

Merkinantoyhteyden ohjausosa (SCCP, Signaling Connection Control Part) tarjoaa verkkokerroksen yhteydellisen ja yhteydettömän tiedonsiirron. SCCP:n osoite muodostuu globaalista otsikosta (GT, Global Title) ja alijärjestelmänumerosta (SSN, Subsystem Number). Globaali otsikko ei sisällä suoraa reititystietoa, vaan esim. näppäillyn numeron. Varsinainen reititys tapahtuu DPC:n (Destination Point Code) avulla. Alijärjestelmänumeroa käytetään paikallisesti SCCP:n käyttäjien tunnistamiseen. /17/

3.3 Tapahtumankäsittelyosa

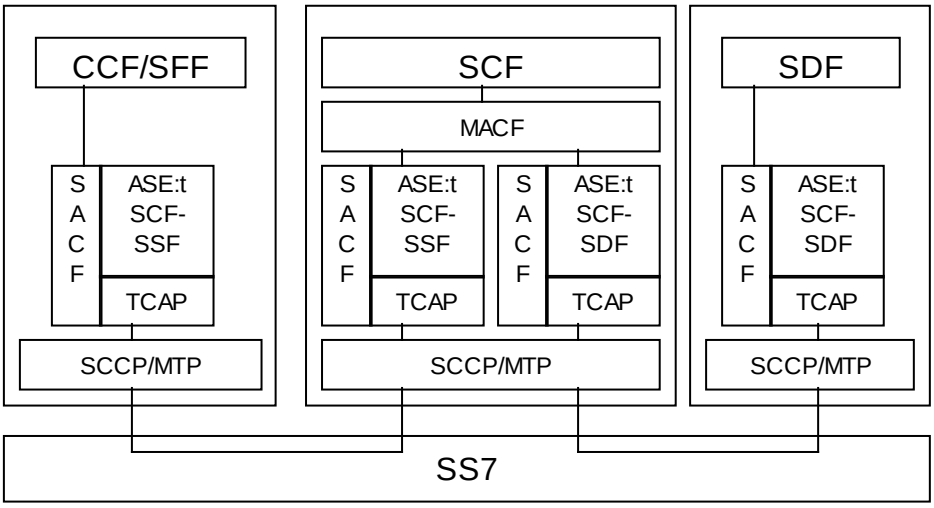
Tapahtumankäsittelyosa (TCAP, Transaction Capabilities Application Part) on yhteydetön sovelluskerroksen protokolla, joka tarjoaa palveluja etäoperaatioiden suorittamiseen. Etäoperaatiot koostuvat operaatiosta ja siihen mahdollisesti saatavasta vastauksesta tai virheestä. TCAP on jaettu kahteen osaan, yhteystapahtumakerrokseen (transaction sublayer) ja komponenttikerrokseen (component sublayer). Yhteystapahtumakerros huolehtii merkinantopisteiden välisestä vuoropuhelusta (dialogue) ja komponenttikerros käyttäjän tiedon lähetyksen hallinnasta. Vuoropuhelut identifioidaan yksikäsitteisillä vuoropuhelutunnisteilla (dialogue identifier). /3/

3.4 INAP

Standardi Q.1218 määrittelee CS-1:sen mukaisen älyverkkosovellusprotokollan INAP:in (Intelligent Network Application Protocol). INAP-protokollaa käytetään älyverkon käsitteellisessä mallissa määriteltyjen toiminnallisten kokonaisuuksien välillä. INAP-protokolla sijoittuu OSI-mallissa ylimmälle eli sovellustasolle. OSI-mallin ylin osio koostuu yleisistä ja sovelluskohtaisista sovelluspalveluelementeistä (ASE, Application Service Element), jotka on kuvattu ASN.1-kielellä (Abstract Syntax Notation no. 1). ASE sisältää operaatiot, niiden paluuarvot ja virhetilanteet. ASE:t voidaan ryhmitellä eri sovellusviitekehyksiin (AC, Application Context). Sovellusviitekehys määrittelee toiminnallisten kokonaisuuksien välillä tarvittavan INAP-protokollan osajoukon (esim. SSF-SCF). /3/

INAP koostuu joukosta sovellusviitekehyksiä ja sen toiminta pohjautuu etäoperaatio ASE:n eli ROSE:n (Remote Operation Service Element) käyttöön. TCAP vastaa OSI-mallin ROSE:a eli OSI-mallin mukaan myös TCAP on ASE, joka tarjoaa yksinkertaisen "pyyntö-vastaus"-palvelun (Request-Reply service) muille sovelluspalveluelementeille. Tästä johtuen esim. INAP-protokollaa kutsutaan TC-käyttäjäprotokollaksi (TC user protocol).

Fyysisellä kokonaisuudella voi olla joko yksi tai useampia yhteyksiä muihin fyysisiin kokonaisuuksiin. Ensimmäisessä tapauksessa sovelluspalveluelementtien hallintaan käytetään yksittäisen sidoksen kontrollifunktiota (SACF, Single Association Control Function) ja toisessa tapauksessa usean sidoksen kontrollifunktiota (MACF, Multiple Association Control Element). /3/



Kuva 7. INAP-protokollan rakenne.

4 UUDET OLIOTEKNOLOGIAT

Vaikka perinteisen proseduraalisen ohjelmoinnin periaatteet ovatkin pikkuhiljaa parantaneet ohjelmien selkeyttä ja luotettavuutta, on oliomenetelmien ja -kielten käyttöönotto ollut kaikkein merkittävin askel ohjelmistojen kehityspolulla.

4.1 Olio-ohjelmointi

Oliosuuntautunut (object-oriented) ohjelmointi mahdollistaa tiedon ja sitä käsittelevien metodien yhdistämisen loogiseksi kokonaisuudeksi eli luokaksi. Luokilla voidaan mallintaa ongelma-alueeseen liittyviä todellisen maailman yksiköitä ja asioita. Tämä lähestymistapa auttaa suunnittelijaa jakamaan ongelma-alueen helposti hallittaviksi koodipaloiksi, joita voidaan myöhemmin esim. laajentaa perimällä. Oliosuuntautunut ohjelmointi on ohjannut suunnittelijoita ja ohjelmoijia omaksumaan aivan uuden ajattelutavan. Vaikka olio-ohjelmoinnilla onkin kiistattomat etunsa, on sen oikea oppiminen varsin pitkä ja vaativa prosessi. Oppimista ja käytännön työtä helpottamaan onkin kehitetty joukko tässä työssä uusiksi oliomenetelmiksi ja -teknologioiksi kutsuttuja menetelmiä ja teknologioita.

4.2 Suunnittelumallit

Oliopohjaisten ohjelmistojen suunnittelu on vaativa prosessi varsinkin, jos ohjelmistojen osien halutaan olevan uudelleenkäytettäviä. Suunnitelman tulisi täyttää tämänhetkiset vaatimusmäärittelyt ja sen tulisi mukautua uusiin vaatimuksiin. Tällaisen suunnitelman tekeminen ei yleensä onnistu ensimmäisellä kerralla edes kokeneelta suunnittelijalta. Hyvän ja kokeneen suunnittelijan erottaa aloittelijasta yleensä se, että hän voi käyttää aikaisemmin hyviksi katsomiaan ratkaisuja ja jokaista ongelmaa ei tarvitse ratkaista alusta asti. Tämä on niin sanottua suunnittelutiedon uudelleenkäyttöä, mistä suunnittelumalleissa (design patterns) on pitkälti kyse. Suunnittelumalleihin liittyy neljä tärkeää elementtiä. Mallin nimi antaa lyhyen nimen ongelmalle.

Nimettyjen suunnittelusääntöjen avulla suunnitelmista voidaan kirjoittaa ja puhua korkeammalla tasolla niin, että muutkin suunnittelijat ymmärtävät mistä on kysymys. Ongelmankuvaus kertoo milloin suunnittelusääntöä voidaan käyttää. Ratkaisunkuvaus määrittelee ongelman ratkaisemiseksi tarvittavat osat (luokat), niiden suhteet, vastuut ja yhteistyön. Yleensä mukana on myös esimerkkikoodi jollakin ohjelmointikielellä ja UML-kuvaus (Unified Modeling Language). Seurauskuvaus kertoo suunnittelumallin odotetun tuloksen, vaikutukset ja mahdolliset suunnittelusäännön käytöstä aiheutuvat ongelmat. /2/

Seuraavassa esitellään lyhyesti joitakin käytännön osuudessa käytettyjä suunnittelumalleja.

- MVC-suunnittelumalli (Model View Control) erottaa mallin (sovellusolion) sen käyttäjälle näkyvän esityksen ja tavan jolla käyttäjä voi vaikuttaa malliin. MVC-suunnittelumallin avulla on helppo muuttaa mallista olevia näkymiä, puuttumatta itse mallin toteutukseen.
- Tehdassuunnittelumalli (factory) määrittelee tehdasrajapinnan, jonka avulla luodaan haluttuja olioita. Tehdas peittää olion luontiin liittyvän konkreettisen toteutuksen.
- Fasadisuunnittelumalli (facade) tarjoaa yhtenäisen rajapinnan useille rajapinnoille. Fasadin määrittelemä korkeammantason rajapinta on helpompi käyttää, kuin useat erilliset alemmantason rajapinnat.

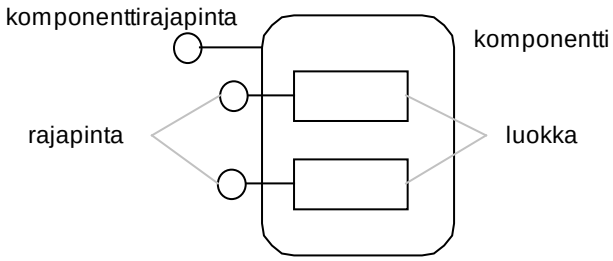
4.3 Sovelluskehykset

Sovelluskehys (framework) sisältää joukon valmiita luokkia, joita ohjelmoija voi käyttää, laajentaa ja räätälöidä liittyen tiettyyn sovellusalueeseen. Toisin kuin luokkakirjastot, sovelluskehykset määräävät sovelluksen arkkitehtuurin, rakenteen ja oletustoiminnan. Sovelluskehys ei siis ole pelkästään joukko luokkia, vaan se määrittelee myös luokkien väliset yhteydet. Usein käytetty esimerkki reaalimaailmasta on piirilevy. Piirilevy määrittelee "sovelluksen" johdotuksen, mutta toimintaa voidaan muunnella vaihtamalla uusia

komponentteja niille varattuihin koloihin. Hyvin suunnitellussa sovelluskehyksestä on käytetty hyväksi useita eri suunnittelumalleja.

