Komponenttiteknologia

Komponenttiteknologia on järjestelmien rakentamisen uusin aalto. Komponenttien avulla on mahdollisuus päästä tehokkaampaan sovelluskehitykseen ja jo tehdyn kehityksen uudelleenkäyttöön. Komponentit määritellään useimmiten oliotekniikalla mutta niiden toteutus ei välttämättä aina ole oliopohjainen.

Ensimmäiset laajemmin käytetyt yleiskäyttöiset komponentit saattoivat olla Visual Basic -ympäristöihin tarkoitetut VBX-komponentit. Joissakin sovelluskehittimissä oli ja on myös komponenttiominaisuuksia. Komponentit olivat usein hyvin käyttöliittymäläheisiä tai teknisiä rajapintoja käsitteleviä komponentteja, esimerkiksi taulukko- tai tietoliikennekomponentteja.

Komponenttien käyttöalue on laajentumassa myös palvelinpuolelle ja lähemmäksi liiketoimintakomponentteja. Yksittäisistä komponenteista on siirrytty komponenttiarkkitehtuureihin, jotka tarjoavat komponenttimallin ja liitettävyyden lisäksi ratkaisuja kommunikointiin, hajautukseen, siirrettävyyteen, transaktioiden käsittelyyn ja turvallisuuteen. Parhaiten tunnettuja komponentteja ovat Active/X-komponentit Microsoft-maailmassa, CORBA-komponentit ja Enterprise Java Beans-komponentit.

Komponenttien toteutus ja mallinnus hajautetuissa komponenttiarkkitehtuureissa tuo järjestelmäkehitykselle uusia haasteita. Kauan lupaillut uudelleenkäytettävät komponentit ovat olleet arkipäivää jo pitkään käyttöliittymäkomponenteissa, mutta palvelinpuolella teknologiat alkavat vasta nyt olla kypsiä näiden toteuttamiseen.

Tämän lehden sisältämien artikkelien on tarkoitus antaa lukijalle katsaus komponenttiteknologian nykytilaan.

Hannu Kokko

Hannu Kokko toimii johtavana konsulttina ja toimitusjohtajana olio- ja komponenttitekniikan alueella SysOpen Object Team Oy:ssä. Hän on Sytykkeen hallituksen jäsen.
Puh: 040 751 6232
Email: Hannu.Kokko@sysopen.fi

SYSTEEMITYÖ 1/2000

Komponenttiarkkitehtuurien vaikutus ohjelmistotuotantoon

Komponenttien hankintaan, hallintaan ja integrointiin liittyviä kysymyksiä

Pekka Kähkipuro,
SysOpen Oyj

Uudet komponenttiarkkitehtuurit, CORBA, COM+ ja EJB, ovat kypsymässä varteenotettavaksi keinoksi rakentaa tietojärjestelmiä. Ruusuisten lupausten takaa on kuitenkin löytynut iso joukko merkittäviä haasteita, ja monet uusiin arkkitehtuureihin pohjautuneet projektit ovatkin joutuneet vaikeuksiin. Keskeisenä tekijänä onnistuneissa projekteissa on ollut **uudenlainen ohjelmistotuotantoprosessi**. Tässä artikkelissa tarkastellaan komponenttiprojektien tyypillisiä vaikeuksia ja esitellään neljä keskeistä teesiä uudenlaisen ohjelmistotuotantoprosessin pohjaksi.

Ruusuisia lupauksia tulevaisuudesta

Uudet komponenttiarkkitehtuurit lupaavat merkittäviä parannuksia tietojärjestelmien toteuttajille. Keskeisiä lupauksia ovat:

- Uudelleenkäytettävyyden helpottuminen,
- Parempi tuki heterogeenisille ratkaisuille,
- Laajojen järjestelmien hallittavampi toteutus,
- Laajojen komponenttimarkkinoiden synty,
- IT:n kyky sopeutua nopeisiin muutoksiin.

Uudelleenkäytettävyyden helpottuminen perustuu ensisijaisesti komponenttien musta laatikko –ajatteluun, joka on helpompi omaksua kuin oliokielissä käytössä olevat perintämekanismit. Siinä on tavoitteena piilottaa komponentin käyttäjältä sen sisäinen rakenne ja tarjota ainoastaan joukko korkean tason rajapintoja komponenttien palveluiden kuvaamiseen.

Heterogeenisten ratkaisujen parempi tuki saadaan puolestaan aikaan kehittyneen infrastruktuurikerroksen avulla. Se peittää järjestelmänosien erilaisuudet ja tekee automaattisesti tarvittavat konversiot esimerkiksi tiedon esitystavassa ja käytetyissä protokollissa.

Laajojen järjestelmien hallintaan komponentti-infrastruktuuri tarjoaa uusia mahdollisuuksia: se on läsnä kaikkialla, joten sinne voidaan helposti upottaa järjestelmänhallinnan tarvitsemat toiminnot. Infrastruktuuri voi helpottaa myös merkittävästi tietoturvan toteuttamista. Riittää kun hankintaan sellainen versio infrastruktuurista, joka toteuttaa halutun turvatason. Sovelluskohtaisten lisäpiirteiden toteutus tapahtuu tyypillisesti callback-rajapinnoilla, joilla tarvittaessa korvataan infrastruktuurin oletustoimintoja tilannekohtaisesti.

Musta laatikko –uudelleenkäyttö ja kehittynyt infrastruktuuri tarkoittavat käytännössä sitä, että komponentit voidaan toimittaa binäärimuodossa ja ne voidaan asentaa lähes automaattisesti (“plug and play”). Nämä yhdessä antavat mahdollisuuden laajojen komponenttimarkkinoiden syntymiselle. Myyjän on helpompi luovuttaa komponentit muiden käyttöön, kun niitä ei tarvitse toimittaa lähdemuodossa, ja ostajan on helpompi ottaa muiden tekemiä komponentteja, kun niiden käyttöönotto ei edellytä suuria ponnisteluja. Tärkeä lisävaatimus on rajapintastandardien olemassalo, sillä muussa tapauksessa komponenttien yhteiskäyttö edellyttää aina myös yhdyskäytävien ohjelmointia tai ostajan ja myyjän keskinäistä koordinaatiota.

Ratkaisujen syntyminen entistä nopeammin perustuu kolmeen seikkaan. Ensinnäkin itse tehtyjä komponentteja voidaan käyttää uudelleen aiempaa paremmin. Toisaalta taas valmiskomponenttien olemassalo vähentää tarvetta toteutustyölle. Lisäksi komponentti-infrastruktuurin tarjoamat palvelut nopeuttavat ohjelmistotyötä merkittävästi. Eräs keskeinen menetelmä liiketoiminnan muutosten parempaan hallintaan onkin olemassaolevien komponenttien järjestäminen uudelleen kulloisenkin tilanteen mukaiseksi.

Todellisuus on osoittautunut kiviseksi poluksi

Käytännön komponenttiprojektit ovat osoittautuneet huomattavasti hankalammiksi kuin edellä kuvatut lupaukset antavat ymmärtää. Vain harvat organisaatiot ovat saaneet komponentit todelliseen hyötykäyttöön.

Uudelleenkäyttö on osoittautunut vähäiseksi verrattuna jopa perinteisiin monoliittisiin järjestelmiin. Eräs syy tähän on komponentti-infrastruktuurien nopea kehitys, joka on houkutellut tai pakottanut toteuttajat rakentamaan keskeisiä osia sovelluksesta moneen kertaan. Esimerkiksi CORBA-ratkaisuissa ollaan siirtymässä version 2.3 mukaisiin järjestelmiin ja käytännössä tämä tarkoittaa palveluiden keskeisten osien uudelleenjärjestelyä, vaikka varsinainen sovelluslogiikka pysyisikin samana. Toinen merkittävä syy uudelleenkäytön vähäisyyteen ovat väärällä tavalla suunnitellut komponenttirajat. Rinnakkaisissa projekteissa tai erillisissä organisaatioissa toteutetut komponentit ovat usein toiminnaltaan limittäisiä tai päällekkäisiä, ja niiden toteuttamat rajapinnat tarjoavat hankalasti käyttäviä kokonaisuuksia. Jopa komponentti-infrastruktuurien keskeiset toimittajat, kuten Microsoft ja Sun, ovat julkistaneet omien tuotteidensa kanssa hankalasti yhteensovitettavia komponenttirajapintoja. Eri sovellusalueille on vasta viime aikoina syntynyt standardointihankkeita, jota pyrkivät määrittelemään komponettipohjaisia arkkitehtuureja.

Heterogeenisuuden tuki on myös ollut rajallista. Keskeinen syy on jälleen komponenttiteknologian nopea kehitys, joka pakottaa valmistajat keskittymään vain muutamaan ympäristöön. Esimerkiksi Microsoftin DCOM-protokollan porttaus

keskeisiin Unix-ympäristöihin tapahtui kiusallisen myöhään tarvisijoiden kannalta – tämä on eräs keskeinen syy CORBA-arkkitehtuurin suosioon sellaisissa ratkaisuissa, joissa käytetään sekä NT- että Unix-ympäristöjä. Käytännössä heterogeenisuuden tukeminen on tarkoittanut sitä, että sovellusprojektissa rakennetaan tapauskohtaisia yhdyskäytäviä eri komponentti-infrastruktuurien välillä.

Komponenttipohjainen järjestelmä on usein vaativa hallinnan kannalta, sillä sovelluspalasten lisäksi järjestelmään kuuluu merkittävä joukko infrastruktuurin tarvitsemia lisäelementtejä. Monet infrastruktuurituotteet tarjoavat toki hallintavälineen, mutta näiden välineiden avulla ei tyypillisesti voi hallita muilla infrastruktuurituotteilla toteutettuja komponentteja. Lisäksi hallintaa tukevat rajapinnat on tyypillisesti jätetty standardoinnissa viimeiseksi, sillä niiden tarve on havaittu vasta, kun sovellustyöhön suunniteltuja rajapintoja toteutetaan. Esimerkiksi OMG ei ole vieläkään määritellyt CORBA-järjestelmille riittävää järjestelmänhallintaan tarkoitettua rajapintojen joukkoa. Tuloksena on joukko epäyhteensopivia tuotekohtaisia ratkaisuja.

Komponenttimarkkinat ovat vasta nyt heräämässä ja käytännössä komponenttiprojektit ovatkin toteuttaneet järjestelmiä alusta pitäen itse. Ongelmana on jälleen teknologian nopea kehitys, joka usein johtaa siihen, että pitkähkön kehitystyön vaativa komponentti on valmistuessaan teknisesti vanhentunut. Nopeasti toteutettavien komponenttien osalta tilanne on puolestaan sellainen, että komponentti on helpompi tehdä itse kuin etsiä jonkun muun tekemä ratkaisu, opetella sen käyttö, ja upottaa se osaksi omaa ratkaisua. Keskeisenä pulmana on ollut helppokäyttöisten komponenttivarastojen puuttuminen. Silloinkin kun komponenttivarasto on luotu, sitä harvoin käytetään tehokkaasti, sillä yrityksiltä tyypillisesti puuttuu varastoa ylläpitävä ja hoitava prosessi.

Edellä kuvattujen pulmien tuloksena monet komponenttiprojektit ovat epäonnistuneet pahan kerran. Onnistuneetkin projektit ovat usein myöhästyneet aikataulutavoitteistaan. Komponenttiprojekteille on lisäksi tyypillistä se, että järjestelmän keskeisiä osia toteutetaan useaan kertaan. Eräänä syynä on teknologiakehityksen oravanpyörä: halutaan käyttää aina viimeisintä versiota, jolloin versionvaihdon seurauksena rakennetaan osia järjestelmästä uudestaan ja selvitetetään lisäksi keskeneräisten tuotteiden virheitä. Toinen syy on komponenttipohjaisen suunnittelutyön laiminlyöminen: komponenttirajapintojen ja sovellusarkkitehtuurin suunnittelu on jätetty väliin tai se on tehty yhtä kevyesti kuin komponenttien sisäinen toteutus. Muitakin syitä löytyy: kokemattomuus uusissa ympäristöissä ja liiallinen usko toimittajien lupauksiin.

Keskeinen ongelma ei ole teknologiassa vaan ohjelmistotuotannossa

Tyypillinen reaktio epäonnistuneeseen komponenttiprojektiin on saman työn tekeminen uudestaan eri välineillä. Usein tällainen projekti onnistuukin, mutta pääsyynä ei ole parempi teknologia vaan se, että tekijäjoukko on oppinut ensimmäisen yrityksen virheistään. Tällä hetkellä kaikki keskeiset komponenttiteknologiat ovat riittävän kypsiä mittavien tietojärjestelmien toteuttamiseen, ja komponenttiprojektien ongelmalähteitä onkin etsittävä muualta: ohjelmistotuotantoprosessista. Keskeisiä pulmia ovat seuraavat:

- Perinteisten projektien järjestelmäajattelu,
- Tietokeskeinen arkkitehtuuri,
- Kehitysvälineiden vähäinen tuki komponenttien hallintaan,
- Sovellusprojektilähtöinen komponenttien hankinta.

Perinteinen ohjelmistotuotanto perustuu ajattelutapaan. jossa projektin tavoitteena on tuottaa kokonainen **järjestelmä** tai jokin rajattu osajärjestelmä. Järjestelmän ja ulkopuolisen maailman välinen raja on selkeä, ja projektin toimintaa ohjataan tämän rajan avulla: rajan ulkopuolelle ei mennä, ja projekti on valmis kun työ on edennyt valittuun rajaan asti. Tällainen järjestelmäajattelu on komponenttiarkkitehtuurissa ongelmallinen, sillä komponenttijärjestelmän “raja” on erittäin häilyvä – uudelleenkäyttöön pyrittäessä sama komponentti esiintyy lähes aina usean järjestelmän osana. Seurauksena on tyypillisesti projektin tavoitteiden ja työn kohteen muuttuminen projektin kuluessa. Kun komponentteja lisätään, poistetaan ja muutetaan, myös sovelluksen tavoitteet muuttuvat lennossa – useimmiten ne laajenevat merkittävästi. Tuloksena on hallitsematon projektin kulku ja merkittäviä aikataulu-ongelmia.

Perinteinen ohjelmistotuotanto perustuu usein **tietokeskeiseen** arkkitehtuuriin, jossa järjestelmän ytimenä käytetään tietokantaa ja sovellusten toiminta määritellään tiedon käsittelynä. Tällainen arkkitehtuuri johtaa vaikeuksiin komponenttien kanssa. Komponenttijärjestelmissä tyypillisesti kukin komponentti hallitsee omaa tietosisältöään, ja järjestelmän tietosisältö muodostuu komponenttien yhteistoiminnan perustella, esimerkiksi rajapintakuvauksien avulla. Tämän seurauksena komponenttien käyttö tietokeskeisen arkkitehtuurien kanssa yleensä johtaa redundanttiin tietoon ja siitä seuraaviin eheysongelmiin. Vaihtoehtoisesti tietokeskeinen arkkitehtuurinäkemys saattaa ohjata komponenttiprojektin ratkaisuun, jossa komponenttien roolina

Ohjelmistotuotantoprosessia on muutettava

Komponenttipohjaisten ohjelmistojen toteuttaminen edellyttää uudenlaista ohjelmistotuotantoprosessia. Tämän uudistuneen prosessin keskeiset vaatimukset voidaan esittää seuraavasti:

- Liiketoimintaprosessit sovellusprojektien lähtökohdaksi,
- Palveluarkkitehtuuria käytetään tietopohjaisen arkkitehtuurin asemesta,
- Komponenttivarastot keskeiseen asemaan,
- Komponenttien hankinta- ja hallintaprosessit erotetaan sovellustyöstä.

Seuraavassa käsitellään kutakin vaatimusta hiukan tarkemmin. On huomattava, että uudenlainen ohjelmistotuotantoprosessi muistuttaa merkittävästi oliopohjaisten järjestelmien rakentamistapaa, ja merkittävimmät onnistumiset komponenttisaralla ovatkin perustuneet hyvin omaksutun oliolähtöisen toteutusprosessin käyttöön. Tämä ei ole mikään sattuma, sillä komponenttiteknologia on perinyt keskeiset piirteet juuri oliomaailmasta.

Lähtökohtana liiketoimintaprosessit

Perinteinen järjestelmäajattelu joutuu vaikeuksiin komponenttiprojekteissa, joten käyttöön on otettava uudenlainen lähtökohta projekteille. Lupaavin vaihtoehto on määritellä **projektin rajat ja tavoitteet liiketoimintaprosessin avulla**: projektin tavoitteeksi asetetaan yhden liiketoimintaprosessin tarvitseman tietojärjestelmätuen rakentaminen. Projektin rajat määrittyvät tällöin selvästi: kaikkien toimenpiteiden täytyy edistää kohteena olevaa liiketoimintaprosessia. Esimerkiksi uuden infrastruktuuriversion käyttöönotto on perusteltua vain, jos se parantaa liiketoimintaprosessin tukea. Tällainen vaatimus ohjaa projektin osallistujia teknologiatyön piiristä sovellustyön pariin ja sovellukset valmistuvat nopeammin. Myös

projektin lopputulos on selkeä ja välittömästi mitattavissa: jos liiketoimintaprosessi pystyy toimimaan määritellyllä tavalla, tulos on saavutettu.

Liiketoimintaprosessien käyttö on kuitenkin osoittautunut ongelmalliseksi erityisesti palvelutoimialoilla, joissa fyysinen maailma asettaa varsin vähän rajoituksia. Prosesseja on vaikea yksilöidä, sillä ne muuttuvat jatkuvasti organisaation toiminnan mukana. Liiketoimintaprosessit ja tietotekniikka ovat myös jatkuvassa vuorovaikutuksessa: muutokset prosessissa heijastuvat teknologiaratkaisuissa ja teknologian mahdollisuudet puolestaan muuttavat prosessia. Tällaisessa tilanteessa voidaan käyttää **komponenttipohjaista jäsennystapaa**, jossa järjestelmä hahmotetaan joukkona komponentteja ja hankkeen tavoitteet määritellään niiden erilaisten liiketoimintaprosessien perusteella, jotka kyseisillä komponenteilla on mahdollista toteutettaa. Näitä prosesseja on yleensä useita, sillä tyypillisesti tavoitteena on hankkia mahdollisimman joustavia komponentteja liiketoiminnan nopeiden muutosten tueksi. Komponenttipohjainen jäsennystapa edellyttää tiivistä vuorovaikutusta liiketoiminta- ja teknologianäkökulmien välillä. Käytännössä tämä voi tarkoittaa esimerkiksi sitä, että ehdotetun komponenttijaon perusteella käydään uudelleen läpi liiketoimintavaatimukset ja etsitään muita mahdollisia tapoja jäsentää liiketoiminta löydetyn teknologiaratkaisun pohjalta.

Liiketoimintaprosessit muodostavat myös hyvän lähtökohdan komponenttirajojen löytymiselle. Jos useampi kuin yksi liiketoimintaprosessi tarvitsee jotain toiminnallisuutta, niin tyypillisesti tällainen toiminnallisuus kannattaa määritellä palveluksi, jonka toteuttaa erillinen komponentti. Komponentin rajapinta tulee tällöin määritellä kaikkien sitä tarvitsevien prosessien näkökulmasta.

Palveluarkkitehtuuri

Tietopohjainen arkkitehtuuri on komponenttijärjestelmissä korvattava **palveluarkkitehtuurilla**, jotta vältetään redundantin tiedon hallintaan liittyviltä vaikeuksilta. Tällöin järjestelmä jo suunnitteluvaiheessa jäsennellään joukkona palveluita, joita komponentit tarjoavat toisilleen. Oliopohjaiset suunnittelumenetelmät tarjoavat hyvän keinon palveluarkkitehtuurin hahmottamiseen. Palveluarkkitehtuurin keskeinen piirre on kunkin palvelun abstrahointi siten, että vain sen rajapinta on muiden järjestelmänosien tiedossa ja että sisäinen toteutustapa jätetään palvelun toteuttajan vastuulle. Käytännössä tämä tarkoittaa sitä, että monet palvelut pitävät sisällään pienen "loogisen tietokannan" pysyväksi tarkoitettua informaatiota varten. Komponentti-infrastruktuuri kuitenkin tarjoaa keinon hallita tämä tilanne: kunkin komponentin tila voidaan edelleen ohjata haluttuun paikkaan, esimerkiksi relaatiotietokantaan tai sovelluspalvelimen sisäiseen tallennusmuotoon. Tietojen tallentaminen, eheyden valvominen ja tietojen hakeminen ovat siirtyneet infrastruktuurin vastuulle.

Komponenttivarasto

Komponenttien uudelleenkäyttö projektin päättymisen jälkeen varmistetaan erillisellä **komponenttivarastolla**, jonne komponentin itsensä lisäksi talletetaan myös teknistä kuvaustietoa, versionhallintaan liittyvää tietoa, komponentin käyttörajoituksia, sidokset nykyisiin käyttäjiin, jne. Suunnittelijoilla ja sovelluskehittäjillä on suora pääsy tähän varastoon ja he voivat sieltä tutkia käytettävissä olevan komponenttijoukon antamia mahdollisuuksia. Parhaimmillaan komponenttivarasto on integroitu sovelluskehitysvälineistöön. Lisäksi komponenttivaraston on tarjottava ajonaikainen rajapinta, jonka avulla infrastruktuuri voi avustaa mm. komponenttien asennustyötä.

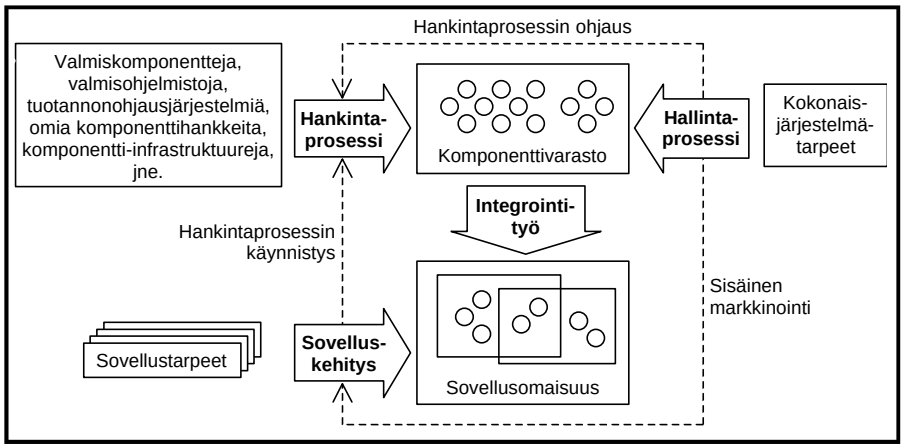
Komponettien hankinta ja hallinta

Komponenttien hankinta ja hallinta on erotettava sovelluskehityksestä, jotta saavutetaan toivottu uudelleenkäyttö. Jos sovellusprojektin aikana havaitaan tarve komponentille, tämä tarve on ohjattava erityiselle **hankintaprosessille**. Siinä kartoitetaan komponentin mahdollinen käytettävyys muissakin kuin kohteena olevissa liiketoimintaprosesseissa ja käytetään mahdollisia lisäkriteereitä, joita ovat esimerkiksi yhteensopivuusvaatimukset muiden komponenttien ja arkkitehtuurivisioiden kanssa. Hankintaprosessi voi myös päättää, ettei komponenttia hankita vaan että se toteutetaan itse. Hankintaprosessin tarkoituksena on kartuttaa organisaation **komponenttiomaisuutta** siten, että kulloinenkin sovellustarve tulee toteutettua tarkoituksenmukaisesti ("komponenttihankinnan imuperiaate").

Uuden komponentin käyttöönotto kuuluu myös hankintaprosessin tehtäviin. Tällöin komponentti julkistetaan komponenttivaraston kautta, siihen tehdään tarvittavat muutokset, ja lisäksi määritellään sen käyttöön liittyvät rajoitukset ja ohjeet. Tällainen alustava **integrointityö** on keskeisessä asemassa komponentin uudelleenkäytön kannalta – ilman sitä komponentti jää yleensä vain yhden projektin käyttöön. Kukin sovellusprojekti luonnollisesti jatkaa integrointityötä omalla sarallaan, jotta komponentti saadaan istutettua kulloinkin työn kohteena olevaan kokonaisuuteen.

Yksittäisten hankintaprojektien rinnalla elää komponenttien **hallintaprosessi**, jossa pyritään ylläpitämään ja kehittämään komponenttiomaisuutta siten, että sen tarjoama kokonaishyöty on mahdollisimman suuri investointeihin nähden. Hallintaprosessin tehtävänä on esimerkiksi määritellä komponenttikohtaisesti ylläpidon taso: mitä versioita tuetaan ja millä aikajänteellä komponentteja päivitetään.

Hallintaprosessissa määritellään myös **visio** tulevasta arkkitehtuurista ja tulevista sovellustarpeista. Näiden visioiden perusteella ohjataan hankintaprosessia. Tällöin tehdään esimerkiksi valintoja kilpailevien komponenttiteknologioiden välillä ja määritellään vaikkapa mahdollinen toiminnanohjausjärjestelmän tarve. Hankinta- ja hallintaprosessit ovat vastuussa sellaisista teknologiakartoituksista, jotka on perinteisesti tehty sovellusprojektien yhteydessä. Keskeinen tae hankinta- ja hallintaprosessin onnistumiselle on **omistajuus**: kullekin



Kuva 1 . Komponentteihin perustuvan ohjelmistotuotannon keskeiset prosessit ja niiden välisiä suhteita.

komponentille tarvitaan omistaja, joka maksaa komponenttihankinnan ja varmistaa investoinnin hyödyllisyyden. Paras omistajaehdokas on useassa tapauksessa liiketoimintaprosessin haltija.

Uudelleenkäytön keskeiseksi ongelmaksi on monesti noussut NIH-ilmiö ("not invented here") eli kehittäjien haluttomuus käyttää muiden keksimiä ratkaisuja. Tämän välttämiseksi tarvitaan hankittujen tai itse toteutettujen komponenttien **myyntityötä** organisaation sisällä. Komponenttivarasto antaa tähän tekniset mahdollisuudet, mutta varsinainen työ perustuu henkilökohtaiseen vaikuttamiseen ja eriasteisiin "keppi ja porkkana" –menetelmiin. Myyntityö on syytä ottaa alusta pitäen osaksi hallintaprosessia.

Kuvassa 1 on esitelty uudistetun ohjelmistotuotannon keskeiset prosessit. Siihen on myös merkitty joitakin edellä kuvattuja yhteyksiä näiden prosessien välillä.

Uudistetun ohjelmistotuotantoprosessin arviointia

Uudenlainen ohjelmistotuotantoprosessi tähtää suoraan komponenttiarkkitehtuurien lupausten lunastamiseen. Uudelleenkäytettävyys paranee merkittävästi komponenttien hankinta- ja hallintaprosessien ansiosta: komponentit eivät ole enää yhden projektin välineitä vaan koko organisaation omaisuutta. Lisäksi uudelleenkäyttöä voidaan edistää hallintaprosessiin liittyvän myyntityön avulla.

Heterogeenisuuden tuki ei suoranaisesti parane, mutta eriytetyn hankintaprosessin ansiosta voidaan varmistua siitä, että yhteiskäyttöön liittyvät kysymykset on otettu huomioon ennen komponentin hankintaa. Tämän ansiosta heterogeenisuuden aiheuttamat ongelmat voidaan pitää aisoissa.

