TAMPEREEN TEKNILLINEN YLIOPISTO

PETRI NIIRANEN

VARIOITUVAN OHJAUSJÄRJESTELMÄN MODULAARISUUDEN KEHITTÄMINEN

Diplomityö

Tarkastaja: professori Asko Riitahuhta

Tarkastaja ja aihe hyväksytty Automaatio, kone- ja materiaalitekniikan tiedekuntaneuvoston kokouksessa
9. joulukuuta 2009

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Konetekniikan koulutusohjelma
NIIRANEN, PETRI: Varioituvan ohjausjärjestelmän modulaarisuuden kehittäminen
Diplomityö, 66 sivua
Toukokuu 2010
Pääaine: Konstruktiotekniikka
Tarkastaja: professori Asko Riitahuhta
Avainsanat: Toiminnanohjausjärjestelmä, modulaarisuus, varioituminen

Tuotannonohjausjärjestelmän (Enterprise Resource Planning system, ERP) avulla logistiikka-alan yritys voi hallita koko kuljetuspalvelun elinkaarta tehokkaasti. Logistiikan toimiala on kuitenkin monimuotoinen ja pirstoutunut. Siksi järjestelmätoimittajan näkökulmasta ohjausjärjestelmältä edellytetään varioituvuutta ja sitä tukevaa modulaarista rakennetta.

Tässä työssä tutkitaan niitä perusteita, joiden avulla varioituvan ohjausjärjestelmän modulaarisuutta voidaan kehittää. Tarkastelun kohteena ovat sekä sovelluksen ohjelmistotekninen että toiminnallinen rakenne. Tavoitteena on löytää ohjelmistotoimittajan mittakaavaan sopivat menetelmät, joilla varioituvien ratkaisujen tekeminen ja hallinta saadaan tehostumaan ja onnistumaan tehokkaasti.

Työ jakautuu kahteen osaan: Teoreettiseen tarkasteluosaan, jossa selvitellään modulaarisuuteen liittyviä tekijöitä esimerkkien avulla ottaen huomioon tuotteen luonteen, toiminnallisuuden ja yrityksen toiminta. Toteutuksen tarkasteluosassa tutkitaan asiakasvaatimuksia, tehtyä toteutusta ja sen apuvälineitä. Tutkimuksen lopuksi verrataan saavutettuja tuloksia esitettyyn teoriaan.

Tutkimus on tehty osana tuotteen orgaanista kehitystä, jossa pääpaino oli ensisijaisesti tuotteen toimintojen kehittämisessä. Toiminnallinen määrittely tapahtui suunnitteluryhmien kanssa. Suunnitteluryhmät muodostuivat toteutustiimin, toimittajan ja asiakkaan edustajista mahdollistaen korkean käyttäjäkeskeisen näkökulman huomioonottamisen.

Tutkimus osoittaa, että moduloinnissa voi olla useampia eri tasoja tai näkökulmia, joilla on erilaiset lähtökohdat ja haasteet. Tutkimuksen perusteella korostuu moduloitavan toimialan tuntemus oikeiden moduulirajojen löytämiseksi ja modularisoinnin iteratiivinen luonne käytännöllisten tulosten saavuttamiseksi.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Mechanical Engineering
NIIRANEN, PETRI: Developing Modularity of Resource Planning system
Master of Science Thesis, 66 pages
May 2010
Major: Machine Design
Examiner: Professor Asko Riitahuhta
Keywords: Enterprise Resource Planning, modularity, variation

A logistics company can enjoy great benefits from a tailored Enterprise Resource Planning (ERP) system by controlling the entire lifecycle of transportation service. The ecosystem is scattered and business models can vary from company to company. Thus from system vendor perspective there is a big demand in variation capabilities and architecture supporting modularity.

This study investigates modularity development of variation capabilities of ERP system. Software architecture and functional structure are all considered. The goal is to find method and model suitable to vendor scale to build and variants efficiently and successfully.

The thesis is divided into two parts. In the theoretical study part, issues related to modularity are explored. Product nature, functionality and business model are all considered. In the implementation part, ecosystem, customer requirements, implementation and supporting frameworks are discussed. Finally implementation model studied here has been examined to provide a practical dimension to theoretical approach.

The study has been made as a part of organic product development concentrating in enhancing the product functionality. In the definition phase multi-disciplinary design teams – having implementation team, vendor and customer representatives – were employed to model the business process and to take users into account.

The study indicates that there can be multiple layers or viewpoints of modularity, each having different pre-condition and challenges. The results of this study suggest that, there is a great impact in how well context under modularization is known, to find correct module boundaries. At the same time it is understood that modularization is iterative process to achieve concrete benefits.

ALKUSANAT

Tämä opinnäytetyö on tehty osana laajempaa kokonaisuutta, jossa Fleetlogis-järjestelmää kehitettiin laaja-alaiseksi logistiikan tuotannon ohjausjärjestelmäksi. Kirjoittajan osuus tässä projektissa oli toimia sovelluskehityksen projektipäällikkönä.