Sovelluskehys ohjaa ohjelmoijaa uuteen ajattelutapaan: ohjelmoija ei enää kutsu funktioita määrittelemässään järjestyksessä, vaan sovelluskehys kutsuu sovelluskehyksen sääntöjen mukaan tehtyjä metodeja. Näin ohjelmoija voi keskittyä ongelman ratkaisuun ja jättää infrastruktuurin sovelluskehyksen suunnittelijoiden harteille. Hyvin suunniteltuun sovelluskehykseen onkin helppo tehdä lisäyksiä säilyttäen kuitenkin yhteensopivuus jo tehtyjen sovellusten kanssa. Myös sovellusten ylläpito, luotettavuus ja aikavaateet saadaan paremmin hallintaan. /16/

4.4 Komponentit



Kuva 8. Komponentin rakenne.

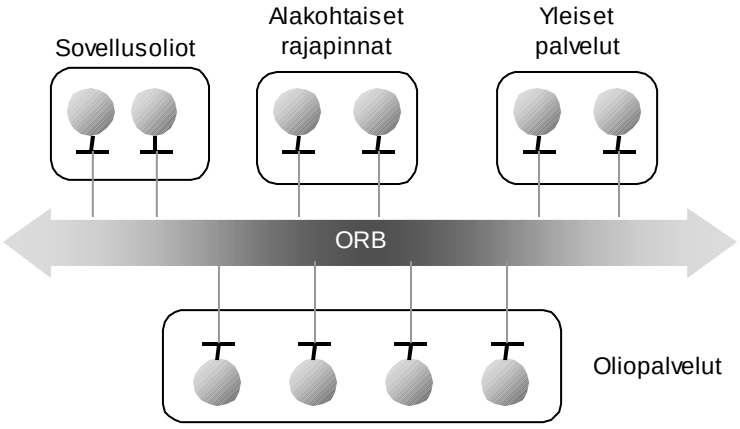
Komponentti (component) on yksilöitävissä oleva ohjelmiston pala, jolla on selkeät rajat. Komponentilla on komponenttimallissa määritelty rajapinta, ominaisuudet ja sen tulee olla suunniteltu uudelleenkäytettäväksi eri tilanteissa. Käyttäjän ei tarvitse tuntea komponentin toteutusyksityiskohtia, vaan hän näkee ainoastaan komponentin tarjoaman rajapinnan tai rajapinnat. Komponentin sisäiseen toimintaan liittyy usein useampia luokkia.

Komponenteista voidaan koota kokonaisuuksia, jotka voivat olla taas vuorostaan suuremman toiminnallisuuden omaavia komponentteja. Halutun

toiminnallisuuden toteuttavat komponentit liitetään toisiinsa tai esim. sovelluskehykseen, jolloin näiden yhteistoiminta saa aikaan halutun toiminnallisuuden eli varsinaisen sovelluksen. Komponentteja voidaan pitää perustellusti olio-ohjelmoinnin seuraavana askeleena, joka mahdollistaa olio-ohjelmoinnin lupaaman helpon koodin uudelleenkäytön.

4.5 CORBA

CORBA-arkkitehtuuri (Common Object Request Broker Architecture) on tärkein ja kunnianhimoisin projekti avoimen (olio) välisovelluksen (middleware) määrittelemiseksi. CORBA-standardia kehittää OMG (Object Management Group) konsortion, jonka jäseninä on yli 700 eri aloilla toimivaa yritystä. Merkittävin standardointityön ulkopuolelle jättäytynyt yritys on Microsoft, joka kehittää omaa DCOM-tekniikkaa (Distributed Component Object Model). CORBA:n perusideana on mahdollistaa eri koneisiin ja ympäristöihin hajautettujen olioiden paikannus ja yhteistoiminta.



Kuva 9. OMA-arkkitehtuuri.

OMG:n viitemallia kutsutaan OMA:ksi (Object Management Architecture). OMA:aan sisältyy neljä osa-aluetta: ORB (Object Request Broker) eli

oliokutsunvälittäjä (kaupalliselta nimeltään CORBA), oliopalvelut (object services), yleiset palvelut (common facilities) sekä alakohtaiset rajapinnat (domain interfaces). Kuvassa 9 esiintyvillä sovellusolioilla (application object) on käyttäjän määrittelemä rajapinta ja näin ollen ne eivät kuulu OMG standardoinnin piiriin. /1/

4.5.1 Rajapinnan kuvauskieli

CORBA-olioiden rajapintakuvauksissa käytetään IDL-rajapinnankuvauskieltä (Interface Definition Language), jolla kuvataan vain rajapintoja eikä toteutusta. IDL-kieli on riippumaton olemassa olevista toteutuskielistä. IDL-kielellä määritellään rajapinnan attribuutit, tietotyypit, kantarajapinta, operaatiot, operaatioiden parametrit ja operaatioiden poikkeustilanteet (exceptions). IDL-kieli on läheistä sukua C++ kielelle, josta se on perinyt osan kieliopistaan. OMG on määritellyt IDL-sidonnan usealle toteutuskielelle, esim. C, C++, Smalltalk, Ada, Cobol ja Java. Toteutuksen yhteydessä IDL-rajapintakuvaus käännetään halutulle IDL-sidonnan omaavalle toteutuskielelle, ja rajapinnan "taakse" lisätään haluttu toiminnallisuus.

4.5.2 ORB

ORB mahdollistaa olioiden etäkutsut, oli kutsuttava olio sitten paikallinen tai toisessa koneessa sijaitseva. Kutsumekanismi on samankaltainen kuin RPC (Remote Procedure Call). Suurimpana erona on se, että oliokutsuväylän kautta kutsutaan tietyn olion tiettyä metodia, eikä määrätyssä prosessissa olevan ohjelman funktiota. Saman metodin kutsuminen eri oliosta voi tuottaa eri tuloksen

CORBA 2.0 standardin määrittelemän IIOP-protokollan (Internet Inter-ORB Protocol) myötä eri valmistajien CORBA-toteutukset tulivat yhteensopiviksi. Tätä ennen eri valmistajien ORB:it olivat käyttäneet omia protokollia, hankaloittaen näin eri toteutusten yhteistoimintaa.

4.5.3 Oliopalvelut

Oliopalvelut sisältävät joukon järjestelmätason palveluita, joilla on IDL-rajapinta. Oliopalveluiden tarkoitus on täydentää ORB:in toiminnallisuutta ja tarjota yleisesti tarvittuja palveluita sovellusolioille. Oliopalveluita on jo nyt määritelty yli 15 kappaletta ja uusia kehitetään jatkuvasti. Tärkeimpiä palveluja ovat mm.:

- nimipalvelu (naming service),
- elämänkaari (life cycle),
- tapahtumapalvelu (event service),
- transaktiopalvelu (transaction service),
- sanomapalvelu (messaging service).

4.5.4 Alakohtaiset rajapinnat ja yleiset palvelut

Alakohtaiset rajapinnat sisältävät eri sovellusalueisiin liittyviä rajapintoja. Sovellusalueita ovat esim. tietoliikenne, terveydenhoito ja kaupankäynti. Yleiset palvelut taas sisältävät rajapintoja, jotka ovat käytettävissä useilla sovellusalueilla ja näiden palveluiden rajapintojen semantiikka on lähempänä loppukäyttäjää kuin oliopalveluiden. Yleiset palvelut voidaan nähdä sovelluskehyksinä, jotka käyttävät sisäisesti oliopalveluita hyväkseen. /1/

Alakohtaisten rajapintojen määrittelijöiden joukosta on erottunut edukseen tietoliikennetyöryhmä (Telecommunications Domain Task Force), joka on määritellyt aktiivisesti rajapintoja tietoliikennekäyttöön. Diplomityön kannalta työryhmän merkittävin dokumentti on "Interworking between CORBA and Intelligent Network Systems". Siinä määritellään perinteisten ja CORBA-pohjaisten älyverkkototeutusten yhteistoiminta.

4.5.5 CORBA 3.0

Uudelle CORBA 3.0 versiolle on asetettu paljon odotuksia. Uuteen versioon liittyy useita parannuksia, joista merkittävimpiä ovat CORBA-

komponenttimalli, tuki skriptikielille, olioparametrit, asynkroniset viestit ja laajennettu Java-tuki. /18/

Komponentin määritelmän mukaan jo nykyisetkin CORBA-oliot rajapintoineen ovat komponentteja, mutta varsinaista komponenttimallia ei ole aikaisemmin määritelty. CORBA-komponenttimalli sisältää olion luontiin, elämänkaareen, viesteihin, rajapintoihin ja pakkaukseen liittyviä asioita. Komponenttimallin tarkoituksena on helpottaa ja yksinkertaistaa CORBA-sovellusten tekemistä ja näin laajentaa CORBA:n käytettävyyttä. Yhdessä skriptituen kanssa komponenttimalli tuleekin varmasti laskemaan CORBA-ohjelmointiin liittyvää oppimiskäyrää. CORBA-komponenttimallissa on paljon yhtäläisyyksiä JavaBeans- ja EJB-komponenttimallien (Enterpise JavaBeans) kanssa. Onkin merkillepantavaa, kuinka paljon Java-standardeilla on vaikutusta OMG:n tämänhetkiseen työhön.

5 JAVA

Java on Sun Microsystems -yhtiön kehittämä oliopohjainen verkko-ohjelmointikieli. Javaan liittyviä määritelmiä ovat: yksinkertainen, oliopohjainen, siirrettävä, tulkattava, vakaa, turvallinen, dynaaminen ja suoristuskykyinen.

5.1 Java-kieli

Java-kieli ammentaa vaikutteita monista aikaisemmista kielistä. Se on syntaktisesti hyvin lähellä C++-kieltä, mutta kielen edistynyt oliomalli on kuitenkin lähempänä Smalltalk-kieltä. Syntaktiset samankaltaisuudet C++-kielen kanssa helpottavat kielen oppimista. Toisaalta Java-kielestä on jätetty pois monia C++-kielen virhealttiita ja vaikeita ominaisuuksia, kuten suora muistiosoitus eli pointterit. Suurin osa C:llä ja C++:lla tehtyjen ohjelmien ongelmakohdista liittyy tavalla tai toisella virheelliseen muistiosoitukseen. Javassa muistiosoitteet on "piilotettu" käyttäjältä, ja fyysisen muistin vapautuksesta huolehtii ohjelmoijalle näkymätön roskienkeruualgoritmi. Java-kieli on puhdas oliokieli eli kaikki ohjelman osat kuvataan luokilla, rajapinnoilla ja niissä sijaitsevilla metodeilla. Perusperiaatteena on ollut pitää kieli mahdollisimman yksinkertaisena, säilyttäen kuitenkin kielen ilmaisuvoima. Vaikka Java-kieltä näin ollen voidaankin pitää helppona, ei se tarkoita sitä, että kenestä tahansa tulisi parissa päivässä Java-ohjelmoinnin ammattilainen. Laajojen, laadukkaiden ohjelmistojen tekeminen on ja tulee olemaan kokeneiden ammattilaisten työsarkaa.