Myös laajojen järjestelmien hallittavuus paranee sen myötä, että komponenttien hankintaprosessi tapahtuu erillään projektityöstä. Komponentin hankinnan yhteydessä pyritään aina selvittämään sen sopivuus valittuihin hallintaratkaisuihin.

Uudistettu ohjelmistotuotantoprosesi ei luonnollisesti pysty luomaan ulkopuolisia komponenttimarkkinoita, mutta organisaation sisällä tällainen voi tapahtuakin. Jos komponentin maksajana ja omistajana on kulloisenkin valinnan käynnistänyt organisaatioyksikkö, hallintaprosessiin liittyvä markkinointi voi jälkikäteen pienentää projektin kustannuksia kun muut projektit ostavat käyttöoikeuden komponenttiin. Tällainen menettely ohjaa komponenttivalintoja sellaiseen suuntaan, että uudelleenkäyttö on myöhemmissä projekteissa mahdollista.

Keskeisin etu uudessa ohjelmistotuotantoprosessissa on kuitenkin tietotekniikan parantunut kyky sopeutua nopeisiin muutoksiin. Liiketoimintaprosessien määräämät ohjelmistoprojektit harhautuvat tyypillisesti paljon harvemmin teknologiaoravanpyörään kuin vastaavat perinteiset projektit. Liiketoimintalähtöinen projektityö myös ohjaa oikeiden komponenttirajojen löytymiseen, jolloin vältytään komponenttien tai niiden osien uudelleenkirjoittamiselta. Tekniset selvitystyöt tapahtuvat pääosin hankinta- ja hallintaprosesseissa, jolloin ne eivät vie resursseja aikakriittisiltä sovellushankkeilta. Sovellusprojektit tehdään toimivilla ja ennakkoon evaluoiduilla välineillä ja komponenteilla.

Komponenttihankinnan eriytyksestä voi tietysti seurata se, etteivät sovellukset valmistuessaan käytä viimeisintä mallia olevia teknologiaratkaisuja. Vastapainoksi saadaan kuitenkin nopeampi tuki liiketoiminnan vaatimuksille ja parempi käyttö teknologiainvestoinneille. Lisäksi teknologiakehitys näyttää tällä hetkellä siltä, ettei juuri mikään järjestelmä ole valmistuessaan uusimman teknologiamuodin mukainen.

Uudella ohjelmistotuotantoprosessilla on myös selkeitä haittapuolia. Keskeisenä ongelmana on hankinta- ja hallintaprosessien vaatima lisätyö. Tämän työn hyödyt eivät näy ensimmäisten hankintojen yhteydessä vaan vasta silloin, kun organisaation komponenttiomaisuus on karttunut sellaiseksi, että se johtaa uudelleenkäyttöön lähes joka projektissa.

Toisen ongelman muodostavat olemassaolevat järjestelmät, joita ei ole rakennettu komponenttipohjaisesti. Tyypillisesti nämä ovat merkittävässä asemassa, kun tarkastellaan laajemmin organisaation sovellusomaisuutta, joten alkuvaiheessa on varauduttava komponenttiajattelulle vieraisiin ratkaisuihin. Esimerkiksi redundanttia tietoa joudutaan kuskaamaan komponenttien ja legacy-järjestelmien välillä. Samoin palveluarkkitehtuuri on rakennettava askel kerrallaan. Tämä voi tapahtua esimerkiksi siten, että uusien komponenttien tarvitsemat palvelut toteutetaan komponenttikäärön ("wrapper") avulla. Tällöin vanhat sovellukset käyttävät tietoja suoraan ja uudet sovelluksen komponenttirajapinnan kautta.

Merkittäviä vaikeuksia aiheuttaa myös liiketoimintaprosessien valinta sovellustyön lähtökohdaksi. Tämä vaikuttaa merkittävästi ohjelmistotyöhön eikä voi olla aiheuttamatta vastarintaa. Esimerkiksi projektin tuloksia ei enää arvioidakaan tietoteknisillä kriteeriellä annetun vaatimusmäärittelyn pohjalta, vaan tulosten kelvollisuus määritelläänkin tietotekniikan ulkopuolisilla kriteereillä: saako liiketoimintaprosessi tarvitsemansa tuen. Sovellusrakentajan onkin paneuduttava kohteena olevan järjestelmän lisäksi myös taustalla olevaan liiketoimintaan.

Yhteenveto

Uudet komponenttiarkkitehtuurit tarjoavat suuren joukon lupauksia paremmasta tulevaisuudesta. Käytännössä on kuitenkin havaittu, että komponenttipohjaiset hankkeet eivät ole onnistuneet odotetulla tavalla. Vaikka osa epäonnistumisista voidaankin lukea teknologian kypsymättömyyden piikkiin, keskeinen syy vaikeuksiin on kuitenkin löydettävissä toisaalta: ohjelmistotuotantoprosessista. Artikkelissa esitellään neljä keskeistä vaatimusta ohjelmistotuotantoprosessille, jotta komponenttipohjaisten järjestelmien rakentaminen etenee tavoitteiden mukaisesti ja uuden teknologian lupaukset voidaan lunastaa. Vaatimukset pureutuvat projektin tavoiteasetantaan, sovellusarkkitehtuuriin, kehitysvälineisiin, sekä komponenttien hankintaan ja hallintaan. Uudenlainen ohjelmistotuotantoprosessi edellyttää merkittävää harppausta organisaation toimintatavassa, eikä sen käyttöönotto onnistu ilman huolellista suunnittelua.

FL Pekka Kähkipuro toimii teknologiajohtajana SysOpen Oyj:ssä. Hän on viime aikoina työskennellyt erityisesti komponenttiarkkitehtuureihin pohjautuvien hankkeiden parissa.

Komponenttipohjainen suunnittelu

Jari Isokallio & Jaroslaw Skwarek, TietoEnator Oyj

Nykyinen komponenttipohjainen suunnittelu riippuu usein käytössä olevasta komponenttimallista ja/tai toteutustekniikkasta. Yleistä yksikäsitteistä suunnittelutapaa ei valitettavasti ole. Tässä artikkelissa esitetään yksi komponenttimalli- ja toteutusriippumaton lähestymistapa. Lähestymistapa on oliopohjainen ja notaationa käytetään UML:ää.

Komponenttipohjaisessa suunnittelussa huomioitavia seikkoja

Komponenttipohjaisessa suunnittelussa korostuvat mm. seuraavat eri seikat:

- Hienojakoinen vs. karkeajakoinen lähestyminen
- Luokan vs. komponentin ero
- Tietotyyppimuunnokset
- Hajautus
- Tapahtumankäsittely
- Tilallinen vs. tilaton
- Pysyvyys.

Hienojakoinen vs. karkeajakoinen lähestyminen. Hienojakoinen tarkoittaa, että komponenttien välillä ei kopioida luokkia ja karkeajakoinen tarkoittaa, että luokkien kopiointi sallitaan. Hienojakoiset komponentit ovat vähemmän riippuvaisempia muista komponenteista kuin karkeajakoiset. Hajautuksen tai suorituskyvyn takia voidaan joutua käyttämään karkeajakoista lähestymistapaa (luokkien kopiointi).

Luokan vs. komponentin ero. Luokka ei ole komponentti ja komponentti ei ole luokka. Luokan avulla kuvataan käsitteen tietoja ja sen tarjoamia toimintoja. Tietoja nimitetään attribuuteiksi ja toimintoja nimitetään metodeiksi. Komponentilla tarkoitetaan laajempaa kokonaisuutta, jota voidaan käsitellä rajapinnan kautta. Rajapinta on joukko palveluja, joita komponentti tarjoaa. Komponentilla voi olla useita eri rajapintoja ja yhdestä rajapinnasta olla useita eri toteutuksia. Komponenttia voidaan käyttää vain sen rajapinnan kautta. Yksikäsitteistä tapaa luokan ja komponentin suhteiden muodostamiseen ei ole, vaan suhteiden muodostaminen riippuu käyttötarkoituksesta. Esimerkiksi liiketoimintaluokan metodit eivät ole suoraan komponentin palveluja, kun taas ns. kontrolliluokan metodit voivat olla suoraan rajapinnan palveluja.

Tietotyyppimuunnokset. Missä suoritetaan tarvittavat tietotyyppimuunnokset ? Rajapinnan palveluissa vai erillisissä sovitinluokissa?

Hajautus. Jos komponenttia hajautetaan, on hajautus huomioitava rajapintoja suunniteltaessa. Tarvitaanko esimerkiksi eri hajautustarpeita varten useita erilaisia rajapintoja tai niiden toteutuksia?

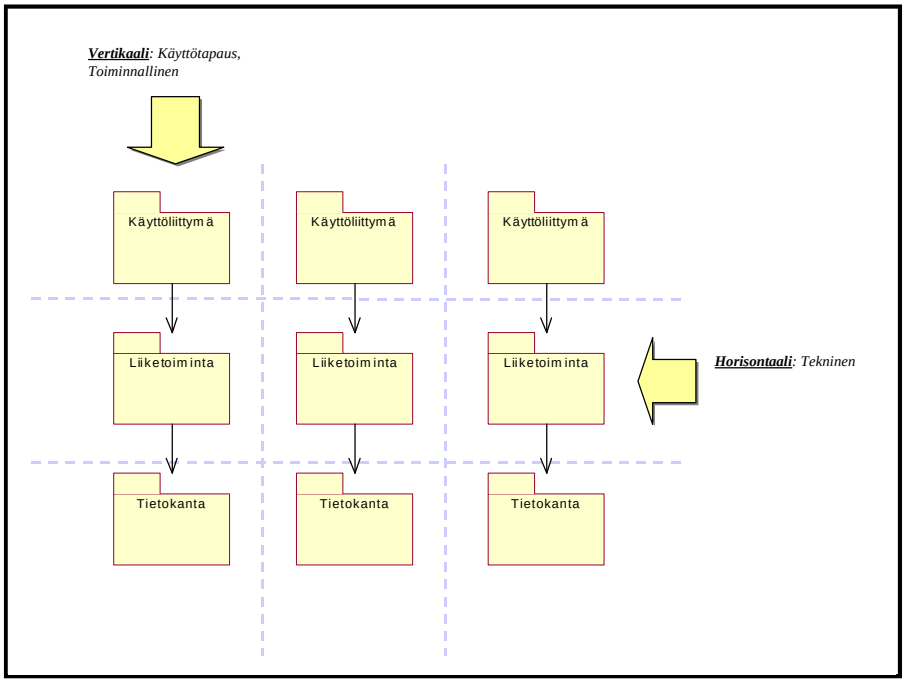
Tapahtumankäsittely. Miten komponentti suorittaa tapahtumankäsittelyä? Luodaanko aina uusi tapahtuma, liitytäänkö olemassa olevaan tapahtumaan vai eikö suoriteta tapahtumankäsittelyä ollenkaan?

Tilallinen vs. tilaton. Tilallinen komponentti säilyttää tietonsa palvelukutsujen välillä, kun taas tilaton komponentti ei. Kun komponentti sisältää tilan, on tila yksilöllinen sovellukselle, joka luo komponentin. Tällaista komponenttia ei voida jakaa muiden sovellusten kesken, sillä ainoastaan komponentin luonut sovellus tietää, mitä se laittoi komponentin tilaksi. Koska tilaton komponentti ei sisällä tilatietoa, voidaan komponentin ilmentymää uudelleenkäyttää ajon aikana eri sovellusten kesken. Tilattomia komponentteja käytettäessä joudutaan tilatieto säilyttämään muualla, esimerkiksi käyttöliittymässä.

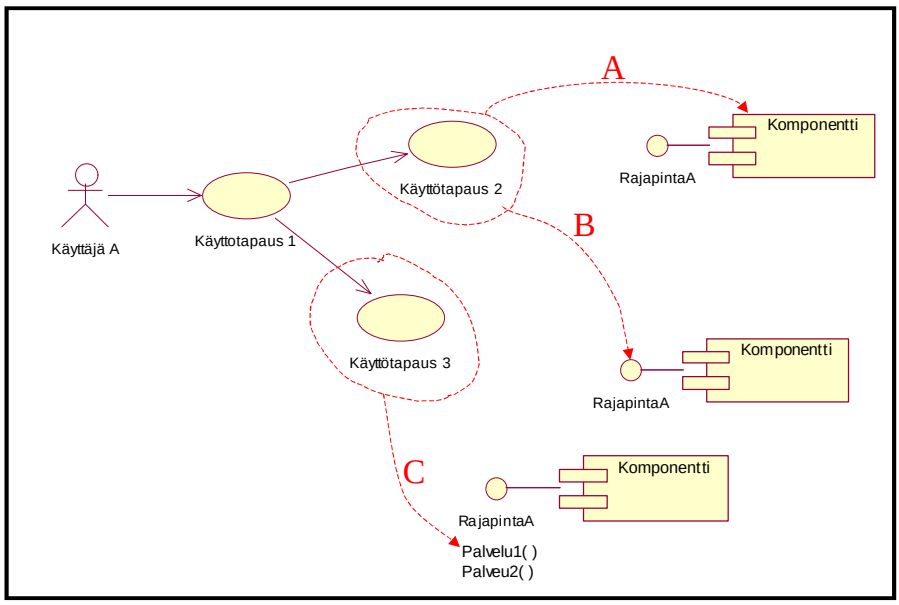
Pysyvyys. Tallentaako komponentti jotain pysyvää tietoa vai onko komponentin käsittelemä tieto ei-pysyvää?

Minkälaisia komponentteja on?

Komponentteja voidaan lähestyä useasta eri näkökulmasta. Esimerkiksi kuvan 1 3-tasoarkkitehtuuria voitaisiin tarkastella toiminnallisesta tai teknisestä näkökulmasta. Toiminnallisesta näkökulmasta katsottuna komponentin toiminnallisuus jakautuu usean eri kerroksen kesken. Tällöin komponentista on osia jokaisessa eri kerroksessa. Teknisestä näkökulmasta katsottuna komponentti voisi palvella vain yhtä kerrosta tai suorittaa jotain tiettyä teknistä toimintaa, esimerkiksi virheinformaation toimittamista järjestelmänhallintaan.



Kuva 1: Toiminnallinen vs. tekninen komponentti.



Kuva 2: Käyttötapauksista komponentit.

Miten löydän komponentit ?

Tässä esitetään kaksi eri lähestymistapaa komponenttien löytämiseksi. Molempien lähtötapojen pohjana on UML:n käyttötapaukset. Ensimmäisessä lähestymistavassa käyttötapauksista muodostetaan suoraan komponentteja. Toisessa lähestymistavassa käyttötapauksista muodostetaan komponentit hyödyntäen paketteja.

Käyttötapauksista komponentteihin. Kuvassa 2 esitetään eri tavat muodostaa käyttötapauksista komponentit.

Kohdassa A yhdestä tai useasta käyttötapauksesta voidaan muodostaa yksi komponentti. Tällöin komponentti toteuttaa ko. käyttötapauksen tai käyttötapauksien toiminnallisuuden.

Kohdassa B yhdestä käyttötapauksesta muodostetaan rajapinta. Esimerkiksi ulkoisista järjestelmistä tietoa hakevat käyttötapaukset voitaisiin sijoittaa yhteen hakukomponenttiin.

Kohdassa C yhdestä käyttötapauksesta muodostetaan yksi tai useampi rajapinnan palvelu.

Käyttötapauksista komponentit paketteja hyödyntäen. Kuvassa 3 esitetään eri vaiheet muodostaa käyttötapauksista komponentit pakettien avulla.

Aluksi käyttötapaukset ryhmitellään loogisiin kokonaisuuksiin joidenkin toiminnallisten ja/tai teknisten ryhmittelytekijöiden mukaisesti. Toiminnallinen ryhmittely voisi olla esimerkiksi asiakastiedot, tuotetiedot, tilaustiedot jne. Tekninen ryhmittely esimerkiksi hakupalvelut, integrointipalvelut muihin sovelluksiin, raportointi jne. Seuraavaksi loogisista kokonaisuuksista muodostetaan paketit siten, että yhdestä loogisesta kokonaisuudesta tulee yksi paketti.

Seuraavaksi piirretään pakettien välille riippuvuudet. Paketti on riippuvainen toisesta paketista, jos paketti tarvitsee jotain toimintoa toisesta paketista. Riippuvuus ilmaistaan yksisuuntaisella nuolella. Esimerkiksi kuvassa 3 paketti 1 on riippuvainen paketista 2. Riippuvuudet saadaan usein suoraviivaisesti käyttötapauksista.

Riippuvuuksien piirtämisen jälkeen suoritetaan syklien poisto ja riippuvuuksien minimointi. Kaikki syklit tulisi poistaa ja riippuvuudet minimoida. Syklien poisto ja riippuvuuksien minimointi tehdään siirtelemällä tai kopioimalla käyttötapauksia pakettien välillä. Toinen tapa on määrittää vastuut paketeille ja siirrellä niitä pakettien kesken riippuvuuksien minimoimiseksi. Molempia tapoja voidaan myös käyttää yhdessä.

Lopuksi paketeista muodostetaan komponentit ja määritellään komponenttien rajapinnat ja rajapintojen palvelut. Tässäkin periaatteena on, että yhdestä paketista tulee yksi komponentti. Tästä voidaan kuitenkin joutua poikkeamaan mm. karkeajakoisten komponenttien osalta.

Komponenttien rajapintojen suunnittelu voidaan tehdä joko toiminto- tai tietopohjaisesti tai yhdistelemällä molempia. Molempien tapojen yhdistelmä on suositeltavampi, koska tällöin huomioidaan rajapinnan kautta välitettävien tietojen lisäksi myös toiminnalliset tarpeet.

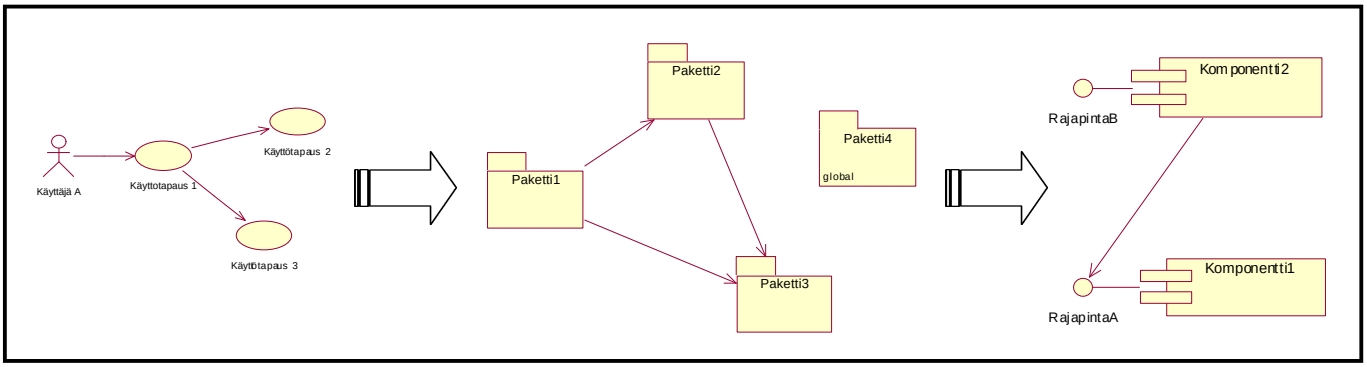
Miten kuvaan komponentit ?

Komponentien suunnittelussa on kuvattava niiden rakennetta ja käyttäytymistä. Luokka-, paketti- ja komponenttikaavioilla kuvataan komponenttien rakennetta. Komponenttien käyttäytymistä voidaan kuvata esimerkiksi vuorovaikutuskaavioilla (sequence diagram).

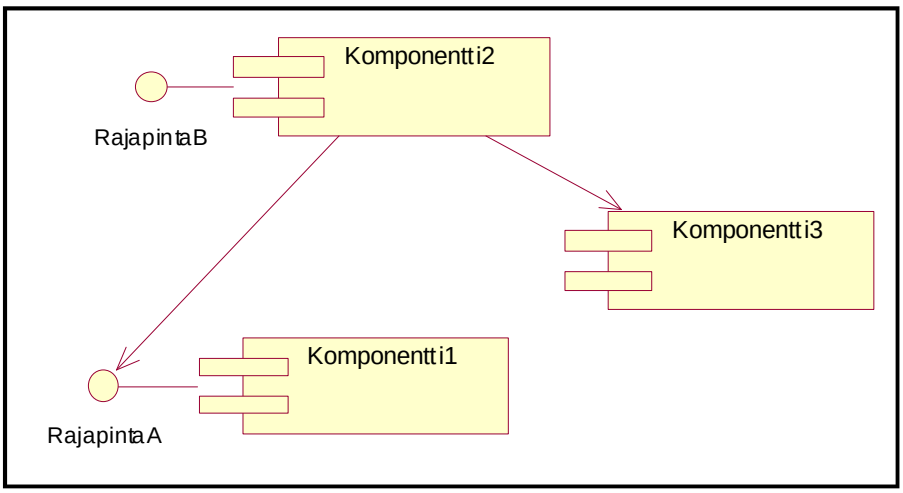
Komponenttien rakennetta ja vuorovaikutusta tarkastellaan eri näkökulmista:

- komponenttien väliset suhteet
- sisäinen rakenne
- suhde liiketoimintaluokkiin.

Komponenttien kuvaustavat riippuvat sovellusarkkitehtuurista ja komponenttimallista.



Kuva 3: Käyttötapauksista komponentit hyödyntäen paketteja.



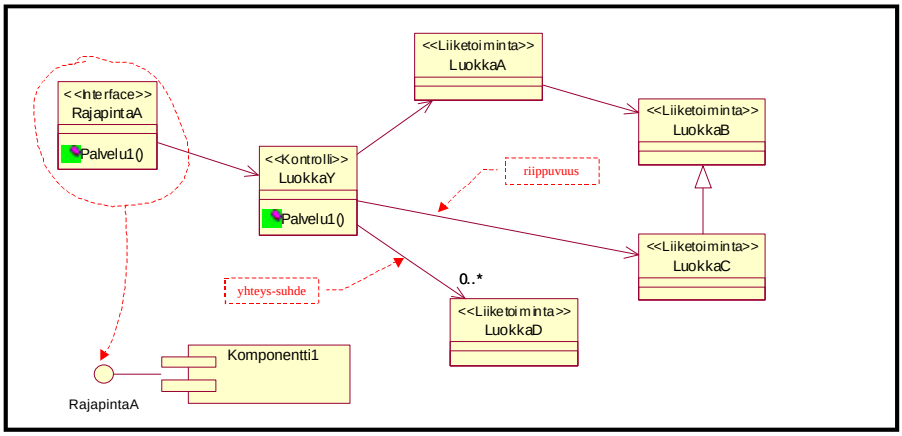
Kuva 4: Komponenttikaavio

Komponenttien välisiä suhteita eli riippuvuksia voidaan kuvata komponenttikaavion avulla (kuva 4).

Riippuvuutta voidaan kuvata karkealla tasolla eli ilman rajapintoja. Kuvausta voidaan tarvittaessa tarkentaa lisäämällä rajapinnat. Niiden mukaan ottaminen voi tehdä kuvauksesta käyttötapauskohtaisen, koska tilanteesta riippuen komponentit saattavat käyttää eri rajapintoja tai rajapintaversioita. Kuten pakettijaossa, myös komponenttien suunnittelussa on pyrittävä yksisuuntaisiin riippuvuuksiin.

Komponenttien välistä keskustelua eli vuorovaikutusta voidaan kuvata vuorovaikutuskaavion avulla. Yksinkertaista vuorovaikutusta ei kannata kuvata, koska se on luettavissa komponentin rajapinnasta. Palvelukomponenttityyppisessä lähestymistavassa komponenttien välinen keskustelu saattaa olla yleisestikin hyvin suoraviivaista jolloin vuorovaikutuskaavio saattaa olla kuvauksena turhan raskas.

Komponenttien sisäisellä rakenteella tarkoitetaan komponenttien rakentamisessa käytettäviä (teknisiä tai teknisluonteisia) luokkia lukuun ottamatta liiketoimintaluokkia. Sellaisia luokkia ovat mm. rajapinta- ja kontrolliluokat (interface ja control) tai muut käytettävästä komponenttimallista johtuvat luokat. Rajapintaluokat ovat useimmiten komponenttimalliriippuvaisia ja käyttävät esim. teknisiä tietotyyppejä, jotka on hyvä eristää liiketoimintaluokista. Kontrolliluokat kuuluvat osaksi liiketoimintalogiikkaa, mutta ovat käyttötapaus- tai rajapintariippuvaisia. Kontrolliluokat paketoivat tiettyyn käyttötapaukseen liittyvät liiketoimintaluokat ja/tai sovelluslogiikan. Kontrolliluokkia ei vältämättä tarvita, jos komponenttimalli ja liiketoiminta käyttävät samoja tietotyyppejä ja tietorakenteita. Komponenttien sisäinen vuorovaikutus on useimmiten samanlaista kaikissa komponenteissa tai noudattaa tiettyjä sääntöjä. Näin ollen vuorovaikutuskaavioita kannattaa tehdä vain kuvaamaan näitä sääntöjä eikä kaikista komponenteista.



Kuva 5: Komponentin suhde liiketoimintaluokkiin

Komponenttien ja liiketoimintaluokkien välisiä riippuvuuksia kuvataan luokkakaaviossa kontrolliluokan kautta (kuva 5).

Tarkoituksena on kuvata mitä olioita/luokkia tarvitaan toteuttamaan kyseisen komponentin toiminta ja mikä on komponentin ja liiketoimintaluokkien välinen suhde. Kontrolliluokan ja liiketoimintaluokkien väliset suhteet (kuva 5) määräytyvät hyvin pitkälle tietotarvepohjaisesti. Jos luokan instanssia tarvitaan vain jossakin palvelussa, niin riippuvuussuhde on riittävä. Yhteys-suhdetta käytetään jos luokan instanssia tarvitaan riippumatta suoritettavasta palvelusta. Yhteys-suhde synnyttää kontrolliluokalle jäsenmuuttujan, joka on mahdollisesti luotava jo kontrolliluokan luomisen yhteydessä. Riippuvuus- tai yhteys-suhteen käytön määrävät monesti suorituskyky, komponenttien tilallisuus/tilattomuus tai komponenttimalli.

Komponentin kontrolli- ja liiketoimintaluokkien sisäistä vuorovaikutusta kuvataan palvelulähtöisesti eli miten palvelun toteuttavat luokat/oliot keskustelevat keskenään ja mitä metodeja ne käyttävät. Tässäkään tapauksessa kaikista palveluista ei kannata tehdä vuorovaikutuskaavioita.

Yhteenveto

Komponenttipohjainen suunnittelu ei välttämättä edellytä oliopohjaista lähestymistapaa, mutta se helpottaa sitä. Tärkeämpää on dokumentoida ne periaatteet ja ratkaisut, joita komponenttipohjaisessa suunnittelussa noudatetaan, oli lähestymistapa sitten oliopohjainen tai ei.

Tässä esitetty suunnittelutapa on toimintolähtöinen. Toimintolähtöisessä tavassa pyritään asiat huomioimaan käyttäjän (esim. loppukäyttäjä tai ulkoinen järjestelmä) näkökulmasta. Lisäksi eri tarpeita varten voidaan asioita lähestyä soveltaen. Lähestymistapa ei pyri olemaan kaikenkattava ja kaikkia tarpeita täyttävä.