Haluan esittää kiitokseni kaikille niille yhteistyötahoille, joiden kanssa olin tekemisissä tämän työn kautta ja jotka väsymättä jaksoivat kertoa logistiikan ihmeellisestä maailmasta.

Työn ohjeistamisesta ja tarkastamisesta haluan lausua kiitokseni työn ohjaajalle professori Asko Riitahuhdalle.

Työn valmistuminen on ollut pitkässä tikussa, mutta vihdoin kasassa. Siksi on vielä syytä muistaa kiittää uskollisia tukijoitani: rakasta vaimoani ja reipasta poikakatrastani.

Ylöjärvi 18.5.2010

Petri Niiranen

SISÄLLYS

KÄYTETYT MERKINNÄT JA LYHENTEET

Lyhenne tai merkintä	Selite
ad hoc	Lat. juuri tähän tarpeeseen.
API	Application Programming Interface, ohjelmointirajapinta.
ADR	European Agreement concerning the international carriage of Dangerous goods by Road, sopimus vaarallisten aineiden tiekuljetuksista, akronyymi vaarallisten aineiden kuljetukselle.
CR	Contract Route, sopimusreitti.
DAO	Data Access Object.
DB	Tietokanta.
ent.	Entinen.
ERP	Enterprise Resource Planning, tuotannonohjausjärjestelmä
Fleetlogis	Sesca Logistics IT Oy:n logistiikan tuotannonohjausjärjestelmä.
IOP	Interoperability testing.
Java	Sun Microsystems Inc. kehittämä objektiorientoitunut ohjelmistokieli.
JSP	Java Server Pages.
HTML	Hypertext Markup Language.
mm.	Muun muassa.
SQL	Standardized Query Language.
TDD	Test-driven development.
UML	Unified Markup Language.
XML	Extendable Markup Language.
XSLT	Extensive Stylesheet Language Transformation.
Δ	Muutos.

1. Johdanto

Tarkastelen tässä työssä Sesca Logistics IT:n logistiikan Fleetlogis-tuotannonohjausjärjestelmän muuttamista soveltuvaksi järjestelmälliseen, suunnitelmalliseen muunteluun. Päähuomio tässä työssä kiinnitetään tuotteen muunneltavuuden kehittämiseen ja niihin asioihin, joihin tulee tai tulisi jatkossa kiinnittää enemmän huomiota. Tuotannonohjauksen yksittäiset prosessit ja niihin liittyvät yksityiskohdat jäävät tämän selonteon ulkopuolelle.

Muunneltavuuden kehittäminen liittyy järjestelmän orgaaniseen uudistumiseen. Tavoitteena oli lisätä järjestelmään uusia toiminnallisuuksia, sovellusrakenteen selkeyttäminen ja muutosjoustavuuden parantaminen. Työssä ei tavoiteltu täysin muunneltavaa tuotetta.

Logistiikan toimiala, jolle tuote on suunnattu, on hyvin monimuotoinen ja rakenteeltaan pirstalloitunut. Yhtenäisen kokonaisuuden muodostaminen on erittäin hankalaa johtuen yrityksiin vakiintuneista useista erilaisista työtavoista ja käytännöistä. Merkittävä osa tuotantotiedon käsittelystä on käsityövaltaisia ammattitaitoa ja tarkkuutta vaativia tarkastustöitä.

Fleetlogisilla hallitaan koko kuljetuspalveluun liittyvää toimintaketjua: kuljetussopimuksia, tilauksia, tuotannon suunnittelua, tuotantoa ja siihen liittyvää jälkikäsittelyä sekä muodostetaan aineistot erilaisia ulkopuolisia järjestelmiä varten.

Aloitan tarkastelun käsittelemällä muunneltavuuden teoriaa luvuissa 2 ja 3. Luku 4 keskittyy toimialan erityisvaatimuksiin ja niiden hallintaan. Viides luku tarkastelee tuotteistamisen lähtökohtia ja kilpailutilannetta. Kuudes luku valottaa valittuja työvälineitä ja teknologioita. Seitsemännessä tarkastellaan case-tyylisesti tehtyjä toteutuksia. Kahdeksannessa luvussa käydään läpi tärkeimmät havainnot ja yhdeksäs avaa näkökulmia tulevaisuuden kehitysmahdollisuuksille.

2. Räätälöityvä modulaarinen tuote

Vaihtelevien asiakasvaatimusten kattaminen hallitusti ja nopeasti edellyttää tuotteelta kameleonttimaista muuntautumiskykyä. Tuotetarjonnalta edellytetään sekä laajuutta että joustavuutta erilaisille vaihtoehdoille ja muutoksille. Kun kehitys keskittyy yksilöivään – *ad hoc* – parantamiseen, kasvaa valikoima usein hallitsemattomasti, missä tehokkaan toiminnan esteiksi nousevat sekä suunnittelun että tuotannon monet toiminnot ja variaatioiden suuri lukumäärä.