Java-kielen siirrettävyys perustuu tavukoodiksi kutsuttavaan välikieleen. Tavukoodi on laitteistoriippumaton binääriesitysmuoto ohjelmasta. Java-lähdekoodi käännetään tavukoodiksi, joka siirretään virtuaalikoneen (virtual machine) sisältävään kohdeympäristöön. Virtuaalikone on tulkki, joka tulkkaa

tavukoodia kohdelaitteen konekielelle. Javalla tehdyt ohjelmat ovat siis ajettavissa kaikissa virtuaalikoneen omaavissa ympäristöissä.

Javan tietoturvaominaisuudet ovat oleellinen osa kielen rakennetta. Koska Java-koodia voidaan ajaa monissa eri kohdeympäristöissä, tarvitaan takuu siitä, että ohjelma ei pääse käsiksi kohdelaitteen kiellettyihin systeemiresursseihin, esim. tiedostojärjestelmään. Turvallisuusajatteluun ja ohjelmien vakauteen vaikuttavia tekijöitä ovat jo aiemmin kuvattu suorien muistiosoitusten puuttuminen, ajonaikaiset tarkastukset, tavukoodin oikeellisuuden varmistaminen ja kattava virheenkäsittelymekanismi.

Java-kieli on luonteeltaan dynaaminen eli luokat ladataan muistiin vasta silloin, kun niihin viitataan. Tämän takia Java-luokkia ei tarvitse kehitysvaiheessa linkittää yhteen, vaan se tapahtuu ajonaikaisesti ja täysin näkymättömästi. Dynaamisuus mahdollistaa myös sen, että ohjelman osat sijaitsevat fyysisesti eri paikoissa, joista ne voidaan ladata tarvittaessa.

Usein Javasta puhuttaessa törmätään epäilevään suhtautumiseen Javan suorituskyvystä. Java on tulkattavan luonteensa mukaisesti luonnollisesti hitaampi kuin esim. konekieleksi käännetty C++-koodi. Turvallisuusvaatimukset, ajonaikaiset tarkastukset ja roskienkeruualgoritmi vaativat osansa suorituskyvystä. Kun puhutaan hajautetusta verkko-ohjelmoinnista, moni ohjelmoija on valmis tinkimään suorituskyvystä saadakseen enemmän apua toteutuskieleltä ja ajoympäristöltä. On myös huomattava, että koneiden suorituskyky kasvaa jatkuvasti, kun taas ohjelmien luotettavuudessa, siirrettävyydessä ja laadukkuudessa ei ole ollut nähtävissä vastaavia harppauksia. Tämän johdosta siirtyminen uusiin ohjelmointikieliin ja ympäristöihin on enemmän kuin suositeltavaa.

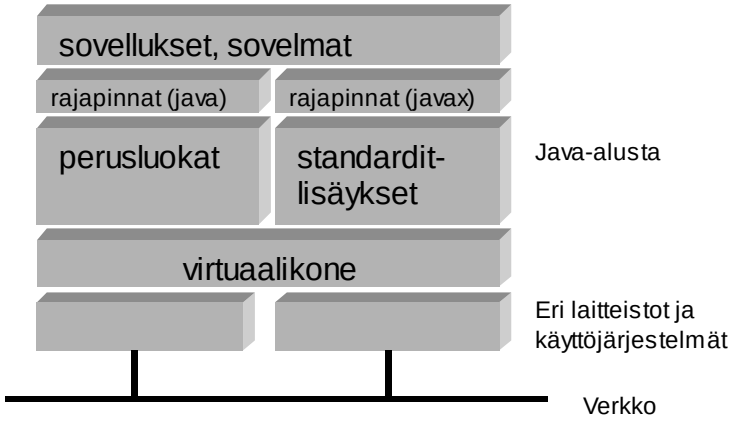
Virtuaalikoneiden puolella on tapahtunut paljon kehitystä. JIT -virtuaalikone (Just In Time) kääntää "lennossa" tavukoodin suoraan konekieleksi. Tässä prosessissa käännetään vain ne osat, joita käytetään. Käännetyt osat säilyvät muistissa ohjelman ajon ajan eli kukin käännösosuus tehdään vain kerran. Käännösvaiheessa ei voida tehdä kovin raskasta optimointia, koska käännöstapahtuma pitää olla hyvin nopea. JIT-virtuaalikone ei myöskään tiedä, mitkä kohdat kannattaa optimoida, joten se yrittää optimoida turhatkin kohdat, kuten alustukset ja harvoin kutsutut metodit. Uusin lähestymistapa on dynaaminen käännös, siinä tavukoodia ajetaan ensin tulkkaavassa virtuaalikoneessa, josta saadaan ajonaikaista suorituskykytietoa. Jos esim. metodia kutsutaan useita kertoja peräkkäin pitkässä silmukassa, tehdään tälle kohdalle optimoitu JIT-käännös ja siirrytään natiivikoodin ajoon. Ajonaikaisen tiedon perusteella voidaan tehdä hyvin pitkälle menevää optimointia. Niin sanotuille kuumille paikoille (hot spot) voidaan tehdä optimistinen käännös, jossa poistetaan virheenkäsittely ja aikaa vievät metodikutsut. Tällä tekniikalla voidaan päästä jopa parempaan suorituskykyyn kuin mihin optimoimaton käännetty C++-koodi yltää. Tavukoodi ja lähdekoodi voidaan kääntää suoraan konekielelle, jolloin tietenkin siirrettävyys kärsii. Tämä lähestymistapa sopii parhaiten staattisiin palvelinpuolen sovelluksiin. Suorituskyvyn puolella ei ole siis vielä sanottu viimeistä sanaa, vaan parannuksia on näköpiirissä.

5.2 Java-alusta

Java-alusta (Java platform) koostuu virtuaalikoneesta, perusluokista (core classes), standardeista laajennusluokista (standard extension classes) ja näiden ohjelmointirajapinnoista (API, Application Programming Interface). Luokat ja rajapinnat muodostavat luokkakirjastoja, jotka on jaettu omiin moduuleihin. Java-kielessä luokkakirjastomoduuleista käytetään ilmausta pakkaus (package).

JDK-kehitystyökalukokoelma (Java Development Kit) sisältää perusluokkakirjastot ja yksinkertaiset työkalut Java-ohjelmien kehittämiseen.

JDK:n ensimmäinen julkinen versio oli 1.0. Se oli tarkoitettu lähinnä pienten, siirrettävien, selaimessa ajettavien sovelmien (applet) tekemiseen ja sisälsi vain kaikkein välttämättömimmät ohjelmointirajapinnat. Seuraava versio, JDK 1.1, olikin jo kuin eri planeetalta. Versionumeron olisi pitänyt olla vähintään 2.0, niin paljon uusia luokkakirjastoja ja rajapintoja se sisälsi. Tässä vaiheessa alkoi jo olla selvää, että Java oli kasvanut kiinnostavasta kielestä täysimittaiseksi ohjelma-alustaksi. Parannusten myötä Java alkoi herättää kiinnostusta palvelinpuolen sovelluskehittäjissä.



Kuva 10. Java-alusta.

Sunin uusin julkistus on Java 2 -alusta, joka tunnetaan myös JDK 1.2:sena. Toiminnallisuus ja koko on kasvanut edelleen häkellyttävää tahtia. Uusien versioiden julkaisutahdin on mahdollistanut se, että Sunin lisäksi myös muut yritykset ovat osallistuneet toiminnallisuuden ja rajapintojen määrittelemiseen. Tämä osittain avoin internet-standardointi on näyttänyt voimansa ja nyt onkin varmistettava, että tämä avoimuus jatkuu ja laajenee. Uusien ominaisuuksien esille marssi on tuonut mukanaan myös versionhallintaongelmia ja harmaita hiuksia sovelluskehittäjille. Java alkaa kuitenkin selvästi lähestyä tiettyä kypsyystasoa ja julkaisutahdin odotetaan rauhoittuvan.

5.3 JavaBeans

JavaBeans-määrittely kuluu Javan perusominaisuuksiin, ja se kuvaa Javan alustariippumattoman komponenttimallin. JavaBeans-komponenttimalli määrittelee säännöt, joita luokan on noudatettava ollakseen JavaBeans-komponentti. Komponentti voi olla joko näkyvä tai näkymätön. Käyttöliittymäkomponentit kuuluvat näkyviin komponentteihin, ja niillä on ajonaikana jokin graafinen ilmentymä (esim. painonappi). Näkymättömät komponentit taas näkyvät vain kehitysvaiheessa jossakin JavaBeans-määritystä hyödyntävässä sovelluskehittimessä. JavaBeans-standardi on suunniteltu alun perin tukemaan nimenomaan graafisia sovelluskehittimiä. /1/

Java-komponentit ovat ns. suljettuja komponentteja (black-box). Tämä tarkoittaa sitä, että niitä ei voi laajentaa, jos lähdekoodia ei ole saatavilla. Sovelluskehittimet pystyvät kuitenkin lukemaan komponentin ominaisuudet ilman lähdekoodiakin. Tämä ominaisuus perustuu nimeämissääntöihin (naming patterns), esim. komponentin ominaisuus (property) määritellään luokkaan kirjoittamalla sille asetus- ja lukumetodit (set, get methods). Jos luokalla on esim. ominaisuus "nimi", niin määrityksen mukaan sille pitää olla "setNimi(string nimi)" ja "string getNimi()" metodit. Metodien nimet voidaan määrätä myös eksplisiittisesti BeanInfo-luokan avulla. Yksinkertaisimmassa tapauksessa riittää, että noudatetaan tiettyjä nimeämissääntöjä ja järjestelmä luo BeanInfo-olion "lennossa". JavaBeans-komponenteilla ei siis ole erillistä rajapintaa, vaan rajapinta muodostuu itse luokasta/oliosta edellä esitettyjen sääntöjen mukaan. Seuraavassa listassa on lueteltu komponenttien tarvitsemia Java-alustaan ja Java-kieleen liittyviä ominaisuuksia. /15/

- Itsehavainto (introspection) tarjoaa mekanismin, jolla luokan metodit voidaan selvittää ajonaikaisesti.