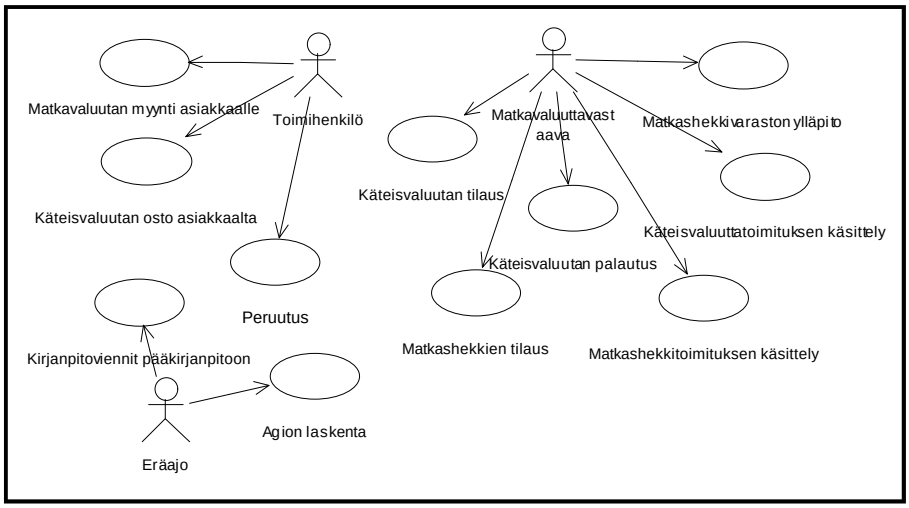
Jari Isokallio, Services
TietoEnator Corporation
jari.isokallio@tietoenator.com

Jaroslaw Skwarek, Public Sector
TietoEnator Corporation
jaroslaw.skwarek@tietoenator.com

Komponenttien suunnittelu case: MAVA

Jarkko Turunen,
Olio- ja internet-ratkaisut
Tieto Entra Oy

MAVA-järjestelmä on matkavaluutan myyntiin ja varastokirjanpitoon tehty sovellus. Sitä käytetään pankkien konttoreissa myytäessä matkavaluuttaa asiakkaille sekä ostettaessa ulkomaan seteleitä asiakkailta. Konttorit voivat järjestelmän avulla myös tilata valuuttaa ja hallita toimituksia. MAVA:sta voidaan lisäksi tulostaa raportteja kirjapidon ja liiketoiminnan tarpeisiin.



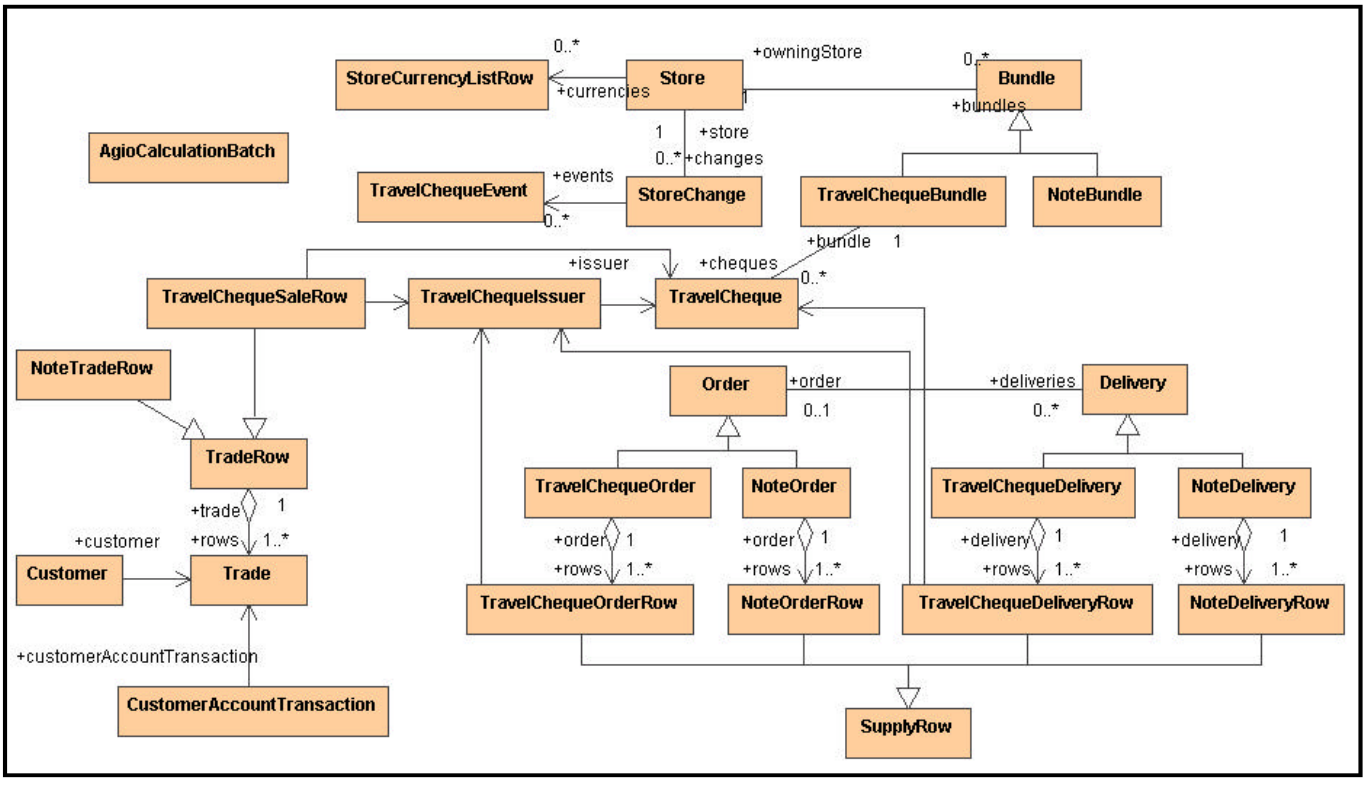
Kuva 1: MAVA:n alustavat käyttötapaukset

Komponenttien suunnittelun lähtökohdat

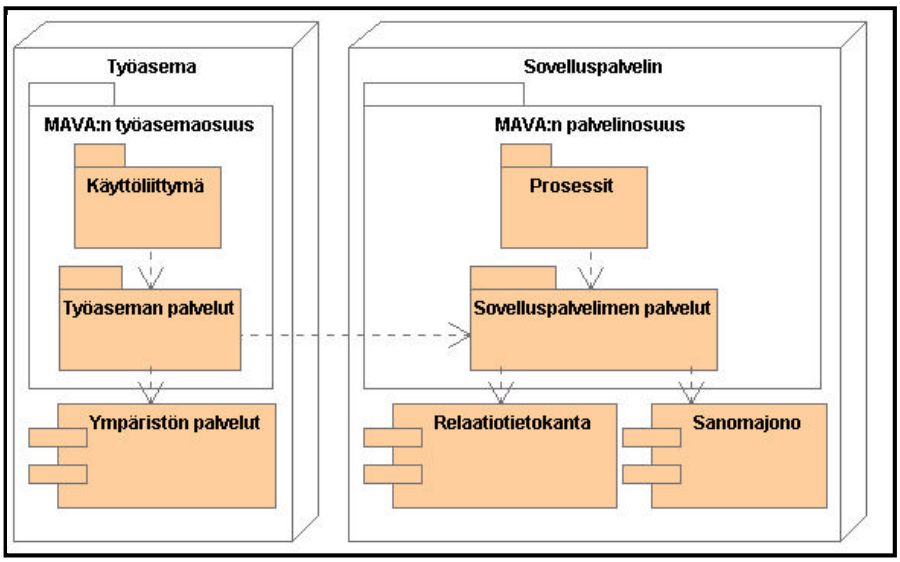
MAVA on komponenttiperustainen sovellus, joka toteutettiin TietoEnatorkonsernin Tieto Object -ohjelmistotuotantoprosessin mukaisesti. Tieto Objectin määrittelyvaiheessa syntyvistä dokumenteissa komponenttien suunnittelussa hyödynnettiin käyttötapauksia, luokkamallia ja käyttöliittymäkuvausta. Prosessin eri vaiheet, kuten luokkamallin ja käyttötapausten teko, tehtiin iteroiden, vaikka ne tässä esitetäänkin erillisinä vaiheina.

Määrittelyvaiheessa laaditut, alustavat käyttötapaukset on esitetty kuvassa 1. Niistä oli kuvan lisäksi olemassa tekstikuvaukset.

Käyttötapausten teon rinnalla liiketoiminnan käsitteitä mallinnettiin UML:n mukaiseen luokkakaavioon ja suunniteltiin sovelluksen käyttöliittymää. Luokkien mallinnuksessa päätet-



Kuva 2: MAVA:n luokkamalli, attribuutit ja operaatiot on jätetty pois



Kuva 3: MAVA:n alustava komponenttijako

tiin, että kauppa on järjestelmän keskeisin luokka ja lähdettiin mallintamaan muita käsitteitä kauppaan liittyen. Näistä käsitteistä luotu alustava luokkamalli on esitetty kuvassa 2. Luokkamalliin ei vielä tässä vaiheessa lisätty operaatioita, vaan siinä esiintyvät ainoastaan pysyvät attribuutit.

Käyttöliittymästä oli luokkamallin teon aikana tehty prototyyppi. Prototyyppi perustui käyttötapauksiin, erään aikaisemman järjestelmän näytöistä tehtyihin havaintoihin ja suunnittelijoiden kokemuksiin käyttöliittymistä.

Arkkitehtuurin määrittely

MAVA:n arkkitehtuuria ruvettiin tarkentamaan käyttötapausten ja luokkamallin perusteella. Aiemmin oli päätetty, että järjestelmä noudattaa kolmitasomallia, ja että teknisessä toteutuksessa käytetään Microsoftin Component Object Model (COM) -teknologiaan perustuvaa Microsoft Transaction Server (MTS) -sovelluspalvelinta. MTS:n käytöstä oli saatu hyviä kokemuksia aikaisemmin toteutetuista järjestelmistä. Tietovarastoksi oli valittu Microsoft SQL Server ja käyttöliittymän ohjelmointikieleksi Microsoft Visual Basic. Muut komponentit on pääosin toteutettu Microsoft Visual C++-kielellä.

Liiketoimintaprosessien vaatimuksesta sovelluksen oli käytettävä jatkuvasti ajantasaisia valuuttakursseja, jotka saataisiin keskuslaitesovellukselta. Samoin järjestelmän piti pystyä välittämään tietoja olemassa oleviin järjestelmiin, muun muassa kirjanpitosovelluksiin. Näiden ulkopuolisten järjestelmien kanssa oli päätetty kommunikoida sovelluspalvelimella sijaitsevan sanomajonon välityksellä. Sanomajono-ohjelmistoksi oli valittu IBM MQSeries.

Arkkitehtuurissa oli lisäksi varauduttu siihen, että sovelluspalvelin sijoitetaan pankin tietotekniikkakeskukseen ja työasemia on ympäri maata. Myös verkkoyhteyksien rajattuun kapasiteettiin piti varautua, heikoimmillaan työasemat saattaisivat olla yhteydessä palvelimeen 64 kb/s siirtävän yhteyden lävitse.

Arkkitehtuurin määrittelyn eräs tulos on alustava komponenttijako. Se kuvaa sovelluksen tärkeimmät komponentit ja niiden sijainnin. MAVA:n alustava komponenttijako esitetään kuvassa 3. Työasemassa sijaitsee käyttöliittymän lisäksi edustajakomponentti (smart proxy), joka tarjoaa käyttöliittymälle kaikki sen tarvitsemat palvelut. Se huolehtii myös käyttöliittymän tilan hallinnasta: käyttöliittymä ohjaa kaikki saamansa syötteet edustajakomponentille, joka käsittelee ne, suorittaa tarvittavat operaatiot ja kutsuu käyttöliittymän toteuttamia tapahtumankäsittelijöitä. Nämä suorittavat varsinaisen näytön päivityksen. Esimerkiksi käyttäjän lisätessä setelien myyntiin uutta valuuttaa, ohjaa käyttöliittymä syötetyn valuuttakoodin ja valuuttamäärän edustajakomponentille, joka laskee kaupan summan. Lopuksi edustajakomponentti kutsuu käyttöliittymän toteuttamaa tapahtumankäsittelijää, joka päivittää näytöllä sijaitsevan summakentän. Edustajakomponentti käyttää palveluja tarjotessaan apunaan sekä työasemassa että sovelluspalvelimella sijaitsevia komponentteja. Tällä ratkaisulla käyttöliittymä saadaan pidettyä kevyenä ja se on mahdollista korvata toisenlaisella ratkaisulla.

Arkkitehtuurissa varauduttiin siihen, että joitakin ympäristöltä tarvittavia palveluja ei ole mahdollista käyttää sovelluspalvelimelta. Näitä palveluja suunniteltiin käytettäväksi työasemalta edustajakomponentin avulla.

Edustajakomponentin käyttämät, sovelluksen varsinaisen liiketoiminnan muodostavat komponentit sijoitettiin sovelluspalvelimelle MTS:n alaisuuteen. Nämä käyttävät tietokantaa ODBC-rajapinnan avulla. Liiketoimintakomponenttien lisäksi palvelimelle tarvittiin prosessi, joka purkaa valuuttakurssit sanomajonolta.

Komponenttijaon tarkentaminen

MAVA-projektin siirryttyä Tieto Objectin mukaiseen suunnitteluvaiheeseen komponenttijakoa alettiin tarkentaa. Tarkennuksen apuna käytettiin luokkamallia, käyttötapauksia ja käyttöliittymästä tehtyä prototyyppiä. Tarkennettu komponenttijako esittää lopulliset, järjestelmään toteutettavat komponentit ja niiden vastuut. Syntynyttä komponenttijakoa tarkasteltaessa voidaan tutkia erikseen työasemaosuuden ja palvelinosuuden komponenttijakoja. Yhteisenä kummassakin oli liittymäpalvelujen erottaminen omiksi komponenteikseen. Kutakin liittymäpalvelua varten määritettiin helppokäyttöinen rajapinta, jonka läpi kyseistä palvelua käytetään. Rajapinta toteutettiin komponentissa, joka varsinaisesti kutsuu ympäristön tarjoamaa palvelua. Näiden rajapintojen ansiosta MAVA:lla on yhdenmukaiset ja helppokäyttöiset rajapinnat kaikkiin tarvitsemiinsa palveluihin. Samalla ympäristön palvelussa tapahtuva muutos eristettiin MAVA:n ydintoiminnallisuudesta.

Ydintoiminnallisuuden jakaminen komponenteiksi tapahtui työasemassa eri lähtökohdista kuin palvelimella. Työasemassa komponenttijakoa ei yleisellä tasolla tarvinnut tarkentaa lukuun ottamatta mainittua liittymäkomponenttien lisäystä. Kaikki työaseman palvelut toteutettiin edustajakomponenttiin.

Sovelluspalvelimen tarkennettu komponenttijako syntyi luokkamallin (kuva 2) perusteella. Luokkamalliin oli mallinnettu järjestelmän pysyvät luokat. Tästä luokkamallista valittiin aluksi ydinluokat, jotka mallinsivat määrittelyvaiheessa löydettyjä keskeisiä käsitteitä. Luokkamalli pilkottiin osiin etsien ydinluokat, kuten Trade, Store ja Supply ja liittämällä kyseisiin luokkiin niihin kaikkein lähimmin liittyvät luokat. Esimerkiksi Trade:n liittyvät lähimmin Customer, CustomerAccountTransaction ja TradeRow-luokkahierarkia. Ydinluokka ja siihen liittyvät luokat muodostivat komponentit. Esimerkiksi MAVATrade-komponentti muodostuu Trade-luokasta ja siihen liittyvistä luokista.

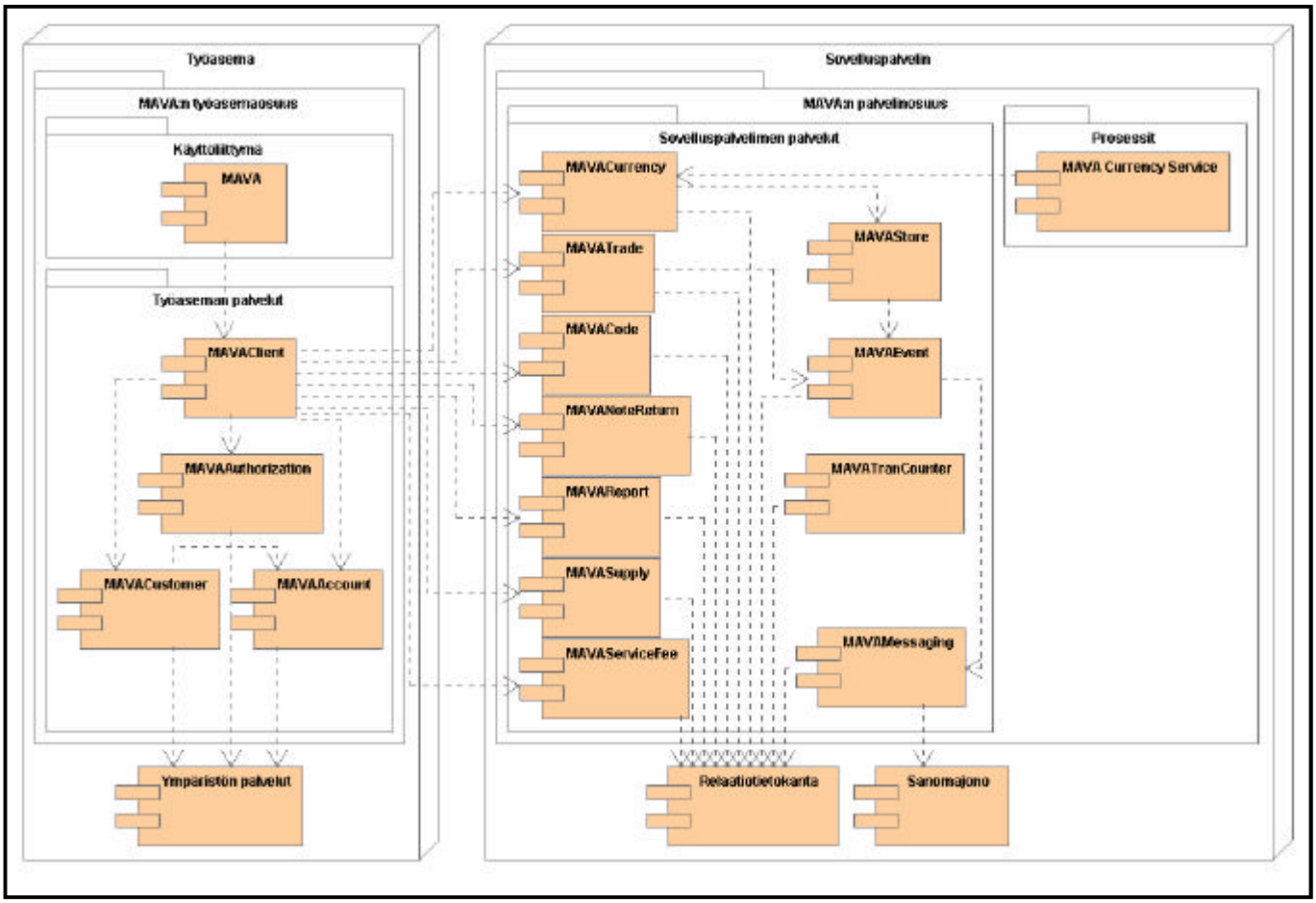
Luokkien välisiä suhteita seuraamalla edettiin luokkiin, jotka liittyivät kahteen tai jopa useampaan tärkeään luokkaan. Tällaiset luokat liitettiin etupäässä intuition perusteella siihen komponenttiin, johon ne olivat kiinteämmässä suhteessa. Toiseen komponenttiin jääneeseen luokkaan lisättiin tietokannan vierasavainta muistuttava attribuutti, jonka avulla tarvittavat tiedot voitaisiin pyytää komponentilta, jonne luokka asetettiin. Joissain tapauksissa luokan tiedot kuitenkin monistettiin useaan komponenttiin. Näin tehtiin esimerkiksi TravelCheque-luokalle, joka sijaitsee MAVAStore-komponentissa. Sen tiedot monistettiin myös MAVATrade-komponentin TravelChequeTradeRow-luokkaan.

Edellä kuvatulla menettelyllä luokkamalli saatiin jaettua komponenteiksi, joiden toteutus voitiin jakaa eri henkilöille. Valmis komponenttijako on esitetty kuvassa 4.

Yksittäisten komponenttien suunnittelu

Yksittäisten komponenttien suunnittelussa komponentit oli jaettu eri suunnittelijoiden vastuille. Yksi suunnittelija vastasi useammasta komponentista, suunnitellen joko palvelinkomponentteja tai työaseman komponentteja. Pääosin samat henkilöt myös lopulta toteuttivat komponentit. Kukin suunnittelija sai kuitenkin varsin vapaat kädet komponenttiensa suunnitteluun: rajapinnat sovittiin palvelun tarjoajan suunnittelijan ja palvelun käyttäjän suunnittelijan kesken. Sisäinen toteutus sen sijaan jätettiin kunkin komponentin ohjelmoijan vastuulle. Joitakin yleisiä periaatteita rajapintojen suunnitteluun ja komponenttien sisäiseen toteutukseen oli kuitenkin sovittu. Kuten komponenttijaon teko, myös yksittäisten komponenttien suunnittelu oli erilaista työasemassa ja sovelluspalvelimessa sijaitsevilla komponenteilla.

Työasemassa sijaitsevilla



Kuva 4: MAVA:n tarkennettu komponenttijako

komponenteilla rajapinnat pyrittiin tekemään Visual Basic -ohjelmoinnin kannalta mahdollisimman yksinkertaisiksi ja oliomaisiksi. Niinpä edustajakomponenttiin toteutettiin lähes sellaisenaan luokkamallia vastaavat rajapinnat, joiden avulla käyttöliittymä saattoi navigoida tarvitsemiensa tietojen luo. Edustajakomponenttiin toteutettiin myös "neljän jengin" observer-suunnittelumallin mukaisia tapahtumapalveluja, joihin käyttöliittymä rekisteröi tapahtumankäsittelijänsä.

Sovelluspalvelimella sijaitsevien komponenttien toteutus oli hieman rajoitetumpaa kuin työaseman komponenttien. Hajautuksen vuoksi komponenttien rajapinnoista oli tehtävä sellaisia, että tiettyyn operaatioon tarvittavat parametrit välitettiin kerralla työasemasta palvelimeen. Teknisesti tämä toteutettiin RPC-tietueiden avulla, jotka olivat testeissämme osoittautuneet nopeimmaksi tavaksi siirtää tietoa C++-kielellä ohjelmoitujen komponenttien välillä. Periaate operaatioiden tarvitsemien tietojen välityksessä oli, että kaikki pakattiin yhteen tietueeseen, joka välitettiin operaation parametrina. Monessa operaatiossa tietue myös palautetaan takaisin työasemaan palvelimen täydennettyä siihen puuttuvia tietoja. Tietueiden tiedot löydettiin tutkimalla palvelimelle toteutettua luokkamallia.

MTS-arkkitehtuurista johtuen MTS:ssä suoritettavista komponenteista tehtiin tilattomia. Lisäksi tietokannan käsittelyssä päätettiin käyttää sisäisessä projektissa kehitettyä OBN-kehystä, johon kuuluu tietokantaa käsitteleviä luokkia ja tietokantamäärityksiä luova generaattori.

Ratkaisujen evaluointi

MAVA-projektin komponenttien suunnittelussa tehdyt ratkaisut ovat osoittautuneet hyviksi. Järjestelmä on jopa ylittänyt odotetun suorituskyvyn mitattuna transaktioiden määrässä sekunnissa. Samoin komponentointi on onnistunut siinä, että tehdyt muutokset ovat vaikuttaneet pääosin yhteen komponenttiin, joka on voitu vaihtaa. Prosessin kehittämistä varten projektin eri vaiheista kuitenkin tehtiin joitain havaintoja, jotka tässä esitän.

Arkkitehtuuripäätöksiä tehtäessä sovelluspalvelimelle harkittiin vaihtoehtona roolia, jossa se olisi palvellut suoraan käyttöliittymän tarpeita. Tällöin työasemaan ei olisi tehty edustajakomponenttia, vaan edustajakomponentin tehtävät olisi jaettu sovelluspalvelimen ja Visual Basic -käyttöliittymän kesken. Tästä vaihtoehdosta luovuttiin useasta syystä: ensiksi, käyttöliittymän olisi pitänyt kommunikoida suoraan sovelluspalvelimen kanssa, jolloin ei oltaisi voitu käyttää optimaalisimmaksi todettua tiedonvälitysmenetelmää. Toiseksi, Visual Basic -käyttöliittymä olisi joutunut käyttämään monimutkaisempaa, hajautuksen huomioivaa rajapintaa sovelluspalvelimeen sen sijasta, että se käyttäisi oliomaista rajapintaa edustajakomponenttiin. Edelleen, luomalla edustajakomponentti, saatiin vielä hyödynnettyä työaseman laskentakapasiteettia, jolloin palvelimen kuorma jäi vähemmälle. Lopuksi, työasemalla sijaitsi muutamia sovelluksen tarvitsemia palveluja, joita ei voitu siirtää palvelimeen.

Suunnitteluvaiheessa tehdyt käyttötapaukset kasvoivat varsin suuriksi ja monimutkaisiksi. Samalla niiden käyttö kommunikointivälineenä vaikeutui. Ongelma on käyttötapausten kohdalla tyypillinen. Joihinkin MAVA:n käyttötapauksiin tuli hyvinkin tarkkoja käsittelysääntöjä, joiden ilmaiseminen tekstinä johti epätäsmällisyyksiin. Näissä tapauksissa olisi voitu hyödyntää UML:n mukaisia aktiviteettikaavioita, joiden avulla käsittelysäännöt voidaan kuvata täsmällisesti ja havainnollisesti.

Komponenttijaon teossa pohdittiin käsitemallin luokkien roolia kolmitasoarkkitehtuurissa: ovatko luokat työasemassa, palvelimessa vai molemmissa? Tuleeko luokista komponentteja vai ovatko luokat piilossa komponenttien sisällä, joille suunnitellaan rajapinnat erikseen? Siirretäänkö luokat palvelimesta asiakkaaseen? Näihin kysymyksiin vastasi osin tekniikka. Palvelimeen luokkamalli tarvittiin, koska siellä sijaitsi varsinainen liiketoimintalogiikka. Toisaalta käyttöliittymälle oli hyvä tarjota selkeä rajapinta. COM-teknologiassa luokkien siirto linjan yli vaatii erityistoimenpiteitä, palvelimen luokkia ei siis sellaisenaan voitu siirtää palvelimesta työasemaan. Työasemaan toteutettiin siksi oma näkemys luokkamallista, joka näkyi edustajakomponentin rajapinnoissa. Palvelimella rajapinnat sen sijaan olivat proseduraalisia ja luokkamalli eli puhtaasti komponenttien sisällä, vieläpä ainoastaan transaktion keston ajan.

Yksittäisten komponenttien suunnittelun yhteydessä oli päätetty yleisistä ohjelmointiohjeista ja -periaatteista, kuten OBN-kehyksen käyttö tietojen tallentamisessa ja haussa sekä fasadiluokan käyttö komponentin rajapinnan toteutuksessa. Nämä periaatteet olisi ollut hyvä päättää osana prosessia ja kirjata omaksi dokumentikseen, jolloin niitä voisi helpommin hyödyntää myös tulevissa projekteissa.