Tunnusomaisia moduloinnilla ratkaistavia ongelmia ovat tilanteet, joissa valmistettavat tuotteet ovat samankaltaisia, mutta joihin sisältyy kuitenkin merkittävä määrä lisäsuunnittelua asiakasratkaisun saavuttamiseksi. Vakiintuneesta perusratkaisusta huolimatta tuotteen toimittaminen muistuttaa enemmän kertaluonteista prosessia kuin tuoteohjelman mukaista tuotetta.

Aidosti asiakasräätälöintiä tukeva tuote on jo lähtökohdiltaan laadittu muunteluun soveltuvaksi. Sen ominaisuuksia voidaan säädellä valittavien toiminto-osuuksien mukaisesti ja jopa asettaa tuotantoon vasta tilausperusteisesti. Tarjolla olevista ominaisuuksista on mahdollista tuottaa kerta toisensa jälkeen erilainen, mutta kuitenkin ennalta suunniteltu tuote.

2.1. Mikä on muunneltava modulaarinen tuote?

Mikä tahansa kokonaisuus, joka voi muodostua tuotteesta ja siihen mahdollisesti liitetyistä palveluista, on tässä yhteydessä kelvollinen modulaariseksi tuotteeksi. Oleellinen osa modulaarisuuden hallintaa on modulointi, joka on väline säännönmukaiseen kokonaisuuden jäsentämiseen ja hallintaan. Perusajatuksena on muodostaa pienempiä kokonaisuuksia, jota voidaan käsitellä, suunnitella ja ohjata erillisinä. Näiden alikokonaisuuksien eli moduulien välillä vallitsee tietyt yhteneväisyydet ja lainalaisuudet sekä ne voivat olla korvattavissa jollakin toisella samat ehdot täyttävällä moduulilla. Tavoite on siis päinvastainen kuin integraatiossa, jossa kokonaisuutta ei voida ajatella pienemmistä kokonaisuuksista muodostuvaksi Pahl ja Beitzin (1990, s. 24) mukaan.

Perinteisesti modulaarisuus on perustunut ensisijaisesti moduulijärjestelmiin, joissa on ensisijaisesti yksittäisen yrityksen, yleensä tuotantoon tai valmistukseen liittyvää, ongelmaa ratkaistu. Modulaarisuutta on käytetty välineenä yrityksen sisäisen toiminnan parantamiseen ja välineenä tehdä massaräätälöintiä. Huomattavasti harvemmin modulaarisuus on merkitsevä ominaisuus tuotteen asiakkaalle. Kirjallisuudessa esitetyistä rat-

kaisuista Lehtonen (2007) on tutkimukseensa kerännyt katsauksen moduloinnin historiallisista vaiheista ja teorioiden kehityksestä.

Samassa tutkimuksessa Lehtonen toteaa, että modulaarisuus terminä että teoriana on ymmärretty hyvin laajasti. Esimerkiksi modulaarisuus yleensä mielletään samaksi kuin moduulijärjestelmä. Eli konkreettisten osien, moduulien tai valmistusvaiheen vaihtokelpoisuudeksi. Tämän tulkinnan esittävät esimerkiksi Pahl ja Beitz. Heidän näkemyksensä perustuu konstruktio-opillisiin näkökohtiin ja on asiayhteydessään perusteltu. (Pahl ja Beitz, 1990, s. 436–439.)

Siksi Lehtonen on käsitteistön selkeyttämiseksi määritellyt muunteluun tähtäävään modulaarisuuden M(=*muuntelu*)-modulaarisuudeksi ja muuntelussa käytettävät moduulit M-moduuleiksi (2007, s. 88). *M*:n puitteissa hän antaa määritelmän moduulille ja moduulijärjestelmälle:

- Kappale (mikä tahansa kokoonpano tai järjestelmän osa) on moduuli, jos se toteuttaa määritellyn rajapinnan ja se kuuluu moduulijärjestelmään.
- Modulaarinen järjestelmä on järjestelmä, joka muodostuu vaihtokelpoisuuden toteuttavista kappaleista.

Tämän työn puitteissa esitettävät viittaukset modulaarisuuteen ja moduuleihin ovat Lehtosen käsitteistön mukaisia.

2.1.1. Modulaarisuus lähtökohtana

Moduulit ja moduulijärjestelmät ovat välineitä modulaarisuuteen. Mutta vain osana, koska modulaarisuudella haetaan ongelmanratkaisuun laajempaa ajatusmallia. Tässä yhteydessä usein viitattuja lähteitä ovat Alexanderin (1977) sekä Gamman ja muiden (1999) design pattern -mallia käsittelevät