- Ominaisuudet (properties) kuvaavat komponentin tilaa (muuttujia), jota voidaan muuttaa esim. sovelluskehittimen avulla ja näin vaikuttaa komponentin toimintaan.
- Metodit (methods). Komponentit sisältävät normaalin luokan tavoin metodeja, joiden kuvaukset saadaan sovelluskehittimen tietoon itsehavaintomekanismin avulla.
- Tapahtumaviestejä (events) tarvitaan, jotta komponentit pystyisivät keskustelemaan keskenään. Komponentit voidaan rekisteröidä kuuntelemaan muiden komponenttien lähettämiä viestejä. Komponentit voidaan ohjelmoida toteuttamaan tiettyjä viestirajapintoja, joiden avulla pystytään määrittelemään komponentit jotka voidaan liittää toisiinsa. Viestin lähetys tapahtuu normaalilla asynkronisella metodikutsulla, jonka parametrinä on viestiolio.
- Graafinen näkymä ja asettelu (visual presentation, layout). Java-komponentit käyttävät grafiikkaominaisuudet tarjoavaa AWT-API:a (Abstract Windowing Toolkit) tai JFC-luokkakirjastoa (Java Foundation Classes) vastaavalla tavalla kuin mikä tahansa Java-luokka.
- Pysyvyysmekanismin (persistence) avulla komponentti voi tallentaa ja lukea oman tilansa esim. levyltä.
- Räätälöinti (customization). Jotta komponentti olisi mahdollisimman monen hyödynnettävissä, tulee sen ominaisuuksien olla helposti muokattavissa. Tähän tarkoitukseen on olemassa ominaisuuseditoreita (property editors), jotka tarjoavat tietylle ominaisuustyypille soveltuvan muokkausliittymän. Komponentille voidaan määritellä myös täysin oma muokkausliittymä, jonka kautta voidaan muuttaa esim. useaa ominaisuutta yhtäaikaa, jollakin käyttäjäystävällisellä ja havainnollisella tavalla.
- Paketoinnin (packaging) standardointi helpottaa komponenttien levittämistä. JAR-standardi (Java Archive) määrittelee miten toisiinsa liittyvät tiedostot kootaan yhteen ja pakataan ZIP-pakkausalgoritmilla. Pakkaukseen lisätään myös meta-tiedosto, joka sisältää tietoa pakkauksen

sisällöstä. Sisällölle voidaan antaa myös elektroninen allekirjoitus, jonka avulla voidaan varmistaa komponenttien toimittaja, ja se että paketin sisältöä ei ole muutettu matkalla.

- Java 2-version myötä komponenteille voi määritellä myös hierarkian. Komponentti voi olla toisen komponentin "sisällä", jolloin tämä muita komponentteja sisältävä olio toimii kontekstina (kokoomakomponentti). Uusiin ominaisuuksiin sisältyy myös rajapinnat kontekstin tarjoamien palveluiden ajonaikaiseen tiedusteluun.

5.4 RMI

RMI (Remote Method Interface) on Java-ympäristöön tarkoitettu oliokutsunvälittäjä. Mitään erillistä rajapinnankuvauskieltä ei ole määritelty, koska RMI:ssä käytetään Java-kieleen kuuluvaa rajapintatyyppiä (interface). Oletuksena RMI ei ole CORBA-yhteensopiva, eikä sitä tällöin voida käyttää kuin Java-olioiden välillä. Merkittävin ero RMI:n ja CORBA:n välillä on niiden olioparametrien käsittelyssä. RMI:ssä voidaan metodin parametrinä määritelty olio välittää arvona (object by value), kun taas CORBA:ssa olioparametri on perinteisesti ollut olioviite (object by reference). CORBA 3.0-standardissa tämä eroavaisuus on kuitenkin korjattu. OMG on myös määritellyt Java-IDL -sidonnan, jonka avulla Java-rajapinnasta voidaan kuvata IDL-rajapinta. Sun ja IBM ovat valmistaneet esiversion tämä määrityksen mukaan. Toteutuksen avulla voidaan RMI:n protokollana käyttää IIOP:ta. Tämä luonnollisesti mahdollistaa yhteistoiminnan muiden CORBA-sovellusten kanssa, eikä rajapinnan määrittämisessä ole tarvittu IDL-osaamista.

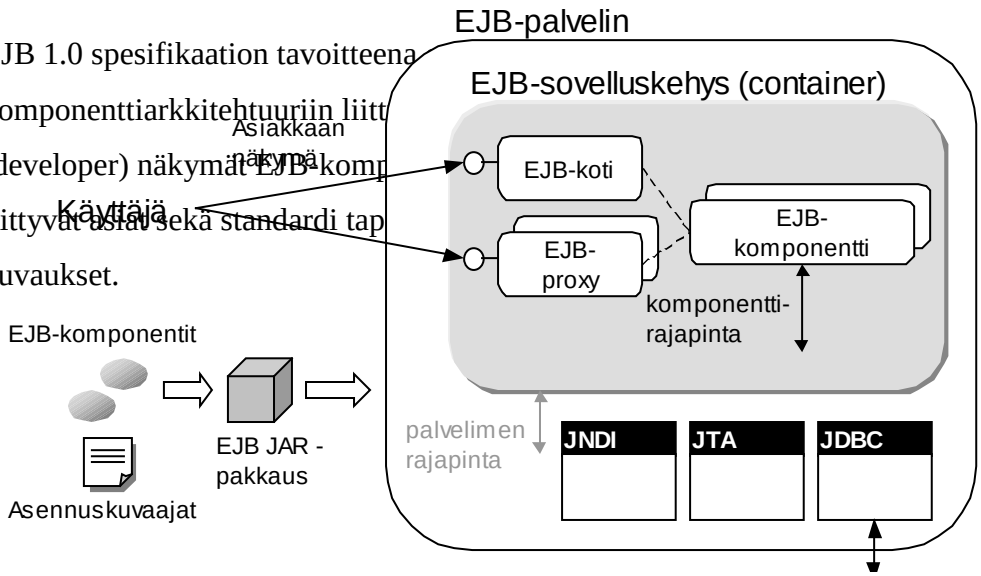
5.5 Enterprise JavaBeans

Enterprise JavaBeans (EJB) on standardi komponenttiarkkitehtuuri ja sovelluskehys hajautettujen, oliopohjaisten liiketoimintasovellusten toteuttamiseen Java-kielellä. EJB eroaa yksinkertaisemmasta JavaBeans-standardista siten, että JavaBeans-komponentit on tarkoitettu lähinnä

käyttöliittymäkomponenteiksi, kun taas EJB-komponentit sijaitsevat aina palvelimessa. EJB API kuuluu standardeihin lisäyksiin eikä näin ollen ole osa JDK:ta.

EJB mahdollistaa sovellusten kokoamisen eri toimittajien komponenteista. Java-tavukoodin siirrettävyyden ansiosta EJB-komponentti voidaan asentaa monelle eri alustalle ilman lähdekoodimuutoksia tai lähdekoodin uudelleen kääntämistä. Siirrettävyyden lisäksi EJB:n kantavana ideana on helpottaa liiketoiminnan kannalta kriittisten palvelinsovellusten toteuttamista. Sovellusten kehittäjien ei tarvitse enää hallita vaikeita alemman tason ohjelmointirajapintoja, koska esim. säikeiden hallinta (multi-threading), resurssien kierrätys (resource pooling) ja transaktiot (transaction) hoidetaan sovelluskehyksen toimesta. Toisaalta edistyneelle ohjelmoijalle jätetään mahdollisuus käyttää näitä ohjelmointirajapintoja myös suoraan. Vaikka EJB onkin Java-standardi ja kaikki rajapinnat on määritelty Java-rajapintoina, on EJB määritelty alusta pitäen myös CORBA-yhteensopivaksi. EJB-standardi onkin saanut lämpimän vastaanoton CORBA-valmistajien keskuudessa, koska CORBA-standardeista ei tällä hetkellä löydy komponenttiarkkitehtuuria. Osa nykyisistä EJB-toteutuksista perustuukin CORBA:n käyttöön oliokutsunvälittäjänä, toinen vaihtoehtohan olisi käyttää RMI-oliokutsunvälittäjää.

EJB 1.0 spesifikaation tavoitteena
komponenttiarkkitehtuuriin liitt
(developer) näkymät EJB-komp
liittyvät asiat sekä standardi tap
kuvaukset.



Kuva 11. EJB-arkkitehtuurin osa-alueet.

5.5.1 Roolit

EJB-arkkitehtuuri määrittelee kuusi eri roolia liittyen sovelluksen kehittämiseen ja asentamiseen. Jokainen näistä rooleista voi olla eri tahon toteuttama, joten EJB-standardin tärkeäksi tehtäväksi nousee rajapintojen ja sopimusten määrittely eri roolien välille.

- Komponenttikehittäjä (EJB provider) on tyypillisesti tietyn sovellusalueen asiantuntija. Hän kehittää uudelleenkäytettäviä komponentteja, jotka toteuttavat jonkin liiketoimintaan liittyvän osa-alueen. Komponenttikehittäjä ei yleensä ohjelmoi tietoturvaan, hajautukseen ja transaktioihin liittyviä asioita komponenttiin vaan luottaa siihen, että EJB-sovelluskehys tarjoaa nämä palvelut. Valmiit komponentit pakataan EJB JAR -tiedostoon, joka sisältää komponenttien tavukoodin, rajapinnat ja asennuskuvaajat (deployment descriptor).
- Sovelluskehittäjä (application assembler) on sovellusalansa asiantuntija, joka kehittää sovelluksia määrättyyn tarkoitukseen käyttäen EJB-komponentteja. Sovelluskehittäjä näkee komponentin rajapinnat, mutta muuten hänen ei tarvitse tietään miten komponentin toiminnallisuus on toteutettu. Komponenttien yhdistäminen ja räätälöinti tehdään tyypillisesti jollakin EJB-sovelluskehittimellä. Sovelluskehityksen tuotoksena saadaan joko uusia korkeammantason komponentteja tai valmiita sovelluksia. Sovelluskehittäjä toteuttaa yleensä myös sovellukseen sopivan asiakaskäyttöliittymän.
- Asentajan (deployer) roolina on asentaa komponentit ja niiden sovelluskehykset ajoympäristöön. Asentaja käyttää sovelluskehystoimittajan tarjoamia työkaluja liittääkseen komponentit haluttuun ajoympäristöön. Asentaja voi myöskin muuttaa EJB-komponentin asennuskuvaajassa olevia turva-asetuksia vastaamaan komponenttia käyttävän organisaation vaatimuksia.
- EJB-palvelintoimittaja (EJB server provider) tarjoaa EJB-palvelimen. Palvelintoimittajan erikoisalaa ovat olioiden hajautuksen, transaktioiden ja

muiden systeemitasonpalveluiden osaaminen. Tyypillisiä palvelintoimittajia ovat käyttöjärjestelmävalmistajat, CORBA ja muut välisovellusvalmistajat (middleware vendor) ja tietokantavalmistajat. Tyypillisesti EJB-palvelintoimittaja tarjoaa myös sovelluskehyksen ja siihen liittyvät työkalut. Jos palvelintoimittaja julkaisee palvelimensa sisäiset rajapinnat, voivat myös muut osapuolet kehittää palvelimeen sopivia sovelluskehyksiä. Palvelimeen voidaan asentaa yhtäaikaisesti useita erilaisia sovelluskehyksiä. Palvelimen rajapintoja ei ole vielä määritelty EJB 1.0 standardissa, joten ne ovat valmistajakohtaisia.