Kokonaisuudessaan projekti onnistui hyvin. Suunnitteluprosessia voidaan kuitenkin vielä kehittää ottamaan huomioon komponenttiperustaisten sovellusten erityispiirteitä, kuten täsmällinen komponenttijako ja rajapintasuunnittelu. Myös käyttötapausten hyödyntäminen komponenttien suunnittelussa voisi olla tietoisempaa. Projekti kuitenkin tuotti lopputuloksena syntyneen järjestelmän lisäksi paljon tärkeää tietoa komponenttien suunnittelusta.

Jarkko Turunen
Järjestelmäasiantuntija
Olio- ja internet-ratkaisut
Tieto Entra Oy
Kutojantie 10, PL 33, 02631 ESPOO
puh. (09) 8626 2650
jarkko.turunen@tietoenator.com

Ohjelmistokomponentit ja Java

Simo Vuorinen,
TietoEnator Oyj

Ohjelmistoyrityksen jokapäiväisenä haasteena on tuottaa sovelluksia asiakkailleen laadukkaasti, edullisesti ja nopeasti. Perinteisessä sovelluskehityksessä on huomattu, että ohjelmistojen rakentaminen on kallista, puhumattakaan ylläpidosta, ja välttämättä laatu ei aina vastaa odotuksia. Ohjelmistokomponentteihin pohjautuva sovelluskehitys on jo pitkään luvannut poistaa ongelmat monimutkaisuuden, hinnan ja laadun suhteen. Komponenttiteknologiaa on tuuletettu sisään ongelmanratkaisijana, mutta onko se saavuttanut tavoitteensa, jää vielä nähtäväksi. Tällä hetkellä suurin osa ohjelmistotaloista uskonee, että ohjelmistokomponentit helpottavat ja nopeuttavat työntekoa, sekä parantavat osaltaan ohjelmistojen laatua, mutta tunnettu tosiasia myös on, että uudelleenkäytettävän ohjelmistokomponentin tuottaminen vie myös aikaa ja rahaa. Tässä artikkelissa esitellään lyhyesti Java-ympäristössä käytettävät JavaBeans- ja Enterprise JavaBeans-komponenttimallit.

Mikä on ohjelmistokomponentti?

Jotta voidaan keskustella komponenteista, lienee syytä esittää jonkinlainen määritelmä komponentille. Szyperski määrittelee kirjassaan "Component Software" komponentin seuraavasti: "Ohjelmistokomponentit ovat itsenäisesti tuotettuja tai hankittuja ja jaeltavissa olevia binäärisiä yksiköitä, jotka toimivat ja kommunikoivat keskenään muodostaen toimivan järjestelmän". Ohjelmistokomponentille löytyy joukko määrityksiä, mutta tässä artikkelissa ohjelmistokomponentilla tai komponentilla tarkoitetaan valmiiksirakennettuja, valmiiksitestattuja, riippumattomia ja uudelleenkäytettäviä ohjelmamoduleita, jotka suorittavat rajattuja toimintoja.

Komponentille tyypillistä on, että se ei useimmiten ole itsenäinen sovellus, ja se voidaan erottaa suoritusympäristöstään käytettäväksi useissa sovelluksissa. Komponentti kehitetään tiettyyn tarpeeseen, ei tiettyyn sovellukseen. Sitä voidaan käyttää ennaltamäärittelemättömissä olosuhteissa plug-and-play-tyyppisesti muiden komponenttien joukossa. Komponentilla on selkeä liittymärajapinta, joka voidaan toteuttaa olioilla, proseduraalisella koodilla tai kapseloimalla olemassaolevaa ohjelmakoodia.

On lisäksi "superkomponentteja", joissa peruspiirteisiin voidaan lisätä tietoturva, lisensointi, versiointi, elinkaaren hallinta, visuaalinen hallinta, tapahtumien (event) hallinta, konfigurointi, itseasennuksen yms.

Java-ohjelmointikieli

Java-ohjelmointikielen alkuperäinen kehittäjä on Sun Microsystemsin James Gosling. Javaa kutsuttiin alunperin nimellä Oak, ja se oli suunniteltu käytettäväksi sulautetuissa (embedded) kuluttajaelektroniikkasovelluksissa (Gosling ym. 1996, s. xxiii). Useiden vuosien käytön jälkeen sitä alettiin muokata Internetiin soveltuvaksi. Viimeisimmän silauksen kielelle antoi pääasiallisesti James Gosling useiden muiden kollegoiden avustuksella. Java-ohjelmointikieli julkistettiin loppuvuodesta 1995.

Java on yleiskäyttöinen, oliosuuntautunut ohjelmointikieli, joka on pyritty suunnittelemaan niin vähän ympäristöriippuvaiseksi kuin mahdollista. Java pyrkii mahdollistamaan sen, että sovellusta voidaan suorittaa mistä tahansa Internetissä.

Javalla voidaan tavanomaisten työasemaohjelmien lisäksi kirjoittaa sovelmia (applet) - ohjelmia, joita voidaan suorittaa WWW:ssä (World Wide Web) selaimen toimiessa suoritusympäristönä. Nämä ohjelmat, sovelmat, tuovat aktiivista, suoritettavaa sisältöä HTML-sivulle. Aktiivinen sisältö on käyttökelpoista vain, jos sitä voidaan suorittaa ympäristöriippumattomasti asentamatta tai kääntämättä sovelluksia uudestaan (McGraw ja Felten 1997, s. 10), ja tämän Java pyrkii tekemään.

Javan kehitysympäristö koostuu kolmesta pääkomponentista, jotka ovat ohjelmointikieli, virtuaalikone, ja suoritusympäristö. Jotta Javaa voitaisiin suorittaa missä tahansa suoritusympäristössä, on ympäristöön asennettava Javan virtuaalikone (Java Virtual Machine, JVM). Virtuaalikone tulkitsee Java-koodin laitekohtaiseksi konekieleksi.

Java-sovellusympäristöt

Javaa löytyy joka käyttötarpeeseen. Java-sovellusympäristöjä on (application environment, AE) neljä, jotka on suunniteltu eri käyttötarkoituksiin. Nämä ovat

- Java AE
- Personal Java AE
- Embedded Java AE
- Java Card

Java AE on yleensä se, minkä me ymmärrämme Java-sovellusympäristönä; perusympäristö, jonka suoritusympäristönä on työasemat, palvelinlaitteet ja muut vastaavat. Personal Java AE on optimoitu kuluttajaelektroniikan tarpeisiin. Sen käyttötarkoitus on suunnattu esim. kännyköihin, kämmentietokoneisiin ja tosiaikakäyttöjärjestelmiin (RTOS). Embedded Java AE on suunniteltu käytettäväksi esim. kirjoittimissa ym. vastaavissa laitteissa, ja Java Card on nimensä mukaan tarkoitettu älykortti- ja vaikkapa Java Ring-sovelluksiin. Java-ohjelmistokomponentteja – JavaBeans- tai EJB-komponentteja on mahdollista rakentaa näillä näkymin Java AE:iin (sekä JavaBeans- että EJB-komponentit) ja Personal Java AE:iin (JavaBeans-komponentit).

Javan JavaBeans- ja Enterprise JavaBeans-komponenttiarkkitehtuurit

Java-kielessä käytetään tai tunnetaan kaksi komponenttimallia, toinen pääasiallisesti GUI-sovelluksissa (graphical user interface) käytettävä JavaBeans-komponenttimalli, ja toinen taas palvelinsovelluksissa käytettävä Enterprise JavaBeans (EJB)-komponenttimalli.

JavaBeans-komponenttiarkkitehtuuri

JavaBeans-komponenttiarkkitehtuuri on Java-sovellusympäristöön tarkoitettu alustariippumaton käyttöliittymäkomponenttimalli. Tällä tarkoitetaan tässä lähinnä GUI-ympäristöön tuotettuja komponentteja, jotka yleensä ovat visuaalisia käyttöliittymäkomponentteja, kuten painikkeita, valintalistoja, valikoita, ym. Komponentti voi tosin olla näkymätönkin, jolloin sillä ei ole sovelluksen suoritusaikana näkyvää esitysmuotoa.

JavaBeans-mallin etuina voidaan pitää lähinnä siirrettävyyttä ympäristöstä toiseen, ja mallin yksinkertaisuutta. Lisänä ovat Javan mukanaan tuomat muut edut, kuten turvallisuus, verkkosuuntautuneisuus ja vikasietoisuus.

Beanin osat

JavaBeans-komponentin kolme tärkeintä elementtiä ovat

- Ominaisuudet (properties)
- Metodit (methods)
- Viestit l. tapahtumat (events)

Ominaisuudet yleensä määritellään komponentissa yksityisiksi (private),ja niitä voidaan muuttaa ja kysellä ko. attribuuttia varten kirjoitetulla saanti- ja asetusmetodeilla.

Metodit edustavat Bean-komponentin julkistamaa rajapintaa, jonka palveluja komponentin käyttäjä voi kutsua.

Viestit ovat komponentin mekanismi välittää ilmoituksia (notification) ympäristölleen tai muille komponenteille ja olioille. Toiset komponentit voivat rekisteröidä mielenkiintonsa kyseisen komponentin viesteihin, ja kun viestin lähetys tapahtuu, kuuntelevaa komponenttia informoidaan tapahtumasta kutsumalla kuuntelevan komponentin tiettyä metodia. Alla kuva

Kuva 1. JavaBean-komponentti.

perus-Beanista.

JavaBeans-komponentit voidaan rinnastaa toimintansa ja käyttötapansa perusteella ehkä Microsoftin Visual Basicista tuttuihin “control”-käyttöliittymäkomponentteihin, alunperin VBX:iin, sittemmin OCX:iin, joiden kattona tällä hetkellä on ActiveX-komponenttimalli.

Tyypillisellä JavaBeans-komponenttimallin mukaisella komponentilla on JavaBeans Specification-dokumentin mukaan yleensä seuraavat 5 peruspiirrettä:

- Ohjelmointirajapinnan julkistaminen (Introspection l. Interface publishing and discovery)
- Visuaalisen ohjelmoinnin tuki (customization)
- Viestien välitys (event handling)
- Ominaisuudet (properties)
- Pysyvyys (persistance)

Ohjelmointirajapinnan julkistaminen (Interface publishing and discovery)

Kun komponentti asetetaan säiliöönsä (container), sen täytyy ilmaista olemassaolonsa ja julkistaa ympäristölleen rajapintansa, jotta sitä voidaan käyttää. Säiliö tarjoaa suoritusympäristön komponentille. Komponentilla voi olla yksi tai useampia rajapintoja, jonka muotoisena se näkyy käyttäjälleen. Käyttäjä päättää, minkä tyyppisenä haluaa komponenttia käyttää.

Javassa tämä tapahtuu siten, että ympäristö, joka voi olla esim. ohjelmointityökalu, tutkii komponentin Javan heijastus- l. reflection-ohjelmointirajapinnan avulla.

Visuaalisen ohjelmoinnin tuki (customization)

JavaBeans-komponenttia on voitava käsitellä visuaalisesti ohjelmointityökalussa. Tämä ei tarkoita kuitenkaan sitä, että ko. komponentin olisi oltava näkyvä suorituksen aikaisessa ympäristössä, vaan se voi myös olla näkymätön komponentti. Jos komponentti on näkyvä, se yleensä periytyy java.awt.Component-luokasta, mutta jos se on näkymätön, komponentti voi edustaa suurinpiirtein mitä tahansa luokkaa.

Viestien välitys (event handling)

Olioparadigman mukaisissa sovelluksissa oliot kommunikoivat toistensa kanssa viestien välityksellä. Olio tai komponentti lähettää viestin, jonka toimittamisesta vastaanottajalle ympäröivä kehys on vastuussa.

Viesti voi olla järjestelmän itsensä lähettämä, tai sen tuottaja voi olla jokin kehyksessä toimiva olio tai komponentti.

Javan sanomankäsittelymalli perustuu viestiolioihin, joita kuuntelevat ko. viestistä kiinnostuneet kuuntelijat. Viesti itsessään on vain sanoma; se ei itse tee mitään, vaan sisältää ainoastaan informaatiota. Oikealla kuva viestin välityksestä.

Ominaisuudet (properties)

Ominaisuudet ovat käytännössä komponentin nimettyjä attribuutteja, joita voidaan käsitellä sekä suunnittelun aikaisessa ympäristössä että ajonaikaisesti. Komponentilla voi olla esim. ominaisuus, jonka nimi on ’status’, jolloin ominaisuutta voidaan käsitellä getStatus- ja setStatus-metodeilla.

Ominaisuus voi myös olla haluttaessa pelkästään read-only- tai write-only-ominaisuus, jolloin sillä on vain vastaavasti joko saanti- (get) tai asetusmetodi (set).

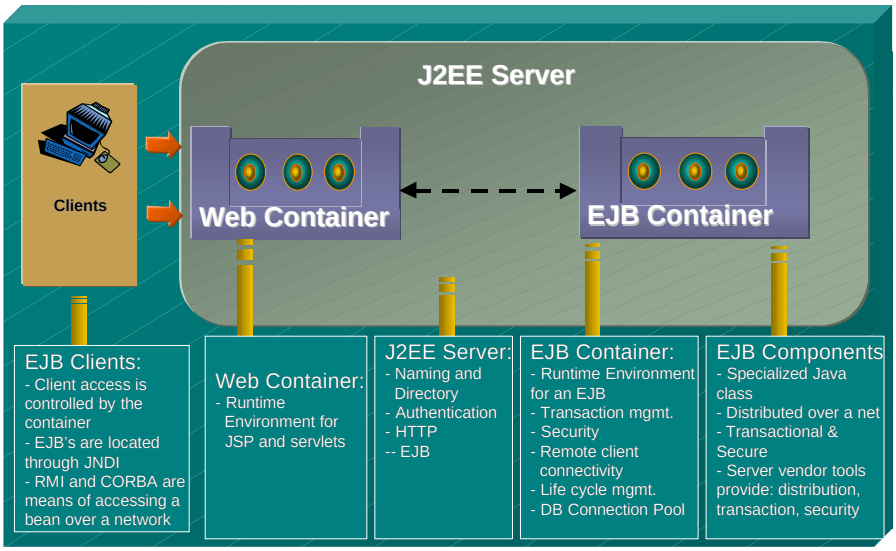
Perusominaisuuksien lisäksi ominaisuus voi olla indeksoitu (indexed property), sidottu (bound property) tai rajoitettu (constrained property). Indeksoitua ominaisuutta voidaan käsitellä indeksin arvolla, sidottu ominaisuus tarkoittaa sitä, että kun ominaisuuden arvo muuttuu, se viestii ympäristölleen arvon muuttuneen. Rajoitetulla ominaisuudella tarkoitetaan sitä, että ominaisuuden arvon muuttuessa jollakin toisella komponentilla voi olla mahdollisuus käyttää Veto-oikeutta ja kieltää muutos.

Pysyvyys (persistance)

Pysyvyydellä tarkoitetaan tässä komponentin kykyä tallentaa tilansa. Jos esim. komponentin käyttäjä on ohjelmointiympäristössään muuttanut painikkeen otsikkoa, on suotavaa, että seuraavan kerran, kun ohjelmoija käynnistää ohjelmointityökalunsa, painikkeen otsikkoa ei tarvitse jälleen muuttaa halutuksi, vaan se on tallennettu edellisen muutoksen yhteydessä. Yksinkertaisin mekanismi on tallentaa komponentin tilainformaatio esim. tiedostoon kovalevylle.

EJB:stä lyhyesti

Enterprise JavaBeans eli EJB määrittää mallin uudelleenkäytettäville Java-palvelinkomponenteille, niiden kehitykseen ja levitykseen. EJB-arkkitehtuuri on

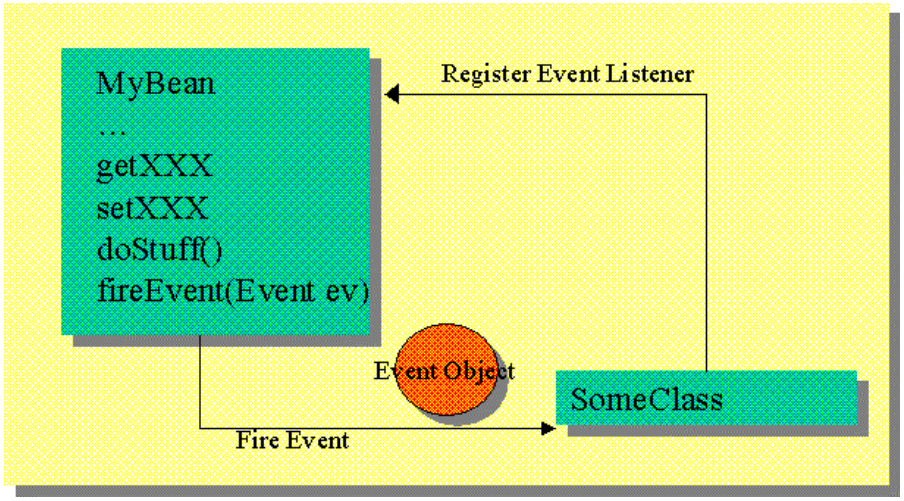


Kuva 3. J2EE-palvelimen rakenne.

hajautettuun tapahtumankäsittelyyn suunniteltu komponenttiarkkitehtuuri. Palvelinkomponentilla tarkoitetaan ohjelmistokomponenttia, jota suoritetaan sovelluspalvelimella (vrt. JavaBeans-malli). EJB-komponentti toimii sovelluspalvelimella, joka tukee EJB-rajapintaa. Tällaisia sovelluspalvelimia ovat esim. BEA:n WebLogic ja IBM:n WebSphere.

Mitä iloa on EJB:stä? Aiemmin jokainen sovelluspalvelin on tukenut omia ohjelmointirajapintojaan, joilla on päästy kiinni palveluihin. Tällaiset palvelinkomponentit ovat huonosti tai ei lainkaan siirrettäviä palvelimelta toiselle. EJB:n tarkoitus on saada aikaan siirrettävä, sovelluspalvelinriippumaton komponenttimalli.

Komponentti on rakennettu yksinkertaisen mallin mukaan, jossa kehittäjä voi keskittyä itse asiaan, sovelluslogiikan ohjelmointiin. EJB-sovelluspalvelin huolehtii komponentin hajautetusta käytöstä sekä palvelimen tarjoamien palveluiden käytöstä. Tällaisia palveluja ovat mm. tapahtumakäsittely (transactions), pysyvyys (persistence), samanaikaisuus (concurrence) ja turvallisuus (security). Ylläoleva kuva Java 2:n Enterprise Edition-arkkitehtuurista selkeyttää käsitteitä.



Kuva 2. Viestin välitys.

EJB-komponenttityypit

EJB-mallissa (EJB Specification, v1.1) voidaan toteuttaa seuraavan tyyppisiä komponentteja:

- tilaton, palvelutyyppinen komponentti
- Lyhyen aikaa tilallinen komponentti, joka säilyttää keskustelullista tilaa sovelluksen asiakkaan ja sovelluspalvelimen komponentin välillä, l. ”istunto”
- entiteetti-tyyppinen komponentti, joka mallintaa sovelluslogiikan käsitteitä ja on monen istunnon käytettävissä, esim. tuote-komponentti.

EJB-arkkitehtuuri määrittelee kaksi komponenttioliota, SessionBeanin ja EntityBeanin, joihin ylläolevat peruskomponenttityypit sisältyvät. Tyypillisinä SessionBeanin piirteinä voidaan pitää seuraavia ominaisuuksia:

- Toimii yksittäisen asiakasinstanssin edustajana, ”istuntona”

- Voi osallistua tapahtumankäsittelyketjuun
- voi päivittää jaettua dataa tietokannassa
- ei edusta suoraan tietokannan dataa
- on kohtalaisen lyhytikäinen
- sen sisältö säilyy EJB- säiliön eliniän.

Tyypillisiä EntityBeanin ominaisuuksia ovat taas seuraavat:

- tarjoaa oliomaisen näkymän tietokantaan
- sallii useiden käyttäjien käsiksipääsyn
- voi olla pitkäikäinen.

Nyrkkisääntönä voidaan pitää, että EntityBean-komponentit voidaan ajatella sovelluslogiikan konsepteina, esim. asiakas, tuote, jne (Monson-Haefel, s.23). EntityBeanit ovat tilallisia komponentteja. SessionBeanit voidaan taas mallintaa toimintoina, kuten paikanvaraus tms. SessionBeanin tila on normaalisti lyhytaikainen, ja sen hallinnasta vastaa ko. komponentin asiakas. On erikseen määritelty myös tilaton SessionBean, joka on yllämainituista komponenteista ensimmäistä tyyppiä, eli palvelutyyppinen komponentti. Tällaista komponenttia voidaan käyttää esim. silloin, kun EJB-sovelluksen osa toimii tiedon hakusovelluksena, jossa ei tarvitse säilyttää komponentin asiakassovelluksen tilainformaatiota.

Javan plussia ja miinuksia

Java on kielenä erittäin helppo oppia, ja sen käyttö on pyritty tekemään helpommaksi kuin esim. C:n tai C + + :n jättämällä pois piirteitä, jotka C/C + + -kielissä ovat tuntuneet aiheuttavan eniten ongelmia. Lisäksi Javan peruspaketti on piirteiltään erittäin laaja. Nämä asiat ovat kuitenkin osittain aiheuttaneet myös sen, että Java on tietyllä tavalla kompromissi. Java on ollut hitaampaa suorittaa kuin perinteisesti käännetyt ohjelmointikielet, mutta tänä päivänä Javassa päästään jo varsin kohtalaiseen suoritusnopeuteen. Palvelinympäristöön on erillinen HotSpot-virtuaalikone, joka optimoi suoritusta nopeammaksi, ja nyt myös työasemapuolelle on luvassa vastaava. Tämä parantaa suorituskykyä entisestään.

On kuitenkin hyvä muistaa, että Java on kielenä vielä varsin nuori, se on nykymuodossaan ollut olemassa vasta vuodesta 1995 lähtien. Tämä tarkoittaa myös sitä, että Javan komponenttiarkkitehtuuritoteutukset ovat nuoria, ja kehittyvät melkoisella vauhdilla.

Mikään tietty komponenttimalli ei takaa autuutta, vaan komponenttien suunnitteluun ja testaukseen tulee erityisesti panostaa. Tämän päivän kiihtyvässä sovellustuotantosyklissä testaus nousee entistä suurempaan arvoon. Java-maailmassa testaukseen alkaa olla hyviä apuvälineitä, ja Javan ollessa kyseessä komponenttitestaus tarkoittaa normaalin moduli- ja ohjelmistotestauksen lisäksi huolellista profilointitestausta, jolla vältetään Javan muistinhallinnan ja roskienkeruumekanismin tuottamat mahdolliset yllätykset.

Yhteenveto

Ohjelmistokomponentilla tai komponentilla tarkoitetaan valmiiksirakennettuja, valmiiksitestattuja, riippumattomia ja uudelleenkäytettäviä ohjelmamoduleita, jotka suorittavat rajattuja toimintoja.

Javassa on käytössä kaksi komponenttimallia, toinen käyttöliittymäsovelluksiin tarkoitettu JavaBeans-komponenttimalli, toinen sovelluspalvelinsovelluksiin tarkoitettu Enterprise JavaBeans-malli. JavaBeansmallia voidaan käyttää sekä Java AE:ssa että Personal Java AE:ssa, EJB:tä Java AE:ssa.

JavaBeans-malli tarjoaa ympäristöriippumattoman, vikasietoisen ja helppokäyttöisen GUI-komponenttiarkkitehtuurin. EJB-malli tarjoaa sovelluspalvelin-riippumattoman palvelinkomponentti-arkkitehtuurin. Javan etuina voidaan ympäristöriippumattomuuden ja helppokäyttöisyyden lisäksi pitää sen monipuolisuutta ja turvallisuutta.

Mikään komponenttimalli ei kuitenkaan takaa itsessään autuutta, vaan komponenttien suunnittelu ja testaus nousevat entistä tärkeämpään rooliin.

Lähdeluettelo ja luettavaa

Englander R.. Developing JavaBeans, O'Reilly & Associates. Sebastopol, CA, 1997.

Gosling J., Joy B., Steele G. Java Language Specification. Addison-Wesley, 1996.

McGraw G. ja Felten E. Java Security: Hostile Applets, Holes, and Antidotes. John Wiley & Sons, New York, 1997.

Monson-Haefel R. Enterprise JavaBeans, O'Reilly & Associates. Sebastopol, CA, 1999.

Sun Microsystems, Inc.. Enterprise JavaBeans Specification, v1.1. , 1999

Sun Microsystems, Inc. JavaBeans Specification, v1.01., 1997.

Sun Microsystems, Inc. PersonalJava Technology White Paper, 1998.

Szyperski C. Component Software – Beyond Object-Oriented programming. Addison-Wesley, Essex, 1998.

Simo Vuorinen,
TietoEnator Oyj

Enterprise JavaBeans

Komponentteja Java-palvelinsovelluksiin

Hannu Kokko,
SysOpen Object Team Oy

Tausta

Enterprise JavaBeans™ (EJB) on hajautettujen järjestelmien toteuttamiseen tarkoitettu Java-komponenttiarkkitehtuuristandardi. EJB:n määrityksen on tuottanut Sun yhdessä merkittävien kumppaneidensa kanssa (IBM, BEA, Oracle, GemStone, Netscape...). EJB-standardin ensimmäinen versio 1.0 julkaistiin huhtikuussa 1998.

Ensimmäiset standardia toteuttavat tuotteet, BEA WebLogic ja Persistence Software, valmistuivat elokuussa 1998. Tuorein versio standardista, EJB 1.1, valmistui syksyllä 1999. Lähes kaikki EJB -pohjaiset tuotteet tukevat tällä hetkellä EJB 1.0-standardia.

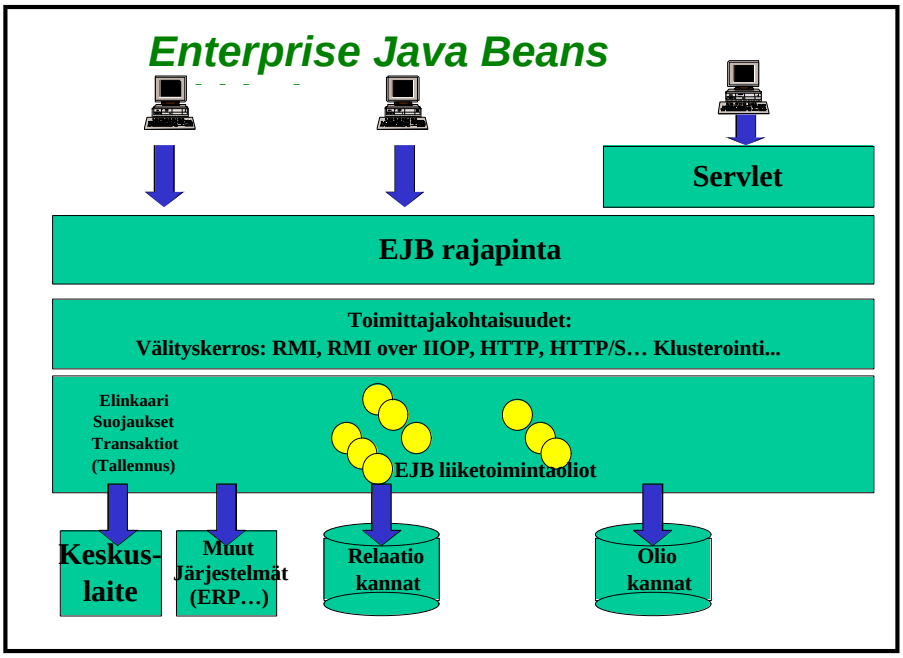
Palvelinpuolen Java-sovellusten rakentamisessa havaittiin jo varhaisessa vaiheessa standardoinnin tarve. Java-koodia palvelimessa ajavia palvelinohjelmistoja (sovelluspalvelimia) syntyi markkinoiden paineessa useita kymmeniä. Jokaisella palvelimella oli oma ohjelmointirajapintansa siihen, kuinka sitä käytetään työasemasta tai webistä ja toisaalta siihen kuinka liiketoimintakomponentteja rakennetaan.

Internet-aikana sovellukset perustuvat hajautettuun client-server –malliin, jossa asiakassovellusta ajetaan käyttäjän selaimessa. Potentiaalisen käyttäjämäärän kasvaessa suuri osa sovellusten palvelinosuudesta toteutetaan hajautettuna mm. skaalautuvuuden helpottamiseksi. Hajautustekniikat ovat usein varsin monimutkaisia toteutustasolla, ja myös Javalla hajautettujen sovellusten toteuttaminen on vaikeampaa kuin perinteisten client-server –sovellusten. Enterprise JavaBeans™ -standardi on kehitetty ratkaisemaan tätä ongelmaa.

Tavoitteet

EJB:n keskeiset tavoitteet ovat

- Helpottaa ja standardoida hajautettujen liiketoimintasovellusten kehitystä ja asennusta
- Erottaa businesslogiikka sovelluspalvelin-, ajoympäristö- ja systeemiohjelmointiläheisestä koodista (tietoliikenne- CORBA-, RMI - tai muu hajautusprotokolla, transaktio-ohjelmointi, säikeisyysohjelmointi)
- Saavuttaa kattava teollisuuden hyväksyntä
- Siirrettävät komponentit
- Sovelluspalvelimen valinta tilanteen mukaan ohjelmakoodia muuttamatta
- Monitoimittajayhteentoimivuus



Kuva 1: Enterprise JavaBeans arkkitehtuuri

Teknologiariippumattomuus

EJB:n keskeisenä tavoitteena on **riippumattomuus sovelluspalvelimista** ja laitealustoista, käyttöjärjestelmistä, jne. Riippumattomuus perustuu seuraaviin tekijöihin:

- Yksinkertaiset Enterprise JavaBeans -rajapinnat työasemassa ja komponenttien välillä
- Komponenttimalli
- Asennuskuvain
- Joukkovalmispalveluita (transaktiopalvelut, suojaukset, elinkaaripalvelut)
- RMI over IIOP -protokolla tuotteiden yhteiskäyttöön (EJB 1.1).
- Java2 Enterprise Editionin osa Enterprise JavaBeans -arkkitehtuurin mukainen sovelluspalvelin voidaan toteuttaa eri tekniikoilla. Tekniikkana voi olla esimerkiksi CORBA-palvelin (Inprise), tietokanta (Oracle 8I), tapahtumamonitori (WebLogic Enterprise), oliokanta (GemStone) tai sovelluspalvelin (WebLogic Application Server, WebSphere Advanced Edition). Toteutustekniikka ei saisi näkyä sovelluskehittäjälle sen paremmin työasema- kuin palvelinosuudenkaan toteutuksessa.

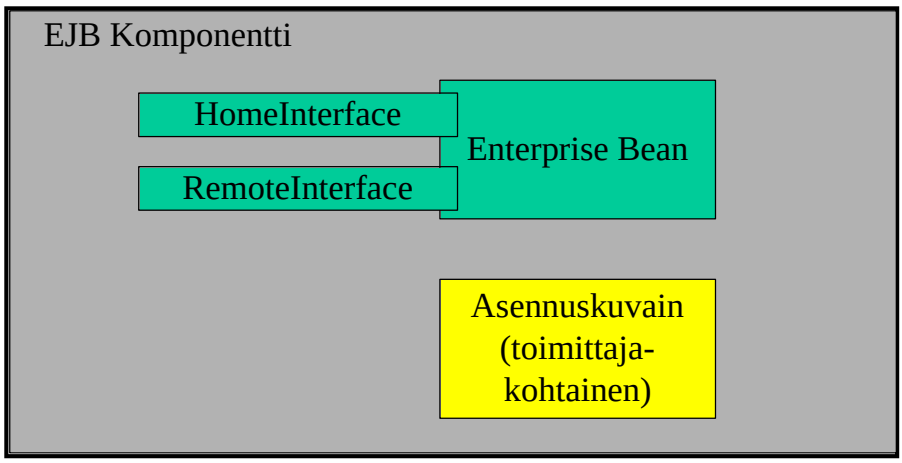
Komponenttimalli sovelluskehiyksessä

Enterprise Bean – businesslogiikan suorittaja

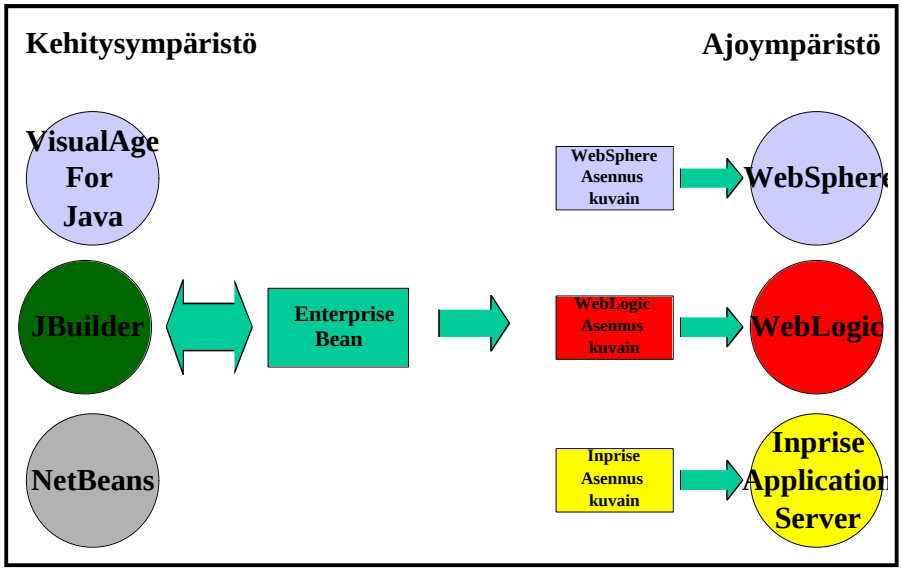
Enterprise Bean on Java-luokka.Java luokassa on toteutettu Enterprise Beanin tarjoamat liiketoiminta- tai tekniset palvelut.

Enterprise Bean voi olla joko

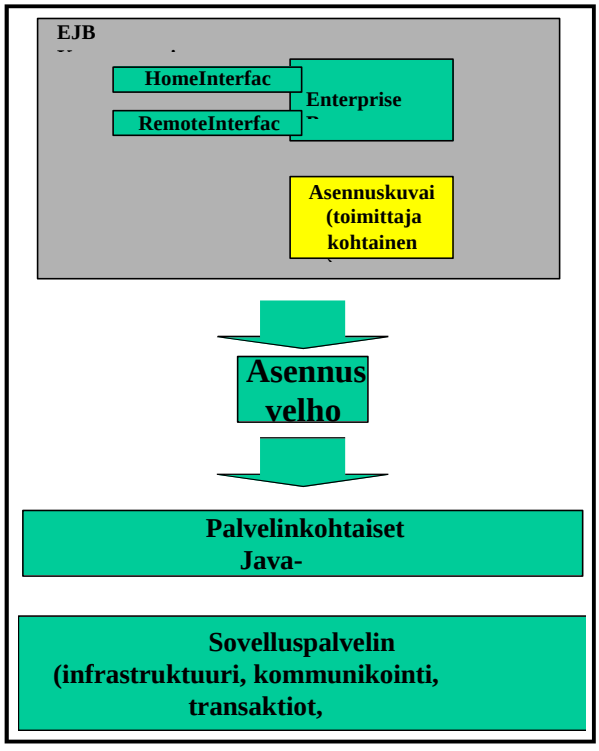
- tallentuva, tietokannan rivin kaltainen EntityBean
- käyttäjäkohtainen tilaton transaktion- tai talletetun proseduurin kaltainen SessionBean
- käyttäjäkohtainen muuttujien tilan käyttäjäistunnon ajan säilyttävä SessionBean



Kuva 2: EJB-komponentti



Kuva 3: Enterprise Bean ja riippumattomuus



Kuva 4 EJB-komponentin asennus

Enterprise Bean voi kutsua muita Java-luokkia. Enterprise Beanin koodi suoritetaan aina palvelimella.

Kotirajapinta – kuinka löydät komponentit

Enterprise Bean voidaan luoda tai hakea käsittelyyn kotirajapinnan (HomeInterface) kautta. Kotirajapinta sisältää etsintämetodit, joiden avulla löydetään esimerkiksi pääavaimella CustomerEnterpriseBean. CustomerHomeInterface rajapinnan etsintämetodin kutsu palauttaisi tässä tapauksessa CustomerRemoteInterface tyyppisen viitteen CustomerEnterpriseBeaniin.

Etärajapinta – kuinka kutsut businesslogiikkaa

CustomerRemoteInterface rajapinnassa esitellään ne CustomerEnterpriseBeanin businesslogiikkametodit, joita se antaa ulkopuolisten kutsua.

Asennuskuvain

Asennuskuvain sisältää Enterprise Beanin kuvauksen ajoympäristöä varten. Se sisältää mm. komponentin nimen, suojauskäytännöt, komponentin osien nimet, transaktiokäytäntöjen kuvaukset, jne. Asennuskuvain sisältää usein myös sovelluspalvelinkohtaista tietoa mm. siitä , kuinka komponentti tallennetaan tietovarastoon. Asennuskuvaimet ovat toistaiseksi kullakin palvelintoimittajalla erilaisia.

Sovellusohjelma

Sovelluskehittäjät kutsuvat komponentin businessmetodeja yllämainittujen rajapintojen kautta. Rajapintojen toteutukset yhteistyössä sovelluspalvelimen infrastruktuurin kanssa huolehtivat palvelupyyntöjen reitittämisestä, palvelujen käynnistämisestä, tiedon esitystavan muutoksista, suojauksista ja transaktioista.

Kutsuva ohjelma etsii kotirajapinnan avulla haluamansa Enterprise Beanit ja kutsuu etärajapinnan kautta liiketoimintametodeja. Jokainen kutsu tapahtuu verkon yli, jolloin sovellus on hajautettavissa halutulla tavalla. Tämä edellyttää kuitenkin sovelluksen rakenteen suunnittelua siten ettei verkosta tule pullonkaula.

Komponenttien asennus

EJB -pohjainen sovelluskehitys perustuu komponenttien rakentamiseen Javalla. Komponentti asennetaan haluttuun ajoympäristöön asennustyökalun avulla.. Asennustyökalu tuottaa annettujen parametrien (asennuskuvain) avulla palvelinkohtaisen koodin (sovittimet), jota ajetaan sovelluspalvelimessa.

Arkkitehtuuri

Enterprise JavaBean -arkkitehtuurin mukainen sovelluspalvelin tarjoaa valmiina joukon skaalautuvuuteen, turvallisuuteen, hajautukseen, transaktioihin, tietokantakäsittelyyn ja ylläpidettävyyteen liittyviä palveluita, jotka esimerkiksi CORBAssa jouduttaisiin tekemään itse.

Sovelluspalvelin tarjoaa aina peruspalvelut komponenttien kutsumiseen. Sovelluspalvelin voi tarjota myös mm. kuormantasausta, tietokantayhteyksiä, tietokantatallennusta, CORBA-yhteyksiä, http-tunnelointia tms., mutta nämä eivät kuulu EJB-standardiin.

Enterprise JavaBean arkkitehtuuri antaa rakentaa sovelluksia monella tavalla:

- Tilattomia palvelinsovelluksia, jolloin olioiden tilamuuttujien arvot säilytetään vain kutsuvassa ohjelmassa ja tietokannassa ja tiedot näiden välillä välitetään vain palvelupyyntöinä. Tällöin tarvittava tila siirretään palvelupyyntöön aina palvelupyynnön tapahtuessa. Sovelluksen rakenne palvelimessa muistuttaa tällöin esimerkiksi Tuxedo-, CICS- tai MTS-sovellusta.

- Tilallisia palvelinsovelluksia, jossa olioiden ja komponenttien tila säilyy kutsusta toiseen myös palvelimessa. Sovelluksen rakenne on tällöin CORBA- tai perinteisen oliosovelluksen kaltainen

- Tilallisen ja tilattoman yhdistelmiä

Suoraan tietokantaa (JDBC:llä) käyttävien sovellusten yhteydessä EJB antaa mahdollisuuden käyttää tallennusvelhoja tietokannan tallennuskoodin automaattiseksi kirjoittamiseksi. Tämä voi lisätä tuottavuutta merkittävästi. Tallennusvelhojen kyvyt vaihtelevat huomattavasti EJB-palvelimesta toiseen.

Enterprise JavaBeans -teknologian tulevaisuus

Enterprise JavaBeans -pohjaiset palvelimet ovat olleet nyt markkinoilla parhaimmillaankin vain puolitoista vuotta (elokuusta 1998). Palvelimissa käytetty tekniikka on useissa tapauksissa kuitenkin jo huomattavasti koestetumpaa, koska palvelimet on usein rakennettu jonkin olemassaolevan tuotteen päälle. Enterprise JavaBeans -standardi on voimakkaasti leviämässä, mutta laajojen sovellusten rakentaminen on paljolti vielä alkuvaiheessa.

EJB-standardi kehittyy jatkuvasti, mutta jo EJB 1.0 -standardin mukaisten sovelluspalvelinten varaan on rakennettu varsin suuria järjestelmiä. Standardin 1.0 -versiossa olevia puutteita on korjattu uudessa standardiversiossa 1.1. EJB 1.1 vaatii sovelluspalvelimia toteuttamaan enemmän EJB:n toiminnallisuudesta kuin 1.0. Se myös määrittelee sovelluksen ja sovelluspalvelimen tehtävät tarkemmin kuin aikaisempi standardiversio. Tässä yhteydessä siirrettävyys ja yhteentoimivuus paranee. EJB 1.0:ssa siirrettävyys vaati sovelluskehittäjältä enemmän työtä.

Markkinoilla on saatavissa useita kymmeniä sovelluspalvelimia, jotka tukevat EJB-standardia. Ainoa merkittävä sovelluspalvelintoimittaja, joka ei tue EJB:tä on Microsoft. Markkinoiden kypsymisen merkkejä alkaa näkyä siinä, että johtaviin tuotteisiin alkaa olla saatavissa add-on -tuotteita esimerkiksi tietokantojen käsittelyyn (TopLink, Versant..).

Tärkeimmät sovelluskehitysvälineet tukevat jo varsin hyvin Enterprise JavaBeans -teknologiaa (Rational Rose, VisualAge for Java Enterprise Edition, Visual Café Enterprise, NetBeans, Together/J, JBuilder Enterprise, Inline AssemblyLine).

Lähitulevaisuudessa voidaan odottaa seuraavia muutoksia

- Kaikki merkittävät EJB-palvelin- ja työkalutoimittajat tukevat EJB 1.1 ja Java2 Enterprise Edition -standardeja

- EJB -sovelluspalvelimien välinen yhteentoimivuus ja siirrettävyys paranee edellämainitun kehityksen toteutuessa

Yhteenveto

Yksi Enterprise JavaBeansin käytön keskeisiä etuja on muita hajautustekniikoita yksinkertaisempi sovelluskehitysmalli. Sovelluksen rakentajalla on nyt myös parempi mahdollisuus siirtää sovellus sovelluspalvelimesta toiseen. Kolmas tärkeä etu on ison tuotetarjonnan ja keskeisten toimittajien panostuksen seurauksena markkinoille kertyvä osaaminen Enterprise JavaBeans -standardin mukaisesta ohjelmoinnista, jota voidaan hyödyntää käytetystä sovelluspalvelimesta riippumatta. Neljäntenä etuna ovat merkit kaupallisten EJB-komponenttimarkkinoiden synnystä.

KTM Hannu Kokko

(Hannu.Kokko@sysopen.fi) toimii johtavana konsulttina ja toimitusjohtajana SysOpen Object Team Oy:ssä. Hän on työskennellyt EJB-komponenttiarkkitehtuurien parissa standardin syntyhetkiltä huhtikuusta 1998 lähtien ja hajautettujen tai oliojärjestelmien parissa viimeiset kymmenen vuotta.

Muutama valittu sana EJB-sovelluspalvelimista

Marko Saarinen,
BEA

Sovelluspalvelimet ja Enterprise Java Beans (EJB) –teknologia edustavat valtaavaa mahdollisuutta yritystason tietojärjestelmien kehitystyöstä vastaaville. Yritykset voivat hankkia kilpailuetua, mikäli ne kykenevät ottamaan nopeasti ja joustavasti käyttöön seuraavan sukupolven sähköistä liiketoimintaa tukevia sovelluksia. Tämä ei luonnollisesti yksinomaan riitä, vaan tarjottavien sähköisten palvelujen tulee kaiken lisäksi olla luotettavia, skaalautuvia sekä integroitavissa olemassa oleviin järjestelmiin. Koska käytettävällä sovelluspalvelinteknologialla on tässä suuri merkitys, herääkin kysymys, mihin ominaisuuksiin ja seikkoihin EJB- sovelluspalvelimen valinnassa tulee kiinnittää erityistä huomiota?

Vaikka markkinoilla on tällä hetkellä lukuisia eri Enterprise Java Beans –spesifikaatiota tukevia sovelluspalvelimia, voidaan suorituskyky-, skaalautuvuus-, luotettavuus- ja käytettävyysominaisuuksien osalta vakavasti otettavien sovelluspalvelinten määrä rajata alle kymmeneen. Tästä huolimatta valinta ei välttämättä ole kovinkaan yksinkertainen, sillä erityisesti useimpien yritystason sovelluspalvelinten ”kypsyminen” teknologiamielessä tulee viemään vielä noin 1 ½ - 2 vuotta. Tämä ei luonnollisestikaan tarkoita sitä, ettei markkinoilla olisi tällä hetkellä EJB-sovelluspalvelimia, joita voidaan käyttää kriittisten sähköistä liiketoimintaa tukevien transaktionaalisten sovellusten kehittämiseen, käyttöönottoon ja hallintaan. Sitä paitsi on syytä muistaa, ettei käytännöllisesti katsoen kenelläkään ole varaa odottaa yritystason EJB-sovelluspalvelinten teknologista kypsymistä pitkälti toista vuotta, vaan projektit on usein aloitettava mieluummin mahdollisimman pian kuin liian myöhään. Niinpä lopputuloksena on tilanne, jossa valinta kohdistetaan parhaiten nykyisiin sovellusvaatimuksiin istuvaan EJB-sovelluspalvelimeen.

EJB-palvelinten ”big picture”

EJB-sovelluspalvelin tarjoaa ympäristön, joka tukee Enterprise Java Beans –teknologialla kehitettyjen sovellusten suorittamista. EJB-palvelimen täytyy tarjota yhden tai useampia EJB-säiliöitä (EJB container), jotka puolestaan tarjoavat ”kodin” yritystason pavuille (enterprise beans). EJB-säiliön tehtävänä onkin hallita sen sisällä pyöriviä yritystason papuja. Jokaisen yritystason pavun kohdalla säiliö on vastuussa mm. objektin rekisteröinnistä, objekti-instanssien luomisesta ja tuhoamisesta, objektin tietoturvaan liittyvistä tarkistuksista, objektin aktiivisen tilan hallinnasta sekä hajautettujen tapahtumien koordinoinnista. EJB-säiliö voi myös hallita kaikkea persistenttia dataa objektin sisällä.

Tällä hetkellä kaikkien keskeisten sovelluspalvelintoimittajien tuotteet tukevat EJB 1.0 –spesifikaatiota. Eroja kuitenkin löytyy. EJB 1.0:ssahan tuki entity-pavuille on valinnainen, mikä käytännössä tarkoittaa sovelluspalvelimissa sitä, että jotkut toimittajat tukevat niitä kokonaan, jotkut osittain ja jotkut eivät lainkaan. Useimmat toimittajat ovat kuitenkin ratkaisseet tuen entity-pavuille ainakin osittain, jolloin siirtyminen täydelliseen tukeen myöhemmin on huomattavasti suoraviivaisempaa. EJB 1.1 –spesifikaatiohan tuo mukanaan pakollisen tuen entity-pavuille. Useimmille EJB-kehittäjillehän tämä on positiivinen asia, sillä se tuo mukanaan vakaamman alustan siirrettäville pavuille.

Oma lukunsa on myös EJB-tuen kypsyys eri sovelluspalvelimissa. Tällä puolestaan on tyypillisesti huomattava vaikutus siihen, kuinka monta tuotannossa olevaa – ja vieläpä toivottavasti riittävän suorituskyvyn, skaalautuvuuden, luotettavuuden ja käytettävyyden tarjoavaa - EJB-palvelintoteutusta kyseisellä toimittajalla on osoittaa.

Ensimmäinen EJB-tuen omaava sovelluspalvelin tuli markkinoille WebLogicilta (nykyisin sulautettu BEA Systems:iin) syyskuussa 1998. Tosin tämän jälkeenhän myös muilta toimittajilta on tullut ulos EJB-palvelimia.

Sovelluspalvelinten arvioinnissa voidaan tietysti käyttää useita eri kriteereitä, joiden kaikkien yksityiskohtainen läpikäynti tässä on mahdotonta. Huomiota kannattaa kuitenkin kiinnittää mm. tarjolla oleviin kehitystyökaluihin, komponenttien ajonaikaiseen ympäristöön sovelluspalvelimissa sekä sovelluspalvelimen hallinta-ominaisuuksiin ja sopivuuteen hajautettuihin ympäristöihin.

Kehitystyökalut

Markkinoilla on tällä hetkellä lukuisia Java IDE –työkaluja, joiden integraatiotaso eri sovelluspalvelinten kanssa vaihtelee varsin suuresti. Integraatiotason tuottavuutta ja kehitystyön lopputulosta parantavasta merkityksestä voidaan perustellusti olla monta mieltä, mutta eräs keskeinen integroituun EJB-sovelluspalvelin – Java IDE –kehitystyökalu –ympäristöön liittyvä hyvä puoli on mahdollisuus saada tarvittaessa tukea yhdeltä toimittajalta (edellyttäen, että käytettävä sovelluspalvelin ja kehitystyökalu ovat samalta toimittajalta).

Tämän lisäksi Java IDE –työkalujen tehokkaan tiimikehityksen mahdollistavissa ominaisuuksissa on myös suhteellisen merkittäviä eroja. Tiimikehitysominaisuuksien kehittäminen on kuitenkin lähes kaikilla merkittävillä työkalutoimittajilla korkealla prioriteetilla massiivisten hajautettujen kehitysprojektien yleistyessä.

Komponenttimalleihin liittyvien visioiden suhteen esiintyykin sitten suurempia eroja. Joillakin toimittajilla on kattava visio keskenään joustavasti kommunikoivista Java- ja C++-komponenteista Java/CORBA-infrastruktuurissa (EJB-viitekehyksessä). Haasteena kuitenkin on se tosiasia, että C++/CORBA- ja Java/EJB-komponenttien ajaminen samalla sovelluspalvelimella on riippuvainen kunkin toimittajan toteutustavasta. Niinpä tämän ominaisuuden hyödyntäminen johtaa tässä vaiheessa siihen, että sovelluksissa esiintyy jonkin verran ei siirrettävissä olevaa koodia. Tämä ainakin siihen saakka, kunnes meillä on virallinen CORBA-komponenttimalli, joka on riittävällä tasolla sovitettu yhteen EJB-mallin kanssa. Toisaalta on myös syytä muistaa, että tämän vision päällimmäinen tarkoitus ei välttämättä ole C++/CORBA-sovellusten yhteentoimivuus EJB-sovellusten kanssa IIOP:n avulla, vaan pikemminkin joustavampi ja saumattomampi kehityspolku eteenpäin C++/CORBA-sovelluksille. Sitä paitsi C++- ja EJB-komponenttien ajaminen samalla sovelluspalvelimella tuo C++-komponenteille joukon EJB-komponentteihin liitettäviä etuja kuten niiden paremman hallittavuuden ja korkeamman luotettavuustason.

Komponenttien ajonaikainen ympäristö

Komponenttien ajonaikaista ympäristöä arvioitaessa on syytä kiinnittää huomiota lukuisiin seikkoihin. Asiakaspäässä keskeisiä arvioitavia kriteerejä tulisi olla webbituki (dynaaminen HTML ja appletit), komponenttituki (Bean ja CORBA), Java-tuki (standardin mukaiset Java-virtuaalikoneet), XML-tuki (XML-rajapinta ja –data) sekä tuki natiivikäyttöliittymille (Windows ja UNIX).

Kaikki johtavat sovelluspalvelimet tarjoavat tuen sekä ”lihaville” että ”laihoille” asiakaspään laitteille. Vaikka ”lihavia” asiakaspään laitteita toisinaan käytetäänkin edelleen mm. call center –ratkaisuissa, on niissäkin enenevässä määrin siirrytty webbipohjaisiin käyttöliittymiin ja portaaleihin, joihin yleisimmin käytettävät sovellukset ja palvelut on keskitetty, ja joista ne voidaan käynnistää.

Se, mitä Java Development Kit (JDK) –versiota sovelluspalvelimet tukevat, vaihtelee myös. Kaikissa toteutuksissa ei välttämättä vielä ole siirrytty Java 2:een (JDK 1.2), vaan tuki löytyy edelleen enintään JDK 1.1.7:lle. Syynä tähän on toisinaan nykyisin käytettävä, suorituskyvyn kannalta optimoitu Java-virtuaalikone, joka on rakennettu 1.1.7:n päälle, tai sitten halu antaa uusimman saatavilla olevan Java-virtuaalikoneen kypsyä suuremman luotettavuustason saavuttamiseksi.

Palvelinpäässä huomioitavia seikkoja ovat mm. nimi- ja tietoturvapalvelut, tilanhallinta, klusterointi- ja kuormantasausominaisuudet sekä tuki eri ohjelmointimalleille ja –kielille. Varsinkin järeimmän pään sovelluspalvelimissa on selkeänä suuntauksena tuki hyvin kattavalle skaalalle eri ohjelmointimalleja ja –kieliä (CORBA, EJB, Java, C++ sekä jopa C ja Cobol) yhden integroidun tuotteen sisällä. Toinen yleinen suuntaus on tapahtumankäsittelymonitorin käyttö pohjalla parantamassa alustan vakautta. Vaikka näitä toteutuksia ei markkinoilla vielä kovin paljoa olekaan, kannattaa nykyisenkin tarjonnan puitteissa kuitenkin pitää mielessä se, että niiden suorituskyvyssä ja luotettavuudessa on vielä tässä vaiheessa suuria eroja.

Myös klusterointi- ja kuormantasausominaisuuksissa on eri EJB-palvelinten välillä suuriakin eroja. Jotkut sovelluspalvelimet tarjoavat pelkän webbisivu-klusteroinnin, kun taas edistyksellisimmissä sovelluspalvelimissa on myös tuki EJB-komponenttien klusteroinnille korkeamman suorituskyvyn ja vikasietoisuuden saavuttamiseksi. Myös mahdollisuus lisätä uusia palvelimia klusteriin dynaamisesti käytön aikana on tärkeä ominaisuus kriittisissä ympäristöissä, samoin mahdollisuus hallita klusteria yhdestä konsolista, josta käsin klusterin yksittäisiä palvelimia voidaan käynnistää ja ajaa hallitusti alas ilman, että tällä on vaikutusta klusterin muihin palvelimiin.