- EJB-sovelluskehystoimittaja (EJB container provider) tarjoaa konteinerin ts. sovelluskehyksen, joka eristää EJB-komponentit palvelimesta ja tarjoaa komponenteille standardin rajapinnan (component contract). Sovelluskehys tarjoaa yhteistyössä palvelimen kanssa skaalautuvan, turvallisen ja transaktiot omaavan komponenttialustan. Sovelluskehys liitetään varsinaisiin komponentteihin generoimalla osa sovelluskehyksen koodista komponenttien rajapintojen ja asennuskuvaajien mukaan. Haluttaessa sovelluskehys huolehtii komponentin tietojen tallentamisesta ja lataamisesta. Työkalujen avulla generoidaan koodia, joka siirtää komponentin muuttujien arvoja tietokannan ja komponentin välillä täysin automaattisesti. Tällöin komponentin suunnittelussa ei tarvitse ottaa kantaa siihen, mitä tietokantajärjestelmää käytetään (esim. relaatiokanta tai oliokanta) tai milloin tiedot viedään kantaan ja milloin ne haetaan sieltä takaisin. Relaatiotietokantojen yhteydessä käytetään standardia JDBC-rajapintaa (Java Database Connectivity). Yleensä sovelluskehykseen liittyy myös työkalut komponenttien ajonaikaiseen monitorointiin ja hallintaan.
- Järjestelmävalvojan roolina on tarkkailla järjestelmän ajonaikaista toimintaa.

5.5.2 Toiminta

Sovelluskehys luo ja hallitsee ajonaikaisia komponenttien instansseja. Käyttäjä ei siis koskaan kutsu komponentin metodeja suoraan, vaan metodikutsu välittyy aina sovelluskehyksen kautta. Komponentti voi olla joko sessio (session) tai yksilö (entity) tyyppinen. Sessio-olio syntyy ja kuolee ohjelman suorituksen aikana, ja se voi sisältää tilatietoa tai olla tilaton. Tilatietoa sisältävä sessio-olio on yleensä vain yhden asiakkaan käytössä. Yksilöolio kuvaa tyypillisesti tietokantaan talletettavaa tietoa. Yksilöolio voidaan hakea yksikäsitteisellä hakuavaimella (Primary Key), ja se voi hallita tilatietonsa itse tai antaa tehtävän sovelluskehyksen harteille.

Asiakkaan näkymä syntyy komponentin rajapinnasta ja kotirajapinnasta (home interface). Sovelluskehyksen tehtävänä on rekisteröidä asennettavan komponentin kotirajapinta nimipalveluun, josta asiakas voi hakea sen nimen perusteella. EJB-standardin mukaan nimipalvelua käytetään JNDI-rajapinnan (Java Naming and Directory Interface) kautta. JNDI määrittelee yhteisen rajapinnan eri nimipalvelutoteutuksille (esim. CORBA-nimipalvelu, RMI-rekisteri). Haetun kotirajapinnan avulla käyttäjä voi luoda uuden olion, tai etsiä jo olemassa olevia olioita. Kotirajapinta on siis laajennettu tehdasrajapinta. Komponentin varsinainen rajapinta määrittelee ne liiketoimintametodit, joita asiakas voi käyttää. Asiakas voi olla yhtä hyvin myös toinen EJB-komponentti. Transaktioiden hallinta tapahtuu JTA-rajapinnan (Java Transaction Api) kautta.

Sovelluskehyksen ja komponentin välillä noudatetaan komponentin tyypistä riippuvaa sopimusta (component contract). Sopimus toteutuu, kun komponentti toteuttaa määrätyn rajapinnan ja sovelluskehys tarjoaa komponentille oman rajapintansa (context interface).

6 ÄLYVERKKOTOTEUTUS

Älyverkkototeutus kostuu palvelunhallintapisteestä (luonti-, testaus- ja hallintaympäristö) ja palvelunohjauspisteestä (ajoympäristö). Työn tämä osio sisältää älyverkkototeutuksen kuvauksen.

6.1 TOVE-projekti

Älyverkkototeutus on osa laajempaan TOVE -projektia (Toimialaverkot tai Transparent Object oriented Virtual Exchance). TOVE on vuonna 1996 alkanut TEKES:in (Tekniikan edistämiskeskus) ja teollisuuskumppaneiden rahoittama kolmivuotinen tutkimushanke, jossa kehitetään laajakaistaisen ISDN-kytkimen (B-ISDN, Broadband Integrated Service Digital Network) ohjelmistoa. Ohjelmisto koostuu C++ -protokollasovelluskehyksestä nimeltään OVOPS++ (Object-oriented Virtual Operations System) ja sen avulla toteutetuista merkinantoprotokollista, kytkimen ohjausosasta ja älyverkko-osasta. Projektin toteutuksesta vastaa Teknillisen korkeakoulun tietoliikenneohjelmistojen ja multimedian laboratorio. Projektiin on osallistunut myös VTT (Valtion teknillinen tutkimuskeskus), jonka FSR -kytkentäarkkitehtuurin (Frame Synchronized Ring) mukaista laitetta on käytetty ATM-kytkimenä.

6.2 Tavoite ja menetelmät

TOVE-älyverkkototeutuksessa on ollut tavoitteena kokeilla ja hyödyntää uusinta olioteknologiaa laajennettavan ja yksinkertaisen älyverkkoympäristön luomiseksi. Älyverkkoympäristön peruslähtökohtia ovat olleet palveluiden siirrettävyys, visuaalisuus, modulaarisuus, yksinkertainen rakenne sekä yhteistoiminta aikaisemmin toteutetun palvelunkytkentäpisteen kanssa. Näiden reunaehtojen perusteella valittiin toteutuskieleksi Java. Toteutusten rakenne ja palvelut määriteltiin komponenttien avulla. Komponentit jaettiin käyttöliittymäkomponentteihin ja logiikkakomponentteihin. Lisäksi määriteltiin komponentteja tukeva geneerinen DCF -sovelluskehys (Distributed Component

Framework). Palvelunkytkentäpisteen ja palvelunohjauspisteen välinen liikenne päätettiin toteuttaa CORBA:n avulla.

6.3 B-SSP

TOVE-laajakaistapalvelunkytkentäpisteen (B-SSP, Broadband Service Switching Point) ajoympäristönä toimii Linux ja OVOPS++ -protokollasovelluskehys. B-SSP:n puhelunohjaustoiminto sisältää CS-1 -standardin mukaiset puhelumallit aloittavalle ja lähtevälle puolelle. Puhelunohjaustoimintoon voidaan liittää B-ISDN -käyttäjämerkinantoprotokollia (UNI, User Network Interface) ja tulevaisuudessa myös verkkomerkinantoprotokollia (NNI, Network-Network Interface). Puhelumallin liittäminen erilaisiin merkinantoprotokolliin on muodostunut varsin haastavaksi tehtäväksi. Puhelumallin tulisi esittää yleistä puhelumallia, mutta samalla se joutuu käsittelemään merkinantoprotokollien yksilöllisiä primitiivejä ja tietoelementtejä. Samoin merkinantoprotokollien eri viestisekvenssit vaikuttavat väistämättä puhelumallin toteutukseen. Ratkaisuksi on esitetty ylimääräisen välitason toteuttamista, joka piilottaisi merkinantoprotokollien toiminnan puhelumallilta.

Puhelunohjaustoiminto on myös yhteydessä kytkentätoimintoon, joka ohjaa eri tyyppisiä kytkentäkenttiä. Myös tässä tapauksessa puhelumalli on päätetty eristää välitasolla varsinaisesta kytkentäkentästä. Kytkentätoimintoon liittyy myös puhelun reitittämiseen tarvittavat ominaisuudet.

Palvelunkytkentätoiminto on liitetty palvelunohjauspisteeseen CORBA:n välityksellä samoin kuin puhelunohjaustoiminnon hallintarajapinta. Hallintarajapinnan kautta voidaan aktivoida havaintopisteiden liipaisimia ja tarkkailla niiden tilaa.

6.4 DCF-sovelluskehys

DCF on graafisesti hallittavien, hierarkkisten, komponenttipohjaisten ja hajautettujen palveluiden luonti- ja ajoympäristöjen toteuttamiseen tarkoitettu sovelluskehys. DCF on toteutettu Java-kielellä, ja se noudattaa soveltuvilta osin JavaBeans-määritystä. DCF erottaa palvelun logiikan ja näkymän erillisiksi osa-alueiksi (MVC-suunnittelumalli). Kuvussa 12 on esitetty DCF:n rakenne. DCF:n alimman osan muodostaa Java-alusta, joka tarjoaa jo itsessään useita tarvittavia palveluita. Seuraava kerros, DCF-ympäristö, rakentuu Java 1.1 -ohjelmointirajapintojen varaan tarjoten joitakin lisäpalveluita. /19/

6.4.1 DCF-ympäristö

Jo DCF:n suunnitteluvaiheen alussa oli selvää, että Javan peruskirjastojen tarjoamat palvelut eivät riitä ja että lisäpalveluiden tekeminen projektin puitteissa olisi resurssien haaskaamista. Vuoden 1998 alussa EJB-standardista oli jo joitakin tietoja, mutta standardi ei ollut vielä lopullinen ja yhtään sen toteuttavaa palvelinta ei ollut olemassa. Niinpä DCF:n toteutuksessa päätettiin pysyä JavaBeans -standardin puitteissa tiedostaen kuitenkin, että EJB:n mukainen ratkaisu olisi ihanteellinen tähän käyttöön. DCF-ympäristön tarjoamat palvelut liittyvät logiikkakomponenttien hallitsemiseen, tallentamiseen ja lataamiseen mutta mitään EJB-palvelimen kaltaisia raskaansarjan palveluita se ei tarjoa.

Logiikkakomponentit on koottu puumaiseen hierarkiaan fasadin taakse, joka piilottaa niiden yksityiskohdat. Fasadi ja siihen liittyvä RMI-rajapinta määrittelee yhtenäisen ja muuttumattoman rajapinnan näille komponenteille. Osa rajapinnan metodeista on dynaamisia, esim. action-metodi, joka sisältää parametrinä olioavaimen, operaation nimen ja sen parametrit. Fasadin takana ko. kutsu saa aikaan varsinaisen metodikutsun, joka on koottu operaation nimestä ja parametreistä dynaamisesti. Kutsu ohjataan olioavaimen mukaiseen

komponenttiin. Ympäristö, jossa ei ole tarvetta komponenttien tilan seuraamiselle tai muuttamiselle, voi kytkeä hallintarajapinnan pois käytöstä.

6.4.2 Komponentit ja viesti

DCF-komponentit noudattavat DCF-sovelluskehyksen sääntöjä ja nauttivat sen tarjoamista palveluista. Varsinainen käyttäjän haluama palvelulogiikka rakentuu komponenttien muodostamasta ketjusta. Ketjun rakentaminen ei vaadi koodin generointia, vaan se tehdään dynaamisesti. Tämä erottaa DCF:n monista muista JavaBeans-sovelluskehittimistä, jotka pohjautuvat lähdekoodin generointiin ja kääntämiseen. Ketjun läpi kulkee viesti, joka vierailee vuoron perään ketjuun kuuluvissa komponenteissa kuljettaen niiden välillä tietoja. Viestin sisään voidaan tallentaa kaikkia Java-kielen olioita. Informaatioelementteinä toimivat oliot tallennetaan ja haetaan niille annettujen nimien perusteella.

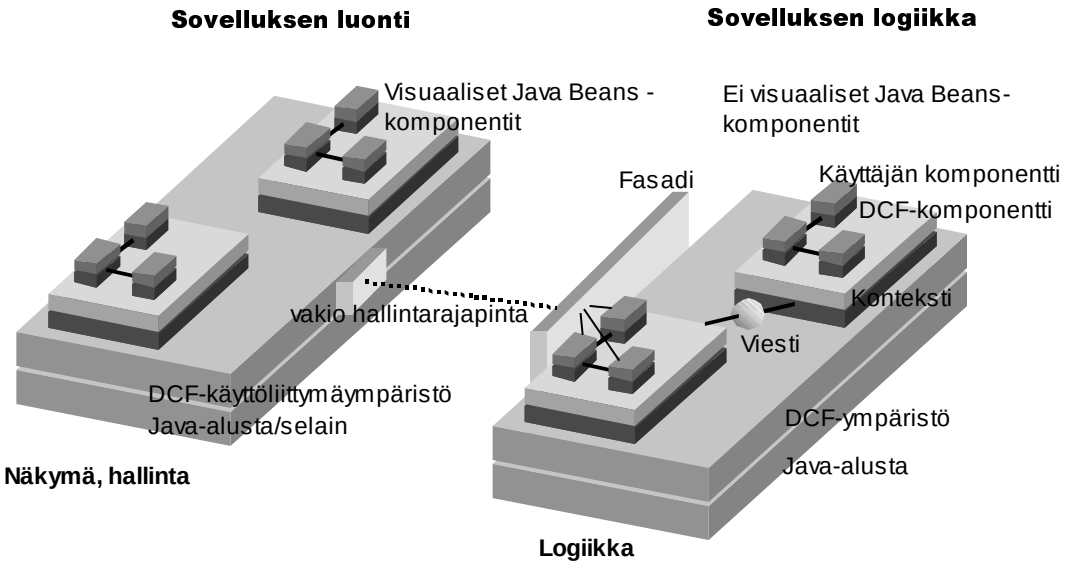
Komponentti voi myös muodostaa kontekstin, jolloin ko. komponentin sisällä voi olla useampia lapsikomponentteja. Näin suurestakin komponenttimäärästä muodostuu hallittava ja selkeä kokonaisuus. Käyttäjä voi käyttää valmiita komponentteja, tai sitten hän voi ohjelmoida tarvitsemansa komponentit Javalla noudattaen DCF:n ja JavaBeans-määrityksen sääntöjä.

6.4.3 Käyttöliittymä

Käyttöliittymä on erillinen osuus, jota voidaan käyttää sovelluksena tai sovelmana. Sitä voidaan ajaa eri koneessa, kuin missä logiikkakomponentit sijaitsevat. Käyttöliittymä käynnistetään vain silloin, kun logiikkakomponentteihin halutaan tehdä muutoksia. Käyttöliittymää sisältää takaisinkutsunrajapinnan, jonka kautta logiikkakomponentit voivat lähettää ilmoituksia käyttöliittymälle. Tämä ominaisuus on käyttökelpoinen varsinkin testaustilanteessa.

Komponenttikehittäjä voi päättää miltä logiikkakomponentit näyttävät määrittelemällä niille yksilölliset käyttöliittymäkomponentit. Tämä mahdollistaa hyvinkin seikkaperäisen ja visuaalisen palvelukuvauksen. Liitteessä 1 on kuva käyttöliittymästä. Komponenteilla voi olla yksi tai useampia tulo- ja lähtöpisteitä, jotka näkyvät halutulla tavalla käyttöliittymässä.

Kontekstikomponentille määritellään sallitut lapsikomponentit, jotka näkyvät työkalurivillä ja ovat näin käyttäjän valittavissa. Käyttäjä voi myös rakentaa uusia komponentteja yhdistämällä jo olemassa olevia komponentteja yhdeksi korkeantason kontekstikomponentiksi.

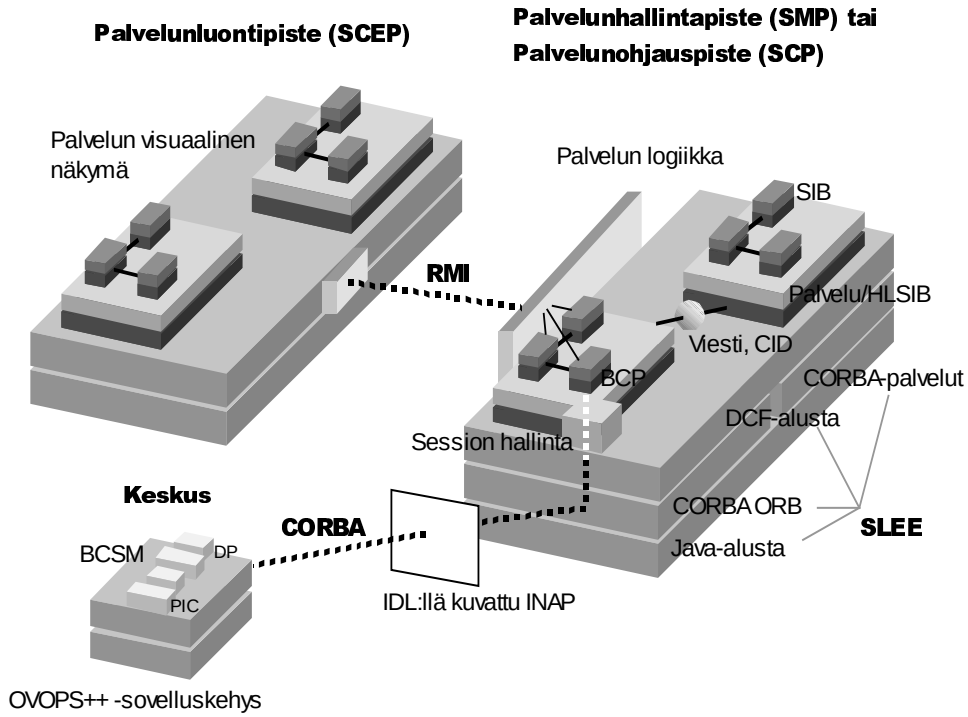


Kuva 12. DCF-sovelluskehyksen rakenne.

6.5 Palvelunhallinta- ja palvelunohjauspiste

Älyverkkopalveluiden luonti-, hallinta- ja ajoympäristö on tehty DCF-sovelluskehyksen avulla. Kuvassa 13 näkyy sekä sovelluskehys että älyverkkoympäristö. DCF-komponentit kuvaavat käsitteellisesti SIB:ejä ja muita älyverkkopalvelukuvauksessa tarvittavia osia. Komponenttihierarkian

juurena toimii joko hallinta- tai ajoympäristöä kuvaava kontekstikomponentti. Seuraavalla tasolla on palveluryhmäkonteksteja, jotka sisältävät eri palveluiden tarjoajia. Palveluryhmä kuvaa tiettyä älyverkkopalvelua, esimerkiksi ilmaispuhelupalvelua. Palveluryhmä erotetaan määritetyllä palveluavaimella, jonka SSP liittää lähettämäänsä ensimmäiseen INAP-sanomaan. Palveluntarjoaja taas tunnistetaan INAP-sanomassa olevan kohdenumeron avulla. Kun ensimmäinen INAP-sanoma saapuu, luodaan sessio-olio, joka kuvaa SSP:n ja palvelulogiikan välistä vuoropuhelua. Sessio-olion tehtävä on tallentaa tilatietoa ja muuntaa INAP-sanomat DCF-viesteiksi. Tämän rakenteen ansiosta protokollan vaihtaminen on erittäin helppoa, eikä se vaikuta palvelukomponenttien toimintaan. Reunaehtona on luonnollisesti se, että protokollan sanomista saadaan palvelulogiikan tarvitsemat tiedot. /20/



Kuva 13. DCF:n avulla rakennettu IN-toteutus.

6.5.1 Palveluesimerkki

Liitteen 1 kuva esittää palvelun tarjoajan yksilöllistä palvelulogiikkaa. Palvelun tarkoituksena on siirtää esim. kauppaan soitettu puhelu johonkin toiseen numeroon, jos kauppa ei ole auki (esim. viikonloppu). Esimerkissä kaupan numero ei ole ns. suoranumero vaan jokin erikoisnumero, joten numeron muunnos tehdään aina. Käyttäjä ei siis tiedä kaupan suoraa numeroa, joten mahdollinen suoran numeron vaihtuminen ei edellytä yhteystietojen päivittämistä. Lisäksi palvelun tarjoaja haluaa laskea tapaukset, jolloin numero on varattu.

Seuraavassa on esitelty palvelussa näkyvät SIB:it:

- Korkeammantason SIB (Higher Level SIB) on CS-2 -standardissa esiintyvä kokoama SIB, joka sisältää muita SIB:ejä.
- Vertailu SIB:iä (Comapere SIB) voidaan käyttää haarautumispäätöksissä. Ehtona voidaan käyttää esim. kellon aikaa tai viikon päivää.
- Algoritmi SIB:iä käytetään laskurina.
- Raportoinnin pyyntö SIB (Request Report BCSM Event). Tällä pyydetään palvelunkytkentäpistettä ilmoittamaan puhelumallissa tapahtuvista muutoksista.
- Muunnos SIB:n (translate) avulla tehdään kohdenumeron (B-numeron) muunnos.
- Vuoropuhelun lopetus SIB (End Dialog).