Olennainen osa sähköisen liiketoiminnan ratkaisua on sovellusten integroiminen olemassa oleviin taustajärjestelmiin, olivatpa ne sitten relaatiotietokantoja, ERP-sovelluksia tai suurkonepohjaisia järjestelmiä. Tälläkin alueella on syytä kiinnittää erityistä huomiota toimittajien integraatioratkaisujen vahvuuksiin sekä niiden kypsyyteen. Esimerkiksi suurkoneliitettävyyden kohdalla tuki sekä API-kutsu tyyliselle että ”screen scraping” tyyliselle integraatiolle on perusteltua, sillä suurkonetapahtumissa ei aina ole mahdollista enabloida API:a.

Unohtaa ei myöskään sovi webbipalvelinintegraatiota. EJB-sovelluspalvelinten kohdalla eräs keskeinen erottava tekijä onkin niiden tuki yrityksille, jotka hyödyntävät Microsoftin Internet Information Server (IIS) / Active Server Pages (ASP) –arkkitehtuuria. Johtavat sovelluspalvelintoimittajat tarjoavat jo tällä hetkellä COM-rajapinnat EJB:ille, joita voidaan kutsua IIS:n päällä ajettavista ASP:ista käsin.

Hallintaominaisuudet ja sopivuus hajautettuihin ympäristöihin

Hallinnan kannalta keskeisiä arvioitavia kriteereitä ovat mm. tuetut hallintakonsolit, integroitavuus hakemistopalveluihin, tietoturvainfra sekä resurssinhallintaominaisuudet. Yleisesti ottaen kaikkien johtavien EJB-palvelinten hallintaominaisuudet ovat hyvät. Mahdollisuus integroida palvelimet osaksi SNMP-yhteensopivaa järjestelmänhallintaympäristöä on kuitenkin tärkeä tekijä korkean käytettävyyden varmistamiseksi.

Toinen tärkeä osa-alue EJB-palvelimia arvioitaessa on niiden sopivuus hajautettuihin ympäristöihin. Tällöin on syytä kiinnittää huomiota mm. tuettuihin protokolliin, mahdollisiin arkkitehtuureihin, tuettuihin verkkoympäristöihin sekä luonnollisesti tuettuihin palvelin- ja asiakaspään käyttöjärjestelmiin. Kuten aikaisemmin jo mainitsinkin, on tuki sekä ”lihaville” että ”laihoille” asiakaspään toteutuksille tärkeää, mikäli halutaan liikkumavaraa toteuttaa esim. call center –tyylinen ratkaisu, jossa hyödynnetään Windows-työasemia ja niiden natiiveja Windows-käyttöliittymiä.

Edellä on luotu jonkin verran ajatuksia siitä, mihin suuntaan markkinoilla olevat EJB-sovelluspalvelimet ovat kehittyneet ja kehittymässä sekä siihen, millä kriteereillä niitä kannattaa arvioida. Kriteereitä on toki useampiakin kuin mitä yllä olen listannut. Jokaisen yrityksen on mietittävä ne ”oikeat” ja strategian kannalta keskeiset ominaisuudet, sekä valittava ratkaisu, joka parhaiten istuu senhetkisiin vaatimuksiin. EJB-palvelimet kehittyvät ja muuttuvat varsin nopeaa vauhtia, ja aivan lähiaikoina voimme odottaa markkinoille päivityksiä nykyisiin sovelluspalvelimiin tukemaan huomattavasti nykyistä kattavammin J2EE-platformia sekä sen alla olevia rajapintoja ja teknologioita.

Marko Saarinen
Marketing Manager
BEA Systems Oy
marko.saarinen@bea.com

Kilpailuetua resurssien optimoinnilla

Petteri Manninen,
SEC

Johdanto

Kaikilla teollisuuden ja liiketoiminnan alueilla asiakkailla on kasvavassa määrin erilaisia tarpeita, joihin pitää pystyä vastaamaan kilpailijoita tehokkaammin kehittyneillä tuotteilla ja palveluilla. Tämä edellyttää yritystoiminnan, asiakaspalvelun ja resurssien tuottavuuden tehostamista. Samalla kuitenkin kustannukset lisääntyvät ja ratkaistavat resurssiongelmat tulevat entistä monimutkaisemmiksi. Tällaiset ongelmat vaativat kehittyneitä optimointityökaluja, jotta kustannuksia voitaisiin vähentää, ylimääräisen raakaaineen määrää minimoida, läpimenoaikoja lyhentää ja toimituksia nopeuttaa. Yrityksen tulos, siinä missä tappiokin, riippuu ratkaisevasti resurssien optimoinnista.

Esimerkiksi lentotoiminnassa varataan kullekin lennolle optimaalisin konetyyppi, lentoreitti sekä kentillä lähtö- ja tuloportit, matkatavaroiden kuljetushihnat ja tarvittava henkilöstö. Metsäyhtiöt hyödyntävät reitinoptimointia tukinkuljetuksissa hakkuualueilta sahoille. Teollisuustuotannossa voidaan optimoida esimerkiksi terässulaton ajoitusta, jotta saavutettaisiin maksimaalinen teräksen läpimeno sekä mahdollisimman lyhyet odotusajat asetettujen turvallisuusrajoitteiden puitteissa. Jos tehtaassa on käytössä useita työkoneita, työmääräykset pitää pystyä jakamaan niin, että koneiden kuormitus on mahdollisimman tasainen. Tietoliikenteessä pyritään mahdollisimman nopeasti reitittämään datapaketteja verkon solmujen läpi, suunnittelemaan esimerkiksi mobiiliverkon tukiasemien paikat kuuluvuuden kannalta optimaalisesti sekä varaamaan niiden huoltoon käytettäviä henkilöresursseja. Pankki- ja vakuutusalalla optimointia hyödynnetään muun muassa järjestelemään mainframeympäristöjen operaattoreiden työaikoja ympäri vuorokauden tai järjestelemään yöllisiä eräajoja sopivimpaan järjestykseen. Lista ei lopu tähän, vaan esimerkiksi sähköisessä liiketoiminnassa hyödynnetään optimointia sopivien tuotekonfiguraatioiden löytämisessä asiakkaan vapaasti määrittelemien kriteerien mukaan. Asiakas voi esimerkiksi suunnitellessaan autohankintaa määritellä web-liittymässä mitä auto saa enintään maksaa ja mitä vähimmäisvarusteita hän siihen toivoo ja miltä alueelta haluaa sen mahdollisesti hankkia, jolloin järjestelmä pystyy annettujen rajoitteiden puitteissa etsimään sopivimmat vaihtoehdot. Piensijoittaja voisi samaan tapaan olla yhteydessä vaikkapa pörssiin, ja tehdä erilaisia analyysejä ja punnita osakkeiden oston tai myynnin erilaisia vaihtoehtoja WAP-puhelimellaan.

Tässä artikkelissa käydään läpi resurssien optimoinnin vaatimuksia ja erilaisia toteutustapoja. Näistä toteutustavoista kaksi keskeisintä on sisällytetty komponenttipohjaiseen toteutukseen, Ilogin Optimization Suite:en, joka on kyseessäolevien ongelmien, kuten resurssien varauksen, suunnittelun ja ajoituksen esittämiseen sekä ratkaisemiseen tarkoitettu kokoelma oliopohjaisia ohjelmistokomponentteja. Se perustuu operaatioanalyysin lineaariseen ohjelmointiin (linear programming) sekä uudempaan rajoiteohjelmointiin (constrained-based programming) ja sitä on käytetty perustana useissa onnistuneesti toteutetuissa liiketoimintakriittisissä sovelluksissa eri puolilla maailmaa. Lopuksi esitellään Ilogin optimointikomponenttien hyötyjä käytännön ongelmien ratkaisussa.

Optimointi ja kombinaatio-ongelmat

Optimoinnilla tarkoitetaan prosessia, jonka tuloksena saadaan paras ratkaisu tai optimi erilaisten mahdollisten ratkaisujen joukosta. Optimiratkaisut jakautuvat kahteen luokkaan: globaaliin ja paikallisiin optimeihin. Globaali optimi tarkoittaa kaikkien mahdollisten ongelman ratkaisujen joukosta parasta ratkaisua, kun taas paikallinen optimi on paras ratkaisu lähellä toisiaan olevien ratkaisujen joukosta. Paikallinen optimi on riippuvainen haun lähtöpisteestä, kun taas globaali optimi on aina sama, mutta sen löytäminen vaatii yleensä huomattavasti enemmän prosessointitehoa.

Joissakin sovelluksissa globaalin optimin löytäminen on lähes mahdotonta tai sitten ei ole olemassa tapaa todeta onko löytynyt ratkaisu globaali optimi. Kuitenkin paikallisen optimin löytäminen saattaa usein osoittautua hyödylliseksi, koska monesti hyvän ratkaisun löytäminen nopeasti on käytännön kannalta parempi kuin ajan hukkaaminen globaalin optimin etsimiseen. Itse asiassa joskus minkä tahansa asetetut rajoitteet täyttävän ratkaisun löytäminen nopeasti voi osoittautua liiketoiminnallisesti hyödylliseksi.

Resurssien optimoinnissa on tyypillisesti useita päätöstä vaativia valintoja, rajoitettu joukko liiketoiminnallisia tai muita seurauksia sekä joukko sääntöjä ja erilaisia rajoitteita. Rajoitteista osa voi olla lineaarisia eli matemaattisesti kuvattavia ja osa mahdollisesti loogisia eli ehdollisia päättelyitä. Lisäksi ne jakaantuvat usein vielä pakollisiin ja vapaisiin rajoitteisiin, joista enemmän myöhemmin. Tämäntyyppisiä ongelmia kutsutaan ns. kombinaatio-optimointiongelmiksi (COP = combinatorial optimization problems). Kaikki johdannossa mainitut esimerkit ovat tyypillisiä COP-ongelmia, jotka voidaan esittää päätösmuuttujien, täytettävien rajoitteiden sekä sopivan kustannusfunktion tai useiden liiketoimintatavoitteiden avulla. Esimerkiksi lentokentällä toimivassa lentoporttien ja kuljetushihnojen optimointijärjestelmässä jokaisella lennolla on

tuloporttia kuvaava päätösmuuttujansa. Porttien varausongelman ratkaisussa näille päätösmuuttujille etsitään mahdolliset uudet arvot, kun samalla otetaan huomioon ehdottomat rajoitteet (esim. turvallisuusohjeet yms.) ja ns. vapaat rajoitteet (esim. lentoyhtiöiden normaalisti käyttämät portit, kenttähenkilökunnan toivomukset yms.). Järjestelmän pitää kyetä vastaanottamaan uusia tietoja mahdollisesti myöhästyneistä tai peruuntuneista lennoista eli muuttuneista rajoitteista ja ajamaan tämän jälkeen optimointiajo yhä uudelleen ja uudelleen.

Ratkaisun pitäisi mahdollisuuksien rajoissa pystyä optimoimaan kustannusfunktio sekä sopia mahdollisimman moneen vapaaseen rajoitteeseen niiden tärkeysjärjestyksen mukaan. Koska COP-ongelmilla on tyypillisesti valtava määrä erilaisia mahdollisia ratkaisuja, joista vain osa on järkeviä ja vielä pienempi joukko optimaalisia, käytettävien hakualgoritmien on oltava erittäin tehokkaita. Tässä artikkelissa lyhyesti esiteltävän Ilogin Optimization Suiten vahvuutena on nimenomaan sen suorituskyky sekä laajennettavuus.

Optimointiongelmien kuvaaminen

Jos kaikki päätösmuuttujat ovat reaalilukuja, ja tavoite sekä rajoitteet lineaarisia funktioita, lineaarinen ohjelmointi yleensä riittää ongelmanratkaisuun. Reaalimaailman ongelmissa on usein kuitenkin mukana epälineaarisia yhtälöitä, joukko-opillisia ja loogisia muuttujia sekä rajoitteita. Loogisista rajoitteista esimerkkinä ovat prosessien vaiheiden aikataulutukset, joissa vaiheilla on tietty pakollinen suoritusjärjestys. Ongelman muodostaminen ja kuvaaminen päätösmuuttujien, tavoitefunktion ja rajoitefunktioiden avulla onkin optimoinnin kannalta oleellisin vaihe, koska sen perusteella voidaan valita käytettävä optimointimenetelmä.

Päätösmuuttujat

Päätösmuuttujilla kuvataan päätettäviä valintoja, kuten esimerkiksi kuinka suuri osuus liuoksesta on vettä, mikä komponentti kolmesta vaihtoehdosta (A, B, C) valmistetaan ensin, mille portille lentokone ajetaan, mille verkon solmulle datapaketti ohjataan jne. Muuttujatyypit voivat olla kokonaislukuja, reaalisia, loogisia, vaihtoehtoisia tai joukkoja.

Tavoitefunktio

Tavoite- tai kustannusfunktio määrittää tuloksen tietyillä päätösmuuttujien arvoilla. Esimerkiksi liuokseen käytettävien aineiden hankinnan ja prosessoinnin kustannukset, lentokonetyyppien (fleet) kulutus tietyllä lentoreitillä vähennettynä matkustajien tuomilla tuloilla jne. Tavoitefunktiot ovat tavanomaisesti lineaarisia (esim. $c(x_1,x_2,...,x_n) = a_1x_1 + a_2x_2 + ... + a_nx_n$), mutta monimutkaisissa optimointiongelmissa ne voivat olla myös epälineaarisia (esim. $c(x_1,x_2) = \log(x_1) + \sin(x_2)$). Riippuen tehtävästä optimi on joko tavoitefunktion minimi tai maksimi.

Rajoitteet

Rajoitefunktiot määrittelevät loogiset tai fyysiset ehdot, jotka päätösmuuttujien pitää täyttää. Esimerkiksi liuoksen suhteellinen veden osuus on oltava vähintään 20%, ohjelmistotuotannossa projektipäällikkö haluaa tietää milloin tietyt vaiheet voidaan aloittaa, mutta tiedetään, että esimerkiksi testausvaihetta ei voida aloittaa ennen kuin komponentin palvelinosa on toteutettu, lentokapteenien vähimmäislepoaika on x tuntia lentojen välillä jne. Osa rajoitteista on esimerkiksi käytännön syistä tai lain mukaan pakollisia, mutta myös ns. vapaita rajoitteita (relaxed constraints) voidaan määrittää. Niiden ehdoton toteutuminen ei ole lopputuloksen kannalta välttämätöntä, mutta niiden tärkeysjärjestys voidaan silti asettaa. Vapaat rajoitteet ovat tyypillisesti erilaisia johdon tai asiakkaiden asettamia toivomuksia tai preferenssejä. Esimerkkejä preferensseistä voisivat olla lentokentän tiettyjen porttien varaaminen tietyille lentoyhtiöille ja huoltohenkilöstön haluttomuus työskennellä peräkkäisinä viikonloppuina jne.

Optimointiin käytettäviä tekniikoita

Eräs käytetyimpiä optimointiongelmien ratkaisutapoja on ollut kehittää kutakin tiettyä ongelmaa varten aina oma algoritminsa. Etuna on ollut luonnollisesti kaiken kyseistä ongelma-aluetta koskevan tietämyksen kokoaminen sovellusta varten. Kääntöpuolena on joustamattomuus ja yleensä tuottamaton kehityskaari, koska tyydyttävän lopputuloksen aikaansaaminen on voinut kestää niin pitkään, että liiketoiminnassa on ehtinyt tapahtua jo ratkaisevia muutoksia kehityksen aikana. Toisaalta uusien ominaisuuksien, rajoitteiden ja päätösvaihtoehtojen lisääminen on voinut osoittautua hankalaksi, koska alkuperäisestä kehitystiimistä ei ole ollut enää toteuttajia käytettävissä.

Simulointiohjelmistot

Toinen tapa optimointiongelmien ratkaisuun on käyttää valmiita simulointi- ja resurssointiohjelmistoja. Nämä työkalut löytävät ajoituksen ja resurssoinnin puutteellisuudet ja muut kriittiset rajoitteet, mutta ne pystyvät parhaimmillaankin löytämään ainoastaan yhden ratkaisun ja arvioimaan tulosta sen perusteella. Simulointiohjelmistot riittävät yksinkertaisten ongelmien arviointiin, kun ratkaisun löytäminen onnistuu yleensä manuaalisesti. Monimutkaiset optimointiongelmat, toistensa kanssa ristiriitaiset rajoitteet ja mahdollisuus paremman ratkaisun hakemiseen vaativat resurssointisovelluksia, joiden ominaisuudet pystytään iteratiivisesti räätälöimään yrityksen tarpeisiin ja vaatimuksiin. Kehitykseen käytettävä aika, kustannukset ja sen aiheuttamat riskit on samalla kuitenkin pystyttävä minimoimaan. Seuraavassa on esitelty nämä vaatimukset paremmin täyttäviä vaihtoehtoja.

Asiantuntijajärjestelmät

Asiantuntijajärjestelmät (expert systems) tarjoavat joustavan tavan

mallintaa ongelma-alueen rajoitteet ja muuttujat. Ne soveltuvat tiedon suodattamiseen sekä yhtäläisyyksien etsimiseen ja niillä on suhteellisen yksinkertaista kuvata ongelmanratkaisuun liittyvää opittua tietoa. Niinpä asiantuntijajärjestelmiä käytetäänkin esimerkiksi diagnostisoinnin apuvälineinä lääketieteessä ja tietoliikenteessä. Asiantuntijajärjestelmät kuitenkin harvoin kykenevät suorituskykyyn, jota liiketoimintakriittisiltä optimointisovelluksilta odotetaan.

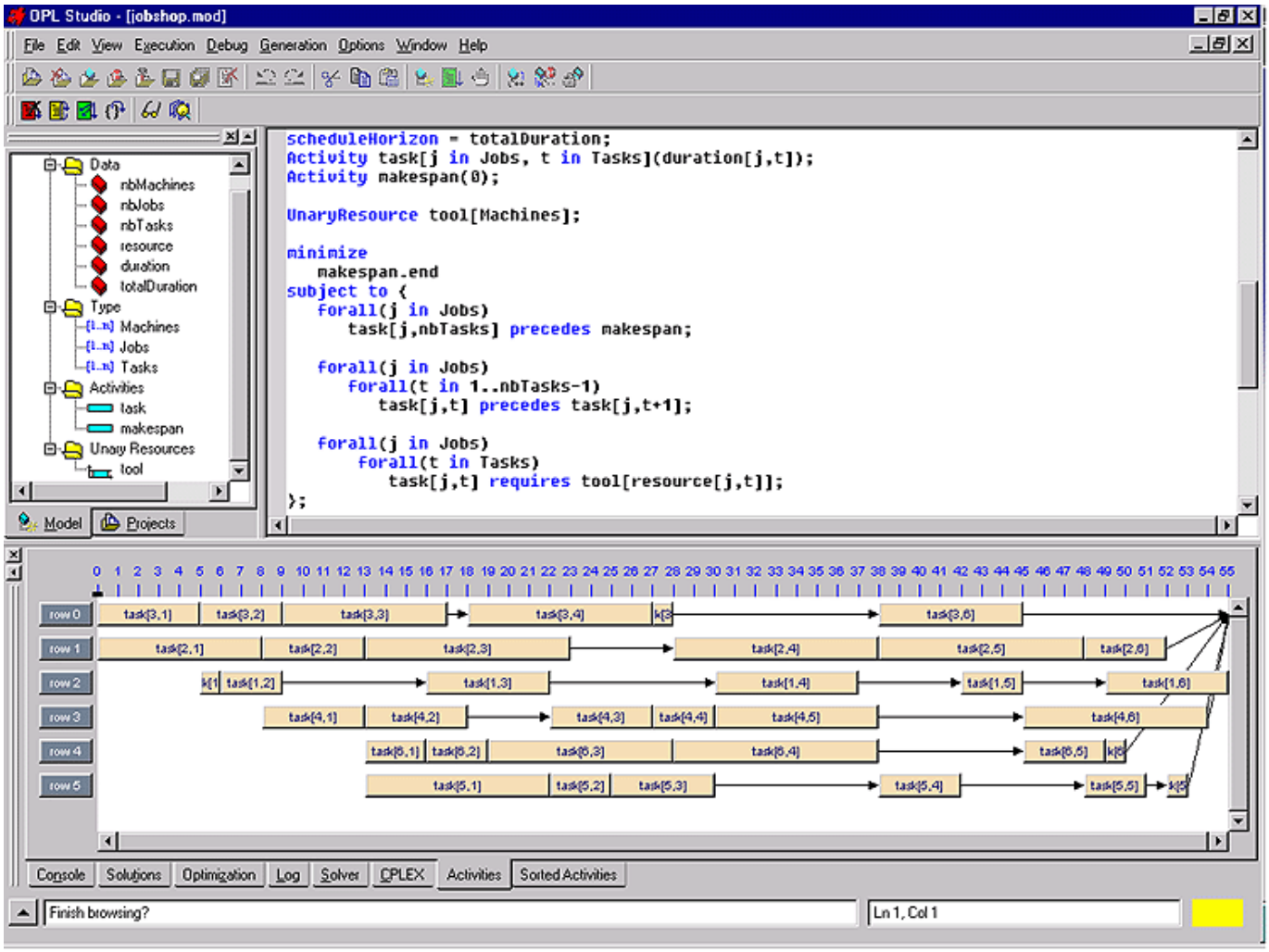
Lineaarinen ohjelmointi

Operaatioanalyysi tuntee useita tehokkaita ohjelmointimalleja, joita käytetään monien optimointiongelmien ratkaisuissa. Lineaarinen ohjelmointi (LP = linear programming, joka tunnetaan myös Simplex-algoritmin nimellä), neliöinti (QP = quadratic programming) ja sekoitettu kokonaislukuoptimointi (MIP = mixed integer programming) ovat erittäin tehokkaita ja hyödyllisiä lähes kaikkien optimointiongelmien ratkaisuissa. Ilogin Optimization Suite käsittää nk. CPLEX-tuotelinjan, jota käytetään ratkaisemaan lineaarisia optimointiongelmia muun muassa kuljetuksen, tietoliikenteen, teollisuustuotannon, rahoituksen, puolustusteollisuuden sekä energiantuotannon aloilla. Koska reaalimaailman optimointiongelmat kuitenkin sisältävät usein monimutkaisempia rajoitteita, mitä lineaarisella optimoinnilla pystytään ratkaisemaan, kehitettiin nk. rajoiteoptimointi (constrained-based optimization). Sillä voidaan edelleen laajentaa lineaarisen ohjelmoinnin käyttöaluetta monimutkaisempien ongelmien ratkaisussa.

Rajoiteoptimointi

Töiden ajoitus ja suunnittelu ovat optimointiongelmia, joita puhtailla lineaarisilla rajoitteilla ei pystytä riittävän helposti kuvaamaan. Esimerkiksi lentoyhtiöt joutuvat päivittäin tekemään valintoja konetyyppien välillä tietyille reiteille, riippuen ennustetusta käyttöasteesta. Tämä voidaan esittää lineaarisena ongelmana, mutta valittaessa tämän konetyypin tiettyä yksittäistä konetta ja sen reittiä, vaaditaankin monimutkaisempia yhtälöitä. Monivaiheiset päättelyt, kuten mitkä tehtaat rakennetaan ja sen jälkeen mitkä tuotantokoneet asennetaan kuhunkin tehtaaseen, ovat lineaarisina liian monimutkaisia esittää.

Rajoiteoptimointi pyrkii tarjoamaan tehokkaita algoritmeja, joiden ohjelmointirajapintoja (C++ API) hyödyntämällä järjestelmän toteuttaja pystyy yhdistämään operaatioanalyysin tehokkuuden ja asiantuntijajärjestelmän joustavuuden mallien tuottamisessa. Optimointityökalut tarjoavat yksinkertaisen tavan esitellä päätösmuuttujat ja monimutkaisetkin rajoitteet. Nämä



Kuva 1. Ilogin OPL Studiolla voidaan mallintaa optimointiongelmia visuaalisesti.

lähtötiedot erotetaan selkeästi itse ongelmanratkaisuun käytettävistä algoritmeista. Rajoiteoptimointiin perustuvat järjestelmät skaalautuvat hyvinkin monimutkaisiin optimointiongelmiin ja ne pystyvät tuottamaan ratkaisuja tehokkaammin kuin muut menetelmät. Lisäksi ne tarjoavat yksinkertaisen tavan laajentaa valmiin sovelluksen muuttujia, rajoitteita ja tavoitefunktioita jopa ajonaikaisesti. Rajoitteita voidaan vapauttaa tai kiinnittää, ja tuloksia voidaan tarkastella väliaikaisesti käyttäen hyväksi "entä-jos"-tekniikkaa. Ilogin Optimization Suite perustuu rajoiteoptimointiin ja se tarjoaa C++-komponentteja, joita käytetään API-rajapinnan kautta, jolloin valmiit testatut komponentit ovat yhä uudelleen hyödynnettäviä uusissa projekteissa. OPL Studio on puolestaan työkalu, jolla voidaan helposti mallintaa ongelman muuttujia ja rajoitteita ilman toteutuskoodin kirjoittamista.

Ilogin rajoiteoptimointikomponentin (Solver) tehokkuus perustuu ns. constraint propagation-tekniikkaan, joka käy optimointiajon aikana läpi rajoitteiden ristiriitaisuuksia ja muokkaa niitä niin, että ne ovat vuorovaikutuksessa toistensa kanssa mahdollisia. Tämä supistaa merkittävästi tutkittavia vaihtoehtoja ja nopeuttaa ratkaisun löytymistä. Esimerkkinä huomataan, että seuraavat tehtävälle annetut yksinkertaiset rajoitteet eivät ole käytännössä mahdollisia:

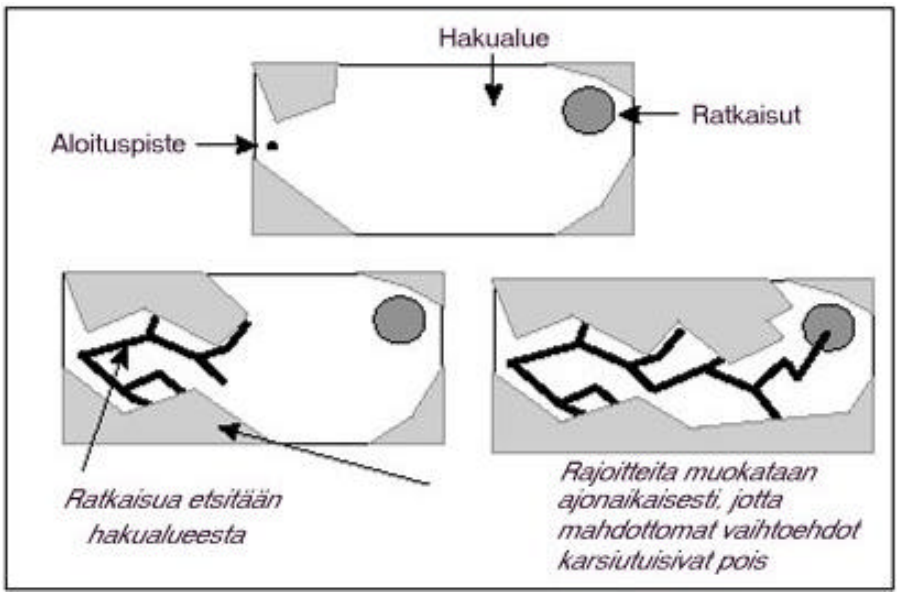
- $(1 < x < 10)$
- $(3 < y < 12)$
- $(x > y)$

Niinpä niitä muokataan optimointiajon aikana muotoon:

- $(4 < x < 10)$
- $(3 < y < 9)$
- $(x > y)$,

ja vasta tämän jälkeen yritetään löytää tavoitefunktion mukainen optimiratkaisu hukkaamatta aikaa turhiin päätösmuuttujien arvoihin.