BCP on jaettu kahteen osaan: tulevaan ja lähtevään. Tulevaan BCP-komponenttiin on liitetty POI-lähdöt, jotka kuvaavat puhelumallin eri kohtia. Lähtevään BCP -komponenttiin on taas liitetty POR-sisäänmenot, jotka kuvaavat paluuta palvelulogiikasta puhelunmuodostukseen. Näiden väliin on koottu SIB:ejä, jotka muodostavat halutun palvelun.

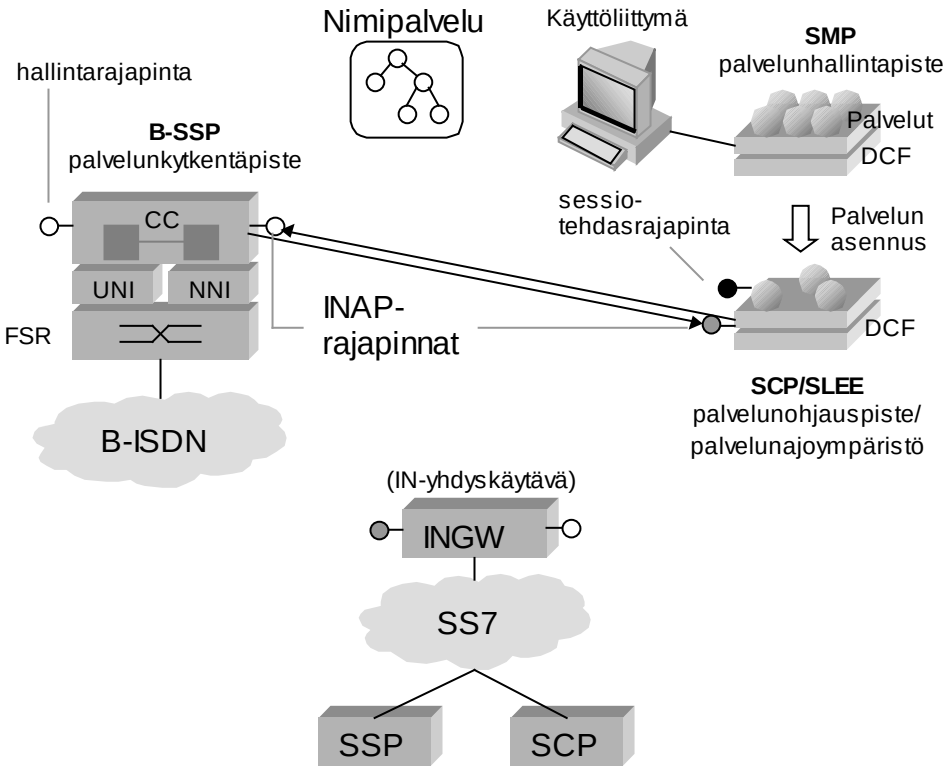
Liitteessä 1 olevassa esimerkissä "Request Report BCSM Event" -SIB:in avulla pyydetään puhelumallia ilmoittamaan numero varattu -tapahtumasta. Seuraavana on korkeammantason SIB, joka sisältää vertailu SIB:ejä (liite 2). Niihin on määritelty aukiolon päivärajat. Vertailujen tuloksen perusteella tehdään numeronmuunnos ja jatketaan puhelunmuodostusta uudella kohdenumerolla. Algoritmi SIB:iä käytetään numero varattu -tilanteiden laskemiseen. Kun laskenta on suoritettu, katkaistaan palvelulogiikan hallintayhteys "End Dialog" -SIB:illä. Tämän jälkeen palvelulogiikka ei voi enää ohjata puhelua, eikä se saa puhelumallilta tapahtumaviestejä.

6.5.2 INAP-rajapinta

INAP -rajapinta ja sen toiminnallisuus on toteutettu OMG:n tietoliikennetyöryhmän "Interworking between CORBA and Intelligent Networks Systems" -dokumentin mukaisesti. Dokumentti määrittelee sen, miten INAP -protokolla kuvataan IDL-rajapinnoiksi (specification translation) ja miten TCAP -vuoropuhelu kuvataan CORBA-metodikutsuina (interaction translation). Tämän määrityksen pohjalta voidaan rakentaa esim. IN -yhdyskäytävä (IN gateway), jonka avulla perinteiset älyverkkopisteet voidaan yhdistää uusiin CORBA-pohjaisiin toteutuksiin /14/. IN-yhdyskäytävän toteuttamista on tutkittu myös TOVE-projektissa, mutta toteutukseen liittyy yhä joitakin teknisiä ongelmia, kuten viestipalvelutoteutusten puuttuminen ja ASN.1-työkalujen puutteet. Nykyisessä TOVE-toteutuksessa sekä SSP että SCP ovat CORBA-toteutuksia, joten näiden välinen kommunikointi hoituu suoraan CORBA:n avulla ilman yhdyskäytävää (kuva 14).

Standardin määrittelemään interaktiologiikkaan liittyy tehdasrajapinta, joka on rekisteröity nimipalveluun globaalilla osoitteella (GT). Palvelua tarvitseva asiakas hakee nimipalvelusta tehtaan viitteen ja pyytä siltä uutta INAP -instanssia, välittäen samalla oman olioviitteensä. Välitetty asiakasrajapinta toimii vastapuolena, kun palvelun vuoropuhelu alkaa. /14/

INAP-sanomien välityksen tulee luonnollisesti olla asynkronista. CORBA-oliokutsut ovat kuitenkin oletuksena synkronisia, joten tarvitaan luotettavan asynkronisen kommunikoinnin mahdollistavaa sanomapalvelua. Näyttää siltä, että sanomapalvelutoteutukset realisoituvat vasta CORBA 3.0 toteutusten myötä, joten tältä osin on tyydyttävä väliaikaiseen ratkaisuun. Tässä ratkaisussa käytetään CORBA:n dynaamista kutsurajapintaa (DII, Dynamic Invocation Interface), joka mahdollistaa varmistamattomien asynkronisten viestien lähettämisen. Tämä ratkaisu ei ole kovinkaan käyttökelpoinen, koska viestin perille menosta ei voida olla varmoja ja DII-rajapinnan käyttö on erittäin hankalaa. DII:n käyttö aiheuttaa myös selkeän nopeusongelman, koska kaikki kutsut rakennetaan dynaamisesti.



Kuva 14. TOVE IN -ympäristö.

6.5.3 IDL ja INAP

Kuvassa 15 on esitetty yhden yksinkertaistetun INAP-operaation IDL-vastine. ASN.1-perustyyppien muunnos IDL-tyyppeihin on määritelty Open Group:in JIDM-dokumentissa (Joint Inter-Domain Management). INAP-sanomien parametrit on koottu IDL:n struct-rakenteeseen. Tämän parametrien lisäksi CORBA-kutsussa on konteksti-parametri, jossa kuljetetaan TCAP-vuoropuheluun liittyviä tietoja, kuten vuoropuhelun tunniste ja operaation numero. INAP-virhetilanteet kuvataan IDL-metodeissa poikkeuksina.

ASN.1

```
InitialDPArg ::= SEQUENCE {
  serviceKey [0] ServiceKey,
  bnumber    [1] CalledPartyNumber
}

initialDP OPERATION ::= {
  ARGUMENT InitialDPArg
  ERRORS     { missingCustomerRecord |
               systemFailure
             }
  CODE       local : 1
  }
```

IDL

```
struct InitialDPArgType {
  ServiceKeyType serviceKey;
  CalledPartyNumberType bnumber;
};

void initialDP( in InitialDpArgType arg,
   inout TcContext ctx )
     raises MissingCustomerRecord,
            SystemFailure;
```

Kuva 15. INAP-operaation IDL-vastine.

6.6 Mahdollisuudet ja jatkokehitys

Älyverkkoympäristöön liittyviä jatkokehitystarpeita: uusien SIB-komponenttien toteuttaminen, INAP-rajapinnan laajentaminen, uusien esimerkkipalveluiden toteutus ja tietokantaliityntä. Älyverkkoympäristö soveltuu tällä hetkellä parhaiten palveluiden kehittämiseen ja testaamiseen.

DCF:n rakenne tarjoaa useita tulevaisuuden laajennusmahdollisuuksia. Palvelut voitaisiin hajauttaa moniin eri pisteisiin ja osa palvelulogiikasta voitaisiin siirtää suoritettavaksi käyttäjän koneessa. DCF-käyttöliittymä on ensisijaisesti ajateltu operaattorin käyttöön, mutta vaadittavilla tietoturvalisäyksillä sitä

voidaan tarjota suoraan palvelun tarjoajalle ja palvelun käyttäjälle. Tällöin näkymä palveluun olisi todennäköisesti erilainen kuin operaattorin näkymä. Näkymän vaihto on helppo toteuttaa, koska se on erotettu palvelulogiikasta ja näkymiä voi olla useampia. Hierarkkisen ja komponenttipohjaisen lähestymistavan ansiosta DCF-malleihin on helppo lisätä uusi tasoja, esimerkkinä tilakonetason lisääminen IN-palvelukuvaukseen.

Palvelunohjauspisteen käyttämän protokollan vaihtaminen on helppoa, koska se on erotettu varsinaisesta palvelulogiikasta. Protokollaksi voitaisiin vaihtaa B-INAP (Broadband INAP), kun sellainen tulevaisuudessa standardoidaan. Langattomien viestimien palvelualustaan voitaisiin puolestaan liittää WAP-protokolla (Wireless Application Protocol).

Kaikkein kiehtovin ja realistisin jatkokehityssuunta on DCF:n ja sitä kautta TOVE-älyverkkoympäristön mukauttamien EJB-standardin mukaiseksi. Tällä ratkaisulla kaikki alemmantason palvelut saataisiin EJB-sovelluspalvelimesta (application server) ja EJB-sovelluskehys hoitaisi tietokantaoperaatiot. Näin voitaisiin keskittyä loppukäyttäjille oleellisten lisäarvopalveluiden toteuttamiseen standardilla tavalla. EJB varmistaisi myös palveluiden siirrettävyyden ja skaalautuvuuden, koska useat valmistajat tulevat tarjoamaan eri käyttäjämäärille tarkoitettuja (ja eri hintaisia) EJB-yhteensopivia sovelluspalvelimia.

7 TULOKSET

Valmistunut TOVE-älyverkkoympäristön prototyyppi koostuu yhdistetystä palvelunluonti- ja palvelunhallintapisteestä (SCE/SMP) ja siihen liittyvästä käyttöliittymästä. Palvelunhallintapisteeseen sisältyy valmiita SIB palvelukomponentteja, joiden avulla voidaan rakentaa uusia palveluita. Palvelunhallintapistettä voidaan käyttää myös palvelun testaukseen. Älyverkkoympäristöön kuuluu myös palvelunohjauspiste (SCP), jossa palvelut varsinaisesti suoritetaan. Palvelunohjauspiste liittyy palvelunkytkentäpisteeseen INAP:ia jäljittelevän IDL-rajapinnan kautta.