Mainittujen tekniikoiden lisäksi optimoinnissa hyödynnetään jonkin verran mm. geneettisiä algoritmeja ja erilaisia simulointitekniikoita, joiden tarkoituksena on parantaa löydettyä ratkaisua ajan myötä. Näiden algoritmien hyödyt saadaan esille erityisesti, kun ongelman päätösmuuttujat ovat tyypiltään diskreettejä ja vaihtoehtoisia. Esimerkiksi satunnaisesti valitsemalla eri vaihtoehtoja voidaan tutkia ratkaisujen optimaalisuutta, mutta näiden luonnollisia prosesseja jäljittelevien tekniikoiden käyttö ei ole parhaimmillaan silloin, kun päätösmuuttujille voidaan antaa rajaton määrä erilaisia arvoja.



Kuva 2. Rajoiteoptimoinnissa voidaan hyödyntää ns. constraint propagation-tekniikkaa.

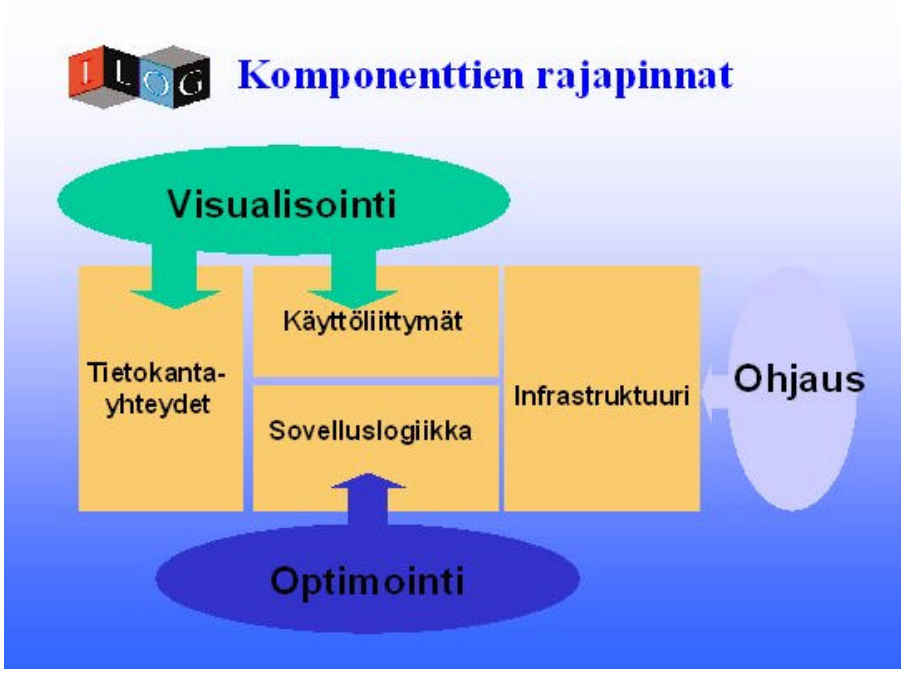
Ilog Optimization Suite

Rajoiteoptimointi yhdistettynä lineaarisen optimoinnin menetelmiin on monissa yhteyksissä todettu tehokkaimmaksi tavaksi toteuttaa mitä erilaisimpia optimointisovelluksia. Ilogin Optimization Suite on esimerkki komponenttipohjaisesta toteutuksesta, joka perustuu mainittuihin tekniikoihin. Ilogin kilpailukykyä komponenttitoimittajana parantavat myös joustavat C++- ja Java-käyttöliittymäkomponentit sekä integraatio relaatiotietokantoihin. Optimoinnin tuloksena syntyneen tietyn päätösmuuttujien yhdistelmän vaikutukset kaikkiin valinnasta aiheutuviin tuloksiin, kuten vaikkapa laitteiden hyötysuhteisiin tai töiden ajoitukseen saadaan helposti visualisoitua.

Interaktiiviset hakumahdollisuudet ja reaktiivisuus

Pelkkä mahdollisuus määritellä rajoitteita ja päätösmuuttujia ei aina riitä käytännön sovelluksissa, koska käyttäjien täytyy pystyä myös vaikuttamaan suunnittelusovelluksen ja optimoinnin toimintaan. Esimerkiksi useiden suunnitelmien, erilaisten skenaarioiden, simulaatioiden ja "entä-jos"-analyysien tekeminen sekä rajoitteiden ajonaikainen vapauttaminen (relaxing) tai kiinnittäminen (fixing) ovat ominaisuuksia, joita vuorovaikutteisilta ohjelmistoilta odotetaan. Reaktiivisuudella tarkoitetaan, että uuden päätöstiedon ja uusien rajoitteiden tuominen sekä tärkeysjärjestyksen muuttaminen pitää olla mahdollista ajonaikaisesti.

Ilogin Optimization Suite käyttää Solver-komponentissa nk. manager-oliota, joka mahdollistaa rajoitteiden lisäämisen ja poistamisen, nykyisen ratkaisun osittaisen tai kokonaisen tallentamisen sekä jonkin tällaisen talletetun ratkaisun palauttamisen.



Kuva 3. Ilogin valmiskomponenttien liitynnät.

Hakustrategioiden ohjelmointi

Hakustrategia on erillään päätösmuuttujista sekä rajoitteista, mikä mahdollistaa hakualgoritmien ohjelmoinnin. Seuraava tutkittava päätösmuuttuja tunnistetaan dynaamisesti heti, kun kaikki rajoitteet, joihin nykyinen käsiteltävä muuttuja vaikuttaa, on muokattu (constraint propagation). Tutkittavien muuttujien järjestykseen voidaan ohjelmallisesti siis vaikuttaa.

Ilogin Optimization Suiten komponenttipohjaisuus ilmenee esimerkiksi niin, että sen ongelmanratkaisukomponentin (Ilog Solver) päälle voidaan toteuttaa myös omia operaatioanalyysiin perustuvia ratkaisumenetelmiä. Ilog Planner, Ilog Scheduler ja Ilog Dispatcher ovat valmiita toteutuksia tällaisista menetelmistä, joiden tehtävänä on ratkaista tietyntyyppisiä optimointiongelmia. Esimerkiksi Scheduler soveltuu käytettäväksi silloin, kun optimoidaan tietyn kestoisten vaiheiden (esim. työmääräykset) ajoitusta ja sijoittelua. Scheduler hyödyntää erittäin tehokasta “edge-finder”-tekniikkaa optimaalisen ratkaisun hakemisessa. Dispatcher on tarkoitettu esimerkiksi henkilöstöresurssien optimointiin ja erilaisten työskentelysäännösten huomioonottamiseen, kun taas Planner soveltuu puhtaasti lineaaristen rajoitteiden ratkaisuun kuten esimerkiksi kapasiteettisuunnitteluun. Plannerilla voidaan ratkaista mitä aktiviteetteja tietyissä etukäteen lasketuissa aikajaksoissa toteutetaan, kun lisäksi otetaan huomioon esimerkiksi asetettu budjetti, tärkeysjärjestys ja liiketoimintasäännöt. Teollisuuden pitkän tähtäimen suunnittelu, henkilökunnan aikataulut ja tietoliikenteen verkonsuunnittelu ovat tyypillisiä sovelluskohteita, joissa Solver- ja Planner-komponentteja käytetään yhdessä.

Ratkaisujen parantaminen

Useissa ongelmissa ensimmäisen ratkaisun löytäminen on helppoa ja nopeaakin, mutta optimaalisen ratkaisun löytämiseen saattaa kulua pitkän aikaa. Tämäntyyppisiin ongelmiin voidaan itse toteuttaa hakumenetelmä, joka antaa nopeasti ratkaisuehdokkaan, jota sitten asteittain parannetaan ajon kestäessä. Tällainen ajo voidaan milloin tahansa keskeyttää, jolloin saadaan tulokseksi siihen mennessä parhaaksi löydetty ratkaisu.

Strateginen ongelma-alueen tuntemus

Ilogin Optimization Suiten käyttäjä voi helposti määritellä etukäteen tiedossa olevaa ongelma-alueen tuntemustaan järjestelmälle, jotta se suoriutuisi varsinaisesta hausta helpommalla. Tämä tarkoittaa yleensä ongelman ratkaisun kannalta vaikeimman osan kiinnittämistä manuaalisesti etukäteen. Tehtaan työvuorolistaa suunniteltaessa esimerkiksi tiedetään, että tietyt ajanjaksot vuodessa ovat hankalia optimoida sen takia, että useat työntekijät haluavat pitää lomansa juuri tiettyjen juhlapyhien aikaan. Tästä syystä on usein järkevää määrätä tietyt työvuorot etukäteen manuaalisesti, jotta jäljellejäävä ajoitus onnistuisi helpommin. Muuttujien valintajärjestystä voidaan kontrolloida, jolloin monimutkaisten ongelmien ratkaisu tehostuu huomattavasti.

Yhteenveto

Rajoiteohjelmointi yhdistettynä tehokkaisiin lineaarisen optimoinnin menetelmiin on osoittautunut erittäin käyttökelpoiseksi resurssien optimointitehtävien ratkaisuissa, joissa joudutaan tekemään useita päätöstä vaativia valintoja ja joihin liittyy joukko sääntöjä ja liiketoiminnasta johtuvia erilaisia rajoitteita. Rajoitteista osa voi olla lineaarisia ja osa loogisia. Ilogin komponentit tarjoavat joustavan ja tehokkaan tavan hyödyntää uudelleenkäytettäviä komponentteja tällaisten ongelmien ratkaisuun. Suomessa Ilogin komponenttien käyttöönoton opastuksesta telekommunikaation ja teollisuuden sovelluksiin vastaa Software Engineering Center SEC Oy. Tarkempia tietoja on saatavissa osoitteista http://www.ilog.com/ ja ilog@sec.fi tai puhelimitse (09) 8045 130.

Petteri Manninen,
SEC

”Oliot ja komponentit meren armoilla”

*****

Sytyke Ry:n jäsenristeily 1999

Pekka Forselius,
STTF Oy

Lauri Laitinen,
Nokia Research Center

Mikä lienee paras tapa yhdistää hyöty ja huvi? Syyskuinen laivaseminaari kamppailee joka tapauksessa kärkisijoituksista - ainakin täällä pohjoisessa ulottuvuudessa, erään maailman kauneimman saariston äärellä. Erinomaiset ulkoiset puitteet yhdistettyinä mukavaan porukkaan takaavat onnistuneen kokonaiselämyksen. Siinä siis tärkeimmät komponentit, jotka olivat läsnä myös Sytykkeen viime syksyisellä Tukholman matkalla. Suurimmat kiitokset kuuluvat luonnollisesti ”hyville olioille” eli aktiivisille risteilijöille. Kiitosta voi antaa sekä rentohenkisestä mukana olosta että runsaasta kirjallisesta palautteesta risteilyn jälkeen. Kaikilla ei selvästikään mennyt muisti!

Jäsen1:
”Kaiken kaikkiaan matkasta jäi hyvä maku. Aivoja on kiva tuulettaa ajoittain.”

Jäsen2:
”Yleisesti ottaen risteily oli positiivinen kokemus ja kiitos järjestäjille. Kun tietää kaikkien kuitenkin olevan kiireisiä omassa työsssään, niin siitä saa vielä lisäpisteet. Mielestäni tällaisia tilaisuuksia tarvitaan, missä alan ihmiset voivat kohdata toisensa ja vaihtaa mielipiteitään vähemmän formaalisti.”

Jäsen3:
”Risteily on hyvä formaatti, koska joutuu pakostakin erilleen työstään ja voi/saa keskittyä aiheeseen. Kaiken kaikkiaan onnistunut seminaari. Odottelen jo seuraavaa risteilyä/seminaaria.”

Jäsen4:
”Kaiken kaikkiaan täytyy todeta, että seminaari oli kiva tapa poiketa hiukan arkirutiineista. Seminaarin hinta-laatu-suhde hakee vertaistaan. Seminaarin onnistumista tukeneiden näytteilleasettajayritysten nimiä olisi vallan hyvin voinut tuoda selkeämmin esiin. Suosittelen seuraavaa vastaavaa seminaaria myös kolleegoilleni!”

Jäsen5:
”Kiitoksia Sytykkeelle hyvin järjestetystä seminaarista. Paitsi, että ohjelma oli täysipainoista niin myös ruoka ja majoitustilat olivat hyvät. Lisäksi olitte tilanneet aurinkoisen sään Tukholmassa kiertelyä varten.”

Jäsen6:
”Tilaisuuden ulkoiset puitteet olivat moitteettomat, eli Silja Linekin hoiti hyvin oman osuutensa. Myös tilaisuuden ajankohta oli sopiva. Seminaari oli suhteellisen halpa, mutta sehän sopiikin yhdistyksen tilaisuuteen, jonka ensisijainen tarkoitus on palvella jäsenistöä.”

Tässä muutamia risteilykokonaisuuteen liittyviä kommentteja osallistujilta. Kaikki olivat kovin positiivisia, mikä systeemityöammattilaiset tuntien ei voi johtua ainakaan yleisestä kritiikittömyydestä. Hyvä niin! Varsinaista kritiikkiä ja moitteita ei tullut siis lainkaan, mutta osallistujajoukon luonteeseen sopivasti koko joukko erilaisia kehittämisehdotuksia seuraavien vuosien varaksi.

Systeemityö on monia tietojärjestelmien kanssa tekemisissä olevia tahoja yhdistävä osa-alue. Se on yhtä tärkeä sekä järjestelmien teettäjille että tekijöille, eikä ole niin pientä tai niin suurta organisaatiota, etteikö systeemityön osaaminen olisi tuiki tarpeellista. Suomalaisen systeemityöväen voidaan katsoa olleen kattavasti ja monipuolisesti edustettuna laivaseminaarissamme, vai mitä muuta voisi sanoa ao. osallistujayritysten luettelon perusteella. Vain muutama oli joukosta poissa.

Nokian Tutkimuskeskus, SysOpen Yhtiöt Oy, TietoEnator Oyj Valtionpalvelut, Valtiokonttori, Pohjola-yhtiöt, Oy Arbor Vitae-Finland Ltd, Oy Eläkesysteemi Ab, Pohjolan Systeemipalvelu Oy, Software Technology Transfer Finland Oy, Tietomyyrät Oy, Tilastokeskus, Taloustutkimus, Tampereen Yliopisto, Teollisuuden Voima Oy, Pohjois-Savon ammattikorkeakoulu, TietoEnator Oyj Tieto Entra Oy, Software Engineering Center SEC, Nokia Networks, Metacase Consulting, Projektihallinto Oy Proha, Vidac Oy, Suomen Posti Oy, Regex Oy, FD Finanssidata Oy, Deltrain Oy, IBM, Helia, WM-data Consulting Systeemilinna, ICL Data Oy, Eläketurvakeskus, HPY, SAS Institute, Acecon Finland Oy, Jyväskylän Yliopisto, Sampo Oyj

Kertatapahtumasta perinteeksi

Yksi pääsky ei tee kesää, eikä yksi kerta perinnettä. Ensimmäinen tähän sarjaan kuuluva Sytyke-risteily järjestettiin syyskuussa 1998, joten nyt toisena peräkkäisenä vuotena järjestettäessä tilaisuudesta muodostui perinne. Määränpäänä oli jälleen Tukholma. Osallistujamäärä kasvoi edellisen vuoden kolmestakymmenestä peräti 50 prosentilla. Mukana oli siis 45 hyvällä risteilymielellä varustettua tietojärjestelmäammattilaista, mikä tarkoittaa yli kolmea prosenttia koko jäsenkunnasta. Noin yksi kolmasosa tämänkertaisista risteilijöistä osallistui jo ensimmäiselläkin kerralla. Mielenkiinnon

Kuva 1. Tarkkaavaista, suorastaan harrasta seminaariyleisöä.

kasvaessa samaa tahtia, tarvitsemme tasaisen vauhdin taulukon mukaan koko laivan viimeistään vuonna 2008. Toisaalta, tätä vauhtia, taitaa Sytyke ry:n jäsenmääräkin kääntyä jyrkkään nousuun. Mistähän me sitten laivan teemme?

Järjestelyt

Tilaisuuden suunnittelusta ja järjestämisestä vastasi kolmimiehinen yhdistyksen johtokunnasta muodostettu työryhmä, jonka pyyteettömän työn tuloksena saatiin paitsi runsaasti yleisöä kiinnostavia esityksiä, myös yhteistä syömistä, saunomista ja sirkushuveja. Jotta asia-aiheet vastaisivat mahdollisimman monen jäsenen kysyntään, ei tämän tyyppisen tilaisuuden teemaa voi rajata liian tiukasti. Jollakin tavalla toisiaan tukevia ja toisiinsa liittyviä esitysten kuitenkin soisi olevan. Tähän järjestelytoimikunta myös oli pyrkinyt. Niinpä tällä kerralla seminaarin teemoitus oli kaksi- tai kolmejakoinen. Kaikki virallisessa seminaariohjelmassa käsitellyt aiheet pyörivät systeemityömenetelmien, niitä tukevien työvälineiden sekä koulutuksen ympärillä.

Jäsen7:
”Ohjelmakokonaisuus oli sujuva, joten jotain teemaa kannattaa seuraavaa kertaakin varten pohtia. Ehkä ensi vuonna on jokin muu teema, joka kiinnostaa eri henkilöitä. Tuskin kaikki samat jaksavat tulla uudelleen ensi vuonna.”

Molempina päivinä virallisen osuuden jämäkkänä puheenjohtajana toimi Jari Jokiniemi Nokialta. Hänen toimintaansa kiitettiin runsain mitoin osallistujien kirjallisissa palautteissa. Antti Kärjen edellisenä vuonna asettama ja Jarin nyt vakiinnuttama ”standarditaso” Sytyke-seminaarien juontotehtäville tulee olemaan merkittävä haaste seuraavien kertojen puheenjohtajille.

Jäsen5:
”Puheenjohtajana toiminut Jari hoiti tehtävänsä hyvin ja toi kommenteillaan lisäväriä esityksiin ja aikataulussa pysyttiin.”

Jäsen4:
”Puolirento meininki takasi viihtyvyyden. Jarille terveisiä hyvästä isännyydestä.”

Jäsen8:
” Puheenjohtajalle erityiskiitos siitä, että pysyttiin aikataulussa ja keskusteluille jäi aikaa.”

Järjestelytoimikunnan lisäksi kokonaisuuden onnistumiseen ja ennen kaikkea paketin edullisuuteen vaikutti esiintymistilojen yhteyteen järjestetty näyttely. Omia tuotteitaan esiin tuoneet näytteilleasettajat saivat varmasti viestiään viedyksi oikealle kohderyhmälle, ja kuulijat taas muistinsa tueksi myös kirjallista materiaalia. Hyödyllisiä muistoja siis. Oheisessa taulukossa aakkosjärjestyksessä tämänkertaiset näytteilleasettajat, joille siis suuri kiitos osallistumisesta! Näitä yrityksiä voidaan pitää suurina systeemityön ystävinä:

Kuva 2. Jari Jokiniemi harjoittamassa kiiteltyä puheenjohtajan tointaan Päivi Hietasen esityksen yhteydessä.

- ICL Data Oy
- Metacase Consulting Oy
- Projektihallinto Oy Proha
- SAS Institute Oy
- Software Engineering Center SEC Oy

Mutta sitten itse asiaan, eli mitä tarkkaan ottaen tapahtui, tunti tunnilta, tuuma tuumalta?

Seminaari aloitettiin jo lähtösatamassa kello 14, sillä osa kiireisistä asiantuntijoista ei voinut osallistua koko risteilyyn vaan vain Helsingissä järjestettyyn aloitusosuuteen. Koska väkeä oli paikalla runsaasti ja käsiteltäviä asioita paljon, käytiin tilaisuudessa ”suoraan asiaan”, kuten oheisesta kommentista voi päätellä:

Jäsen6:
”Oikeastaan esittelykierroksen puuttuminen ei haitannut, sillä väkeä oli jo niin paljon, että kukaan ei olisi esittelykierrosta jaksanut kuunnella.”

Ensimmäisen päivän osalle seminaarin kaikenkaikkiaan kymmenestä esityksestä lankesi peräti seitsemän. Periaatteina olivat siis vanhat tutut suomalaiset ”ensin työ ja sitten huvi” sekä ”älä jätä huomiseksi sitä minkä voit tehdä jo tänään”. Seuraavissa kappaleissa on referoitu lyhyesti ensimmäisen päivän jokaista esitystä, siten kuin satunnaiset matkailijat ilman määrätietoista muistiinmerkitsemistä ne sattuvat muistamaan.

UML ja sen menetelmät

Seminaari alkoi Helsingin yliopiston Tietojenkäsittelytieteen laitoksen apulaisprofessori Harri Laineen esityksellä teollisuusstandardiksi muodostuneesta Unified Modelling Language (UML) kuvauskielestä.

Laineen mukaan UML on nopeasti muodostunut kaavioiden esitystavan teollisuus- ja opetusstandardiksi. Helsingin tietojenkäsittelytieteen laitoksella UML:ää on opetettu keväästä 1997 lähtien. UML on nykyisin nähtävä nimenomaan kuvaustapana - graafisena kielenä. Vasta menetelmä määrittää mitä kaaviotyyppiä tulee käyttää missäkin vaiheessa. UML määrittelee mm. päällekkäisiä kuvastapoja. Työvälineiden tulisi pystyä näkemään ja tukemaan yksittäisten kuvausten sijasta kokonaisuuksia.

Laine kävi esityksessään lyhyesti läpi UML-peruskaaviotyypit ja niistä Helsingin yliopiston opetuksessa käytetyt suomenkieliset nimet kuten luokkakaaviot, käytöstapamalli ja tilakaaviot. Olioiden yhteistyötä palvelujen aikaan saamiseksi kuvataan palveluketjukaavioilla ja yhteistyökaavioilla sekä edellisten kanssa osittain vaihtoehtoehtoisilla pitempiaikaisia olioita kuvaavilla aktiviteettikaavioilla. Viimeksi mainituilla korvataan itse asiassa perinteiset kulkukaaviot. Mutkikkaiden olioiden kuvaamiseen on lisäksi käytettävissä komponentti- ja sijoittelukaaviot.

Perus-UML on hyvin standardisoitunut, joten nykyisin on muotia kehitellä UML:ään erilaisia laajennuksia. Esimerkiksi tietomallinnuksessa perus-UML on osittain suppeampi kuin monet muut vastaavat kuvausmenetelmät. Tämän puutteellisuuden paikkaamiseksi Helsingin yliopiston tietojenkäsittelylaitoksella on kehitelty laajennusta tukemaan relaatiotietokantojen generointia.

UML-mallinnus ja visuaalinen simulointi

Seminaarin toinen esiintyjä, Insoftin Jari Lehikoinen, jatkoi saman menetelmän tiimoilta painottuen Insoftin oman Prosa-ohjelmiston tarjoamaan menetelmän soveltamistukeen. Hän kertoi kuinka UML:n

käyttö on tuettu Prosan nykyisessä versiossa. Siinä on Lehikoisen mukaan muun muassa suunnitteluympäristöön lisätty tilakaaviomallinnuksen mahdollistama C/C++ koodin generointi ja ns. ”token”-simulointimahdollisuus. Varsinkin reaaliaikajärjestelmien suunnittelussa usein eteen tuleva tilakaavioiden paisuminen epähavainnollisiksi voidaan hallita hierarkkisilla tiloilla. Päinvastoin kuin Laine, Lehikoinen ei pitänyt UML:ää mitenkään puutteellisena relaatiokantojen kuvaamiseen ja generointiin.

Komponenttipohjaisten ohjelmistokehitysmenetelmien arviointi

Marko Forsellin Jyväskylän yliopistolta kertoi ohjelmistokehitysmenetelmien vertailututkimuksesta. Aihe on tuore ja melko vaikea, mutta joitakin selkeitä kiteytyksiä Forsell oli siitä löytänyt. Tutkimuksen mukaan esimerkiksi UML on selvästi tarkoitettu oliopohjaisiin menetelmiin, ei rakenteisiin menetelmiin. Rakenteisten menetelmien tuki vaatii Forsellin mukaan laajennuksia standardi-UML:ään.

OMT++ on käytännöllinen laajoihin interaktiivisiin sovelluksiin, koska analyysi ja suunnittelu on siinä selvästi erotettu toisistaan. Tosin joku yleisön joukosta tiesi, vedoten erääseen väitöskirjaan, että ryhdyttäessä soveltamaan OMT:tä oli jossakin toisaalla havaittu, ettei se soveltunut suunnittelujärjestelmiin.

Luokkakaavioiden käytössä on havaittavissa ero amerikkalaisen ja skandinaavisen koulukunnan vaikutusalueiden välillä. Amerikassa luokkakaaviota käytetään kuvaamaan ohjelmakoodin periyttämistä. Sen sijaan tietokeskeisen suunnittelutradition vaikutuksesta Skandinaviassa käytetään luokkakaaviota myös käsitteiden kuvaamiseen.

Forsell toi esiin myös erään vanhemman, menetelmien vertailuun kannustaneen ajatuksen. Vuonna 1994 sanottiin ohjelmointimenetelmien vain vaikeuttavan uudelleenkäyttöä, koska niiden lähtökohtana oli puhtaalta pöydältä lähteminen, ilman valmiiden komponenttien hakemisen tukea.

Tutkimuksen mukaan tarkastellut menetelmät ovat varsin heikkoja ongelmaratkaisuun. Lisäksi ongelmia analysoimalla ei olioita löydy, eli työ ei etene jouheasti vaiheesta toiseen. Menetelmille asetetut odotukset ja vaatimustaso ilmeisesti myös muuttuvat ajan kuluessa, sillä vuonna 1988 pidettiin luokkien suunnittelua helpoimpana tehtävänä, mutta nykyisin sitä pidetään kaikkein vaikeimpana.

Liiketoiminnan mallintamisesta koodin generointiin

ICL Data Oy:n Ulla Ivaska kertoi IEF:stä Composerin kautta muotoutuneesta COOL:GEN strukturoidun järjestelmän kehittämis- ja kuvausvälineestä. Toisin kuin Forsellin tutkimuksen mukaan, Ivaskalla oli 90 % ratkaisusta ongelman ymmärtämistä. COOL:GEN on melko suosittu väline etenkin vakuutusjärjestelmien kehittämisessä, mutta käyttökokemuksia on myös muilta toimialoilta. Sen avulla tuetaan järjestelmäkehitystä hyvin aikaisista suunnitteluvaiheista aina ylläpitovaiheeseen asti, kuten esitelmän otsikkokin itse asiassa lupaa.

Eräs ajatus joka tämän esityksen perusteella syntyi, on että komponenttiteknologialla voi olla osuutensa prosessointipohjaisen tietojenkäsittely-perinteen (datacom) ja infrastruktuuripohjaisen tietoliikenneperinteen (telecom) yhdistämisessä.

Systeemityökoulutus tänään ja huomenna

Tieturi Oy:n Päivi Hietanen esitteli systeemityökoulutuksen nykyisiä trendejä, ennusteita kasvualueista ja trendien toteutustavat, sekä eri pedagogisten menetelmien käyttöasteen.

Systeemityö kokonaisuutena kiinnostaa taas entistä enemmän. Vaikka yleensä sanotaankin, että koulutuksen on oltava käytännönläheistä, niin esim. workshopit eivät nykyisin toteutustapana myy. Yllättävää kyllä, perinteistenkin menetelmien koulutusta vielä tarvitaan, sillä olio-ohjelmoinnin jo koulussa oppinut nuori, voi törmätä esimerkiksi kesätöissä funktio-pohjaiseen suunnitteluun.

Erityistä mielenkiintoa ja vilkasta keskustelua herätti Hietasen nelikenttä, jossa koulutettavat oli jaettu kokemattomiin noviiseihin, kokeneisiin noviiseihin, kokemattomiin konkareihin ja kokeneisiin konkareihin. Kokemattomat noviisit ovat alalla uusia, vasta koulusta tulleita. Tämä ryhmä on tunnistettu aina. Kokeneet noviisit ovat alalla kauan olleita, perinteistä työtä tehneitä, joille uudet tekniikat ovat tulleet yllätyksenä. Heitä esiintyy etenkin koulutusta vähän tukevissa organisaatioissa, joissa ihmisten ammattitaito voidaan päästää vanhentumaan. Kokemattomat konkarit ovat ammattilaisina uusia, tietokone-harrastuksensa perusteella omaan osaamiseensa uskovia, joille systemaattiset menetelmät ja perinteiset tekniikat ovat yllätys. Viimeinen ryhmä eli kokeneet konkarit ovat ammattilaisina kokeneita, usein omatoimisesti uusiin asioihin kouluttautuneita.

Keskustelun aiheena oli kullekin ryhmälle sopivin koulutustarjonta. Keskusteltiin myös runsaasti siitä, missä määrin ja missä tilanteissa eri tyyppien edustajien yhdistäminen samaan kurssiryhmään on hyödyllistä. Joissakin tapauksissa esimerkiksi kokemattomien konkareiden ja kokeneiden noviisien yhdistäminen samalle kurssille voi synnyttää opettavaista ja mielenkiintoista vuorovaikutusta, mutta joskus tämän tyyppiset taustaerot pakottavat kouluttajan kenellekään sopimattomaan opetustahtiin.

Automatisoi omat menetelmäsi

Metacase Consulting Oy:n Juha-Pekka Tolvasen esitelmä perustui hänen vuonna 1998 hyväksyttyyn väitöskirjaansa systeemityön menetelmistä. Keskeisenä kysymyksenä oli, missä määrin CASE-työväline voi olla yleinen ja missä määrin sen on oltava sovellusspesifinen tai jopa yrityskohtaisesti muokattavissa. Sovellusspesifinen suunnitteluympäristö/menetelmä-ajattelu on jossain määrin vastakkainen UML lähestymistavalle, joka johtaa helposti ns. valmispaketteihin. Niille, jotka haluavat paneutua aiheeseen syvällisemmin, voidaan muistuttaa että sitä käsittelevä Tolvasen artikkeli julkaistiin Systeemityölehden numerossa 4/1999.

IMAP dokumenttien hallintaan

Ensimmäisen seminaaripäivän päätteeksi Software Engineering Center SEC Oy:n Jonni Ampuja esitteli IMAP-dokumenttien hallintajärjestelmää, jonka keskeinen idea on pilkkoa informaatio pieniksi uudelleenkäytettäviksi tyypitetyiksi (luokkakirjasto) paloiksi. Tällöin dokumentit voidaan rakenteellistaa, jolloin lukemiseen käytetty aika vähenee. IMAPissa kulloinkin tarvittava näkymä muodostetaan käyttötarkoituksen mukaan ja myös talletettavan tekstin määrä vähenee, koska yhteiset osat kirjoitetaan vain kerran. Kognitiivisen tutkimuksen vaatimukset ja menetelmät on yritetty huomioida dokumenttien hallintajärjestelmässä. Ampujan mukaan IMAP on sama kuin ”ihmisten välinen TCP/IP”,.

Jäsen6:
”Ensimmäisen päivän luennoitsijoita oli tarpeeksi, itse olin jo valmis iltaohjelmaa varten.”

Jäsen9:
” Tietoiskutyyppisissä luennoissa ei missään aiheessa päästä kovinkaan syvälle asioissa, mutta se tuskin oli tarkoituskaan. Lyhyet luennot ja luennoitsijoiden vaihtuminen pitivät kuulijoiden mielenkiinnon hyvin yllä.”

Jäsen6:
”Luennoitsijat olivat sisäistäneet hyvin oman tehtävänsä, eli eivät ylimainostaneet omia tuotteitaan vaan kertoivat systeemityöstä.”

Yhteensä kokonaiset viisi tuntia kestäneen asiaosuuden jälkeen hengähdettiin hetki, minkä jälkeen nautittiin yhdessä illallinen laivan seisovassa pöydässä. Sytykeläisille oli varattuna joukko omia pöytiä, eikä niissä totisesti istuttu ikävissään. Suurin ongelma taisi olla se, ettei ruoka suussa saa puhua. Mielenkiintoisia ”kollegiaalisia keskusteluja” virisi kaikissa pöydissä ja välillä oli jopa mahdollista vaihtaa pöydästä toiseen ruokalajien vaihtuessa.

Jäsen8:
”Ilallisella syntyi varmaan joka pöydässä mielenkiintoisia keskusteluja. Olisikin syytä pohtia kuinka porukka saatasiin samalla tavalla yhteen ja vaihtamaan mieleipiteitään. Yleensä jokainen hakeutuu tuttujen pariin ja uusia ulottuvuuksia ei siten helpolla löydy tai keskustelu menee pelkästään vanhojen muistelemiseksi.”

Ne, jotka jaksoivat koko ohjelman läpi, voivat vielä yökerhon show-ohjelmassa nauttia todella suurten kuuluisuuksien esiintymisistä. Esiintyjäryhmä marssitti peräperää lavalle sellaisia tähtiä kuin Tina Turnerin, Marilyn Monroen, Elvis Presleyn ja monta muuta. Eipä olisi etukäteen uskonut.

Risteilymatkustajilla on Tukholman päässä mahdollisuus rauhalliseen aamuun, jota moni käyttikin hyväkseen. Tällä kerralla ei kaupunkipäivään ollut sisällytetty mitään yhteistä ohjelmaa, joten joukot hajautuivat päivän ajaksi kukin omille tahoilleen. Päivä oli kaunis ja aurinkoinen, joten reipas kävely ja kaupungilla kierteleminen oli monien suosima vaihtoehto. Laivalle palailtiin iltapäivällä, sillä seminaariosuus jatkui paikallista aikaa kello 16. Toisen päivän aiheet liittyivät tietojärjestelmätyön ja projektien johtamiseen sekä niiden tueksi tarjolla oleviin apuvälineisiin.

Jäsen6:
”Tukholmassa pohdin mahdollista yritysvierailua. Sellaisen järjestäminen on aina haasteellista, joten tässä olisi oltava luotettavat suhteet johonkin yritykseen. Tämä ei välttämättä onnistu käytännössä. Toisaalta kaikilla ei ole mielenkiintoa tehdä yritysvierailua.”

Jäsen4:
”Seminaarille voisi muuten varata hieman nykyistä enemmän aikaa. Ainakin osa Tukholmassa vietettävästä päivästä olisi oivaa aikaa erilaisille keskustelufoorumeille. Lisäksi kun Tukholmaan mennään, niin voitaisiin selvittää, onko siellä paikallista “Sytykkeen” toimintaa ja käydä vaikkapa tapaamassa heitä.”

Jäsen6:
”Jos luennoitsijoita halutaan joskus enemmän, voisi toinen päivä alkaa klo 14.00. Esimerkiksi luennoitsija Ruotsista, paluupäivänä, oli yksi heitetty ajatus. Tosin tämäkin vaatisi jälleen suhteita, ja jokaisen luennoitsijan pitäisi tuoda jotain lisäarvoa seminaarille.”

Tietotekniikkainvestointien ja projektisalkun hallinnan välineet

Markku Hietala Projektihallinto Proha Oy:stä aloitti paluumatkan asia-aiheet kertomalla moniprojektitilanteen hallinnasta ja sen tueksi tarjolla olevista ohjelmistoapuvälineistä. Keskeinen ongelma Hietalan mukaan on oikeiden projektien käynnistäminen. Tekemisestä sinänsä ei alallamme ole puutetta, kuten kaikki tietävät. Hallittu kehittäminen vaatii paitsi etukäteen tehtävien investointilaskelmien osaamista, myös projektien onnistumisen mittaamista. Kokemuksien kerääminen kokemuspankkiin auttaa pitkällä tähtäyksellä projektisalkun entistä parempaan hallintaan ja sitä kautta tärkeimpien eli oikeiden projektien tehokkaaseen läpivientiin.

Informaation jalostaminen

Pekka Särkkinen SAS Institute Oy:stä käsitteli esityksessään muodikkaita termejä ”data warehousing” ja ”data mining”. Keskeisenä sanomana Särkkisen esityksestä erottui ajatus informaatiotietokantojen hyödyllisyyden prosessiluonteisuudesta. Sellaista asiaa kuin ”data warehouse” ei hänen mukaansa ole olemassakaan, vaan informaation jalostaminen, ”data warehousing”, on jatkuva, koskaan päättymätön prosessi.

Tiedon avulla johtaminen on vaativaa toimintaa, joka edellyttää aivan erityisten taitojen hallitsemista. Särkkisen mukaan tämän alueen ammattilaisia, ns. ”data mining-osaajia” on Suomessa toistaiseksi melko vähän. Useimmissa organisaatioissa heitä ei ole lainkaan, vaikka varmasti pitäisi olla. Heitä koulutetaan nykyisin ainakin Svenska Handelshögskolanissa ja Turun yliopistossa.

Projektipäällikön ammattitutkinto - SPC

Toisen seminaaripäivän päätteeksi Pekka Forselius Software Technology Transfer Finland Oy:stä kertoi uudesta, aiemmin tarjolla olleita vaihtoehtoja monipuolisemmasta tietojärjestelmäprojektipäälliköiden koulutusohjelmasta. Projektien johtaminen on niin vaativa taitolaji, ettei sitä perinteisillä paripäiväisillä menetelmäkursseilla tai muutamasta jaksosta koostuvilla yrityskohtaisilla ohjelmillakaan tahdo oppia riittävästi. Niiden ajoittuminen kunkin opiskelijan työn kannalta optimaalisesti ja oppien soveltaminen käytäntöön muodostuvat yleisimmin ongelmiksi. Nimenomaan näihin yritetään saada parannusta yhteensä 8 opintoviikon laajuisella, 2-3 vuoden aikana suoritettavalla Software Project Collegella eli SPC:llä.

Ensimmäiset pioneerit aloittivat SPC-opiskelun huhtikuussa 1999, joten ensimmäisiä valmistumisia on odotettavissa vuonna 2001. Suurimmat hyödyt opiskelijoiden näkökulmasta tulevat opiskelun pitkäjänteisyydestä ja määrätietoisuudesta. Kukin opiskelee oman, henkilökohtaisen suunnitelmansa mukaisesti. Jokaisella on myös henkilökohtainen opintojen ohjaaja, joka varmistaa suunnitelman järkevyyden ja auttaa tarvittaessa sopivien kurssien löytämisessä. SPC-opiskelijoiden taustaorganisaatiot hyötyvät paitsi kehittyvästä henkilöstöstä, myös ihan konkreettisesti ohjelmaan sisältyvien lopputöiden tuloksista. Forseliuksen SPC-ohjelmasta tarkemmin kertova artikkeli on julkaistu Systeemityölehdessä 3/1999.

Toinen seminaaripäivä päätettiin täsmälleen aikataulun mukaisesti kello 18, jonka jälkeen sekä naiset että miehet nauttivat suomalaiskansalliseen tyyliin saunomisesta. Osallistuminen tähänkin ohjelman osaan oli kiitettävän, jopa poikkeuksellisen aktiivista, ja monta erinomaista keskustelua virisi tässä niin inspiroivassa ympäristössä. Mitä olisikaan systeemityö ilman saunaa!

Paluumatkan illallinen oli juhlallinen ja miellyttävä, sillä a la carte –ravintolan ilmapiiri herkullisine ruokineen ja juomineen ei mihinkään muuhun voine johtaakaan. Illallisella ei enää pidetty varsinaisia puheita, mutta pöydissä keskustelu sujui hillityn iloisesti ja vapaamuotoisesti. Olihan nyt jo tutustuttu entistä paremmin. Illallisen jälkeen suurin osa vetäytyi hytin rauhaan, sillä odottihan lähes jokaista heti aamulla taas tavallinen työpäivä. Showkin olisi ollut sitäpaitsi sama kuin edellisenä iltana. Paluu arkeen tapahtui varmasti vähän väsyneenä, mutta onnellisena.

Sytyke-risteilyjen tulevaisuus näyttää vahvasti siltä, että perinne jatkuu ja vankistuu. Johtokunta on jo nimennyt seuraavan risteilytoimikunnan, varannut budjettiin sopivan summan ja paikat laivalta 5.-7. syyskuuta. Tällä kertaa odotamme 60 osallistujaa. Ilmoittautumismenettelyistä ynnä muista käytännön asioista tulee jokaiselle jäsenelle erillinen kutsu myöhemmin tänä vuonna. Pullein purjein siis tähänkin vuosituhanteen sytyttävässä seurassa!

Pekka Forselius,
STTF Oy,
teksti

Lauri Laitinen,
Nokia Research Center,
kuvat ja osa muistiinpanoista

SysOpen/rekry

1/2
4-väri

STTF

1/2
4-väri

TOIMINTAKERTOMUS VUODELTA 1999

1. Yleistä

Vuosi 1999 oli yhdistyksen kahdeskymmenes toimintavuosi ja kolmastoista täysi vuosi itsenäisenä Tietotekniikan liiton jäsenyhdistyksenä.
Vuoden 1999 alun ((31.3. tilitys) jäsentietojen mukaan yhdistykseen kuului:
(maksaneet / kaikki; suluissa vuoden 1998 vastaava luku)

- 163 / 1188 (1131) henkilöjäsentä ja
- 20 / 20 (21) yritysjäsentä.

Jäsenmäärä kasvoi vuoden aikana ollen vuoden 1999 lopussa (18.8. tilitys; suluissa vuoden 1998 vastaava luku)

- 1232 (1179) henkilöjäsentä ja
- 21 (22) yritysjäsentä.

2. Yhdistyksen kokoukset ja johtokunnan toiminta

Johtokunnan puheenjohtajana oli Silja Räisänen (TietoEnator Oyj).
Johtokunnan varsinaisia jäseniä olivat:

- Pekka Forselius (Software Technology Transfer Finland Oy)
- Päivi Hokkanen (SysOpen Oyj)
- Jari Jokiniemi (Nokia Networks)
- Hannu Kokko (SysOpen Object Team Oy)
- Antti Kärki (Cap Gemini Oy)
- Jarmo Mäkelä (Kela)

Varajäseninä olivat

- Lauri Laitinen (Nokia Research Center) ja
- Eija Hamina-Mäki (TietoEnator Oyj)

Varapuheenjohtaja oli Pekka Forselius ja sihteerinä Jarmo Mäkelä.

Johtokunta kokoontui 10 kertaa. Kokoukset olivat 14.1. (kutsuttuina myös edellisen vuoden johtokunta, liittokokousedustajat ja toimisto), 18.2. (mukana myös yhdistyksen toimistosihteeri sekä lehden toimitussihteeri), 25.3., 13.4., 5.5., 26.5., 12.8., 13.10., 19.10. (mukana myös liittokokousedustajat) ja 11.11.

Kevätkokous pidettiin 25.3.1999 Heliassa ja syyskokous 11.11.1999 Heliassa.

3. Yhdistyksen kerhotoiminta

Yhdistyksen kerhoina toimivat:

- Relaatiokantakerho eli RELA-kerho (entinen STST- kerho) liittyen relaatiokantojen suunnitteluun ja tehokkuuteen. Kerhon toiminta on ollut säännöllistä ja aktiivista.

Vetäjänä toimi Tapio Lahdenmäki (IBM); johtokunnan yhteyshenkilönä Jarmo Mäkelä.

4. Jäsentilaisuudet

20-vuotisjuhlat
20-vuotisjuhlat pidettiin torstaina 20.5. ravintola Kaisaniemessä. Juhlien suunnittelusta vastasivat Päivi Hokkanen, Hannu Kokko ja Lauri Laitinen muun johtokunnan tuella.
Juhliin oli nettobudjetoitu 20 000 markkaa, nettokuluina oli n. 21 500 mk.
20-vuotisjuhliin tulikin kiitettävä määrä osallistujia (65), vaikka kutsuista olikin paikka unohtunut mainita. Onneksi systeemityöihmiset ottavat tällaiset asiat ennen kaikkea haasteena.
Ohjelmana

- Markus Rantapuu (Järjestelmäkehityksen rappio: näkyykö valoa?!)
- Risto Linturi (Systeemityön kiehtova tulevaisuus 1999-2019)
- Showryhmä Comets vei meidän Charlestonin rytmeihin ja Tomi Parkkonen Trio loi svengaavan tanssimusiikin ja tunnelman.

20v-juhliemme kunniaksi Tietotekniikan liitto ry luovutti 6000mk:n stipendin edelleen luovutettavaksi menestyneelle opintojensa päätösvaiheessa olevalle opiskelijalle tai opiskelijoille. Stipendin saajaksi johtokunta valitsi Elina Syrjäsen (Turun kauppakorkeakoulu, Tietojärjestelmätiede).
Juhlissa muistettiin diplomilla seuraavia yhdistyksen toimintaan aktiivisesti vaikuttaneita henkilöitä: Tuija Helokunnas, Heikki Honkela, Ari Hovi, Kirsti Jalasoja, Jorma Järvinen, Eija Kalliala, Veikko Kehä, Merja Korpela, Tapio Lahdenmäki, Lauri Laitinen, Risto Nevalainen, Jorma Nikunen, Erkki Rajala, Anneli Rantanen, Raija Ristola, Silja Räisänen, Pentti Salmela, Arvo Somermeri ja Tapani Talikainen.

Syysristeily
Syksyllä (14.-16.9.) pidettiin jäsenille suunnattu seminaariristeily Tukholmaan. Risteilyn suunnittelusta vastasivat Pekka Forselius, Jari Jokiniemi ja Antti Kärki muun johtokunnan tuella.
Jäsenristeilyyn oli nettobudjetoitu 20 000 markkaa, nettokuluina oli n. 16 700mk. Osaksi matkakuluja sponsoroivat eri yritykset (ICL Data Oy, Metacase Consulting Oy, Projektihallinto Oy Proha, SAS Institute Oy, Software Engineering Center SEC Oy).
Risteilylle osallistui 45 henkilöä.
Seminaarissa luennoijat eivät saaneet erillisiä palkkioita. Luennoijina olivat:

- Harri Laine, Helsingin Yliopisto (UML ja sen menetelmäkytkennät)
- Harri Lehikoinen, INSOFT (UML-mallinnus ja visuaalinen simulointi)
- Marko Forsell, Jyväskylän yliopisto (Komponenttipohjaisten ohjelmistonkehitysmenetelmien arviointi)
- Ulla Ivaska, ICL Data Oy (Liiketoiminnan mallintamisesta koodin generointiin)
- Päivi Hietanen, Tieturi (Systeemityökoulutus tänään ja huomenna)
- Juha-Pekka Tolvanen, Metacase Consulting Oy (Automatisoi omat menetelmäsi)
- Jonni Ampuja, Software Engineering Center SEC Oy (IMAP dokumenttien hallinta)
- Markku Hietala, Projektihallinto Oy Proha (Tietotekniikkainvestointien ja projektisalkun hallinnan välineet)
- Pekka Särkkinen, SAS Institute Oy (Informaation jalostaminen)
- Pekka Forselius, STTF Oy (Projektipäällikön ammattitutkinto - SPC)

Jäsenillat
Syyskokouksen yhteydessä 11.11. pidettiin jäsentilaisuus, jossa Hannu Kokko (SysOpen Object Team Oy) johdatteli kuulijat suunnittelumallien maailmaan.
Tilaisuuteen osallistui n. 30 jäsentämme.

5. Yhdistyksen lehti

Lehti ilmestyi neljä kertaa. Lehtien teemat olivat (suluissa toimituskunta):

1. Ohjelmistotyön mittaaminen ja arviointi (Pekka Forselius)
2. Sulautetut järjestelmät (Jari Jokiniemi)
3. Henkilöstö (Pekka Forselius)
4. Data Warehouse (Jarmo Mäkelä, Relaatiokantakerho)

Lehden päätoimittajana on toiminut Lauri Laitinen. Toimitussihteerinä Susanna Koskinen (Toimistopalvelu Hennax Tmi). Johtokunta osallistui toimitussihteerin ohella ilmoitusten hankintaan.
Lehden painotaloa vaihdettiin vuoden aikana: ensimmäisen numeron painoi Offset-Koppinen ja kolme viimeistä Tuokinprint Oy.
Lehti menee jäsenille jäsenetuna. Lehteä on saanut myös tilata sekä vuosi- että irtotilauksena.
Lehden kustannuksia saatiin vuoden aikana huomattavasti pienentymään painontalon vaihdon ansiosta, myös ilmoitustuloja saatiin hieman enemmän kuin edellisenä vuonna. Nettokuluina lehteen meni n. 4 600 mk.

6. Muu toiminta

Tietotekniikan liitto
Yhdistys osallistui Tietotekniikan liiton kevät-, kesä- ja syyskokouksiin. Liittokokousedustajina olivat Raija Ristola (Eläkesysteemi), Lauri Laitinen (Nokia Research Center) ja Jari Jokiniemi (Nokia Networks). Varaliittokokousedustajina toimivat kaikki johtokunnan jäsenet.
Raija Ristola toimi TTL:n vaalitoimikunnassa ja Jari Jokiniemi TTL-lehtitoimikunnassa.
Silja Räisänen osallistui TTL:n jäsenyhdistyksien puheenjohtajien kokouksiin.

Tiedotus
Yhdistyksellä oli ilmoitus- ja tiedostussivut PC-käyttäjien palvelimella. Lauri Laitinen toimi yhdistyksen www-tiedotus vastaavana.
Toimintavuoden aikana lähetettiin kolme jäsenkirjettä.

- Toukokuussa lähetetyssä jäsenkirjeenä oli 20-vuotisjuhlien kutsu.
- Elokuussa lähetetyssä jäsenkirjeessä aiheina olivat laivaseminaarin kutsu sekä stipendin saajan haku.
- Lokakuussa lähetetyssä jäsenkirjeessä oli kutsu syyskokoukseen sekä syyskokouksen yhteydessä järjestettyyn jäsentilaisuuteen.

Lisäksi ilmoitettiin ajankohtaisista asioista yhdistyksen lehdessä ja Tietoviikon jäsenpalstalla.

7. Yhdistyksen talous

Yhdistyksen taloudenhoitajana ja toimistonhoitajana toimi Anne-Maj Viio, Puhelinvastaus- ja sihteeripalvelut VT Oy. Yhdistyksen toimisto sijaitsee Helsingissä, Takkatie 6.
Kirjanpidosta ja tilinpäätöksestä vastasi Merja Autio, Tilite Oy, Helsinki.
Yhdistyksen tilintarkastajina v. 1999 olivat Pentti Salmela (Työterveyslaitos) ja Reijo Peltola (Arthur Andersen Oy). Varatilintarkastajina olivat Jorma Nikunen (Eläkesysteemi) ja Kirsti Jalasoja (Helia).
Tilinpäätöksen tappio oli 14 067,57 mk poistojen jälkeen. Johtokunta esittää, että tilikauden tappio kirjataan kokonaisuudessaan aikaisempien vuosien voittovarat -tilille.

Yhdistys ei ole maksanut palkkaa eikä palkkioita johtokunnan jäsenille.

TAPAHTUU SYTYKEessä:

Jäsenilta 16.3. klo 14

Jäsenillan aiheet:

- XML: teoriaa ja käytännön kokemuksia (Lasse Akselin, TietoEnator Oyj, Julkinen sektori, eGov)
- Tieto Object: TietoEnatorin tapa tehdä systeemityötä olioilla ja komponenteilla (Silja Räisänen, TietoEnator Oyj, Julkinen sektori, eGov)

Paikka: TietoEnator, Tekniikantie 15, Espoo, Otaniemi

Kevätkokous 16.3. klo 17

Paikka: TietoEnator, Tekniikantie 15, Espoo, Otaniemi

Oletko kiinnostunut systeemityön koulutuksesta?

Oletko systeemityön koulutus- tai opetustehtävissä työskentelevä tai muuten vain koulutuksesta kiinnostunut tietojenkäsittelyn ammattilainen tai opiskelija?
Olemme perustamassa Systeemityön koulutus –työryhmää koulutuksen ja itsemme kehittämisfoorumiksi, ajatusten ja näköalojen vaihtopaikaksi ja miksei myös epävirallliseksi yhteistyön rakentajaksi. Työryhmän tämän vuoden konkreettisena tavoitteena on seminaarin järjestäminen alan ammattilaisille tarjottavasta koulutuksesta ja systeemityön opetuksesta eri oppilaitoksissa. Keskustelun arvoisia olisivat varmaan myös verkko-opetuksen tuomat mahdollisuudet ja muut uudet tavat koulutuksen kehittämiseksi.
Jos olet kiinnostunut työryhmässä toimimisesta tai yleensä systeemityön koulutuksen kehittämisestä, ota yhteyttä mahdollisimman nopeasti joko Maritta Korhoseen (Maritta.Korhonen@pspt.fi) tai Pekka Forseliukseen (Pekka.Forselius@Sttf.Fi).
(Tarkemmat yhteystiedot ks. http://www.pcuf.fi/sytyke tai lehden johtokunta-sivuilta).

Systeemityön tehostaminen: Jäsenkysely 20.-31.3.

Uuden vuosituhannen alkaessa pysähdymme selvittämään, mikä on systeemityön tila juuri tällä hetkellä. Uusia malleja, menetelmiä, työtapoja ja välineitä on tarjolla runsaasti. Onko asiat nyt kohdallaan, vai vieläkö jotain olisi syytä viilata... ja jos, niin mitä?
Näkemyksesi voit kertoa vastaamalla osoitteessa http://www.pcuf.fi/sytyke/kysely2000 oleviin kysymyksiin 20.-31.3. Aikaa kuluu vain muutama minuutti, mutta hyöty meille kaikille on suuri.
VAIN VASTAAMALLA VOIT VOITTAA paitsi hyvää mieltä, myös paikan SYTYKEen perinteisellä syysristeilyllä 5.-7.9. tai jonkun yllätyspalkinnoista.
Kyselyyn on kaikkien systeemityöammattilaisten vastaukset tervetulleita!

Systeemityön tehostaminen-jäsenilta 17.5.

Kyselyn tulokset läpikäydään jäsenillassa 17.5. Tilaisuudessa on mukana myös asiantuntijapuheenvuoroja ja näkemyksiä systeemityön tilasta Suomessa vuonna 2000. Lisäksi vedämme yhdessä johtopäätökset kuulemastamme ja keskustelemme, onko tarvetta ryhtyä jatkotyöhön SYTYKEen puitteissa.

Laivaseminaari

Perinteisen syysristeilymme ajankohta on varmistunut: 5.-7.9. Varaa aika jo nyt kalenteriisi!

TTL-järjestää

12.-13.4. TTL:n Ammattilaispäivät 2000.

Ks. muu tarjonta
http://www.ttlry.fi