7.1 Kokemukset ja suositukset

Käytännöntoteutuksen tärkeitä tuloksia ovat uusien olioteknologioiden soveltamisesta syntyneet kokemukset ja suositukset. Ohjelmien jakaminen komponentteihin ja moduuleihin on osoittautunut erittäin hyväksi tavaksi selkeyttää ohjelmien rakennetta. Tämä lähestymistapa vaatii pitkäjänteistä analyysi- ja suunnitteluvaihetta, joten niihin liittyviä menetelmiä tulisi kehittää ja opiskella ennen uuden olio-projektin aloittamista. Komponentteihin liittyy oleellisesti rajapintojen määrittely, joten suunnittelun ensimmäiseen vaiheeseen tulisi liittyä vaatimusmäärittelyjen ja käyttötapausten muuntaminen selkeiksi rajapinnoiksi. Rajapinnat ohjaavat vuorostaan komponenttien toteuttajia, joten he voivat toimia itsenäisesti ja rinnakkain. Komponentin toteuttajalle tulisi antaa suhteellisen vapaat kädet komponentin toteuttamiseksi. Hänen tulisi vastata yksityiskohtien suunnittelusta ja valmiin komponentin testauksesta hyvinkin itsenäisesti. Tässä lähestymistavassa nousee tärkeään rooliin ohjelman kokoaja, joka käyttää valmistuneita komponentteja. Hänen tulisi antaa jatkuvasti palautetta komponenttien yhteistoiminnasta ja korjata arkkitehtuurin suunnittelussa tapahtuneita virheitä ja epäloogisuuksia.

Sovelluskehyksellä on tärkeä rooli olio-projektin onnistumisessa. Parhaimmillaan se ohjaa kehitystä oikeaan suuntaan ja ehkäisee lyhytnäköisyyden aiheuttamia suunnitteluvirheitä. Pahimmillaan sovelluskehyksen iteratiivinen kehitystyö imee projektin voimavarat ja itse tuotetta ei saada valmiiksi. Siksi onkin suositeltavaa käyttää valmista sovelluskehystä jos sellainen vain suinkin löytyy.

Javalla on ollut suuri merkitys älyverkkoympäristön toteutuksessa. Ilman Javan dynaamisia ominaisuuksia projektin tavoitteita olisi jouduttu karsimaan. Ei pidä myöskään aliarvioida Javan vaikutusta tuottavuuteen. Kun ohjelmistokehittäjät ovat sinut ohjelmointikielensä kanssa, lisää se selkeästi tuottavuutta. Monet ovatkin löytäneet ohjelmoinnin innostuksen uudelleen siirtyessään vaikeasta C++ -kielestä Javan pariin. Javan etuihin lasketaan koodin todellinen siirrettävyys. Tässä kohtaa on paikallaan pieni varoituksen sana. Java-alustoissa on pieniä mutta kiusallisia toteutuseroja, joten koodin siirrettävyys ei ole aina sitä mitä luvataan. Jos jotain yhdistelmää ei ole testattu, niin todennäköisyys virheelliselle toiminnalle on olemassa. Tässä puhutaan kuitenkin todella pienestä ongelmasta verrattuna esim. C++ -koodin kääntämiseen uudessa laiteympäristössä.

8 JOHTOPÄÄTÖKSET

Uudet olioteknologiat, varsinkin komponentit, CORBA ja Java, ovat tuoneet uusia vaikutteita ohjelmistokehitykseen nopealla aikataululla. Teknologiat ovat jo osaksi lunastaneet niihin asetettuja toiveita, mutta kehitys on yhä käynnissä. Tämä näkyy yhtä hyvin standardien kuin toteutustenkin jatkuvana kehittymisenä. Varsinkin CORBA-maailmassa näyttää siltä, että toteutukset eivät pysy uusien standardien tasolla. Tämä voi muodostua kohtalonkysymykseksi OMG:n standardoinnille. Jos asiakkaat eivät saa tarpeeksi nopeasti standardeja, laadukkaita ja suorituskykyisiä tuotteita, niin houkutus siirtyä epästandardien mutta toimivien tuotteiden käyttäjäksi on suuri. Onneksi Internetin suuri suosio ja levinneisyys on osaltaan vaikuttanut avoimien standardien aatteen leviämiseen ja hyväksymiseen. Tästä hyvänä esimerkkinä on Linux ja Java, joiden esille marssi olisi voinut epäonnistua täysin ilman Internetin tuomaa nostetta.

Ohjelmistokehitys on muuttumassa ja se vaikuttaa väistämättä myös tietoliikenneohjelmistoihin. Tietoliikenneohjelmistot ovat perinteisesti olleet suljettuja ja valmistajakohtaisia. Laitteiden väliset protokollat on standardoitu, mutta ajatus siitä, että esim. oman protokollapinon sisällä olisi muiden valmistajien komponentteja, on ollut vieras. Tähän on vaikuttanut varmasti se, että täysin oma ohjelmisto on ollut kilpailuetu. Älyverkkostandardointi on omalta osaltaan viitoittanut uutta suuntaa. Protokollien lisäksi olisi houkuttelevaa standardoida myös siirrettäviä palveluita. Tämä tarkoittaisi käytännössä palvelunajoympäristön standardointia tai jonkinlaisen palvelunkuvausvälikielen kehittämistä, jota eri ajoympäristöt ymmärtäisivät. Tietoliikenneteollisuuden vaihtoehtoina on kehittää omat standardinsa tai suunnata katseensa tietotekniikan standardoinnin puoleen. Tietoliikenneohjelmistoissa vaatimukset ovat aina olleet kovia. Isojen järjestelmien on pitänyt olla luotettavia, skaalautuvia ja hallittavia. Osittain

Internetin, verkottumisen ja asiakas-palvelin -arkkitehtuurin myötä nämä vaatimukset ovat siirtyneet myös muihin tietotekniikkasovelluksiin. Suosittujen web-palveluiden tulee mukautua suuriin käyttäjämääriin, ja niiden tulee toimia 24 tuntia vuorokaudessa. Nämä vaatimukset ovat vaikuttaneet suuresti Javan ja CORBA:n kehitykseen. Uusi ohjelmistoalue, sovelluspalvelimet, käyttääkin näitä uusia olioteknologioita tehokkaasti hyväkseen. Tässä kehityksessä tietoliikenneteollisuuden tulisikin siirtää omaa osaamistaan yleisten standardien vahvistukseksi ja samalla ottaa nämä standardit ja teknologiat käyttöön. Tämänkaltaista liikettä onkin ollut havaittavissa ja toivottavasti se yhä voimistuu.

Teoriaosassa kuvattu älyverkkoympäristö todistaa osaltaan olioteknologioiden vahvuutta ja käyttökelpoisuutta. Vastaavia ominaisuuksia sisältävää ohjelmistoa ei olisi voitu toteuttaa samassa ajassa vanhoilla menetelmillä. TOVE-älyverkkoympäristön tyyppisessä kokeilutoteutuksessa edut ovat kiistattomia. Toisaalta olioprojekteissa kannattaa pitää jalat massa ja suhtautua varauksella kaikkein uusimpiin työkaluihin. Olioteknologioiden ja -standardien epäkypsyys ja nopea kehittyminen saattaa aiheuttaa yllätyksiä, varsinkin jos lähdetään liikkeelle liian tiukalla aikataululla. Toteutusten ja standardien kypsyessä oliotulevaisuus näyttää kuitenkin erittäin lupaavalta!

LÄHTEET

1. Orfali, Robert. Harkey, Dan. Edwards, Jeri. Client/Server Programming with JAVA and CORBA Second Edition. John Wiley & Sons. 1998. ISBN 0-471-24578-X.
2. Gamma, Erich. Helm, Richard. Johnson, Ralph. Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Comapny. 1995. ISBN 0-201-63361-2.
3. Magedanz, Thomas. Popescu-Zeletin, Radu. Intelligent Networks: Basic Technology, Standards and Evolution. International Thompson computer press. 1996. ISBN 1-85032-293-7.
4. Venieris, Iakovos. Hussmann, Heinrich. Intelligent Broadband Networks. John Wiley & Sons. 1998. ISBN 0-471-98094-3.
5. Faynberg, Igor. Gabuzda, Lawrence. Kaplan, Marc. Shah, Nitin. The Intelligent Network Standards: Their Application to Services. McGraw Hill. 1997. ISBN 0-07-021422-0.
6. Puro, Panu. CORBA hajautetussa ohjelmistossa. Diplomityö. LTKK, tietotekniikan osasto. 1998.
7. ITU-T Recommendation Q.1200: Q-Series Intelligent Network Recommendation Architecture.
8. Mowbray, Thomas. Malveau, Raphael. CORBA Design Patterns. Johon Wiley & Sons, Inc. 1997. ISBN 0-471-15882-8.
9. Matena, Vlada. Hapner, Mark. Enterprise JavaBeans specification 1.0. Sun Microsystems Inc. 1998.
10. Remote Method Invocation (RMI) Specification, Revision 1.50 JDK 1.2. Sun Microsystems Inc. 1998.
11. Bellur, Umesh (Editor). CORBA Components. OMG. 1997. Orbos/97-11-24.
12. CORBA Component Imperatives. OMG. 1997. Orbos/97-05-25.
13. Cobb, Ed (Editor). CORBA Components. OMG. 1998. Orbos/98-12-02

14. Brennan, Rob (Editor). Interworking Between CORBA and Intelligent Networks System, Version 2.0 Revised RFP Submission. OMG. 1998. Telecom/98-08-11.
15. Hamilton, Graham (Editor). JavaBeans Specification 1.01. Sun Microsystems Inc. 1997.
16. Taligent. The Power of Frameworks. Addison-Wesley. 1995. ISBN 0-201-48348-3.
17. Black, Uyless. ISDN & SS7, Architectures for Digital Signaling Networks. Prentice Hall. 1997. ISBN 0-12-259193-6.
18. OMG. CORBA 3.0 announcement at Comdex Enterprise 98'/Object World Conference.
19. Nummisalo, Pasi. TOVE DCF. TKK. 1998.
20. Nummisalo, Pasi. TOVE SCE/SCP. TKK. 1998
21. Koponen, Petteri. Puro, Vesa-Matti. Räsänen, Juhana. Nummisalo, Pasi. Martikainen, Olli. Distribution of Switch Control. Proceedings of the Workshop on Broadband Network Intelligence , Espoo, Finland (Nov. 1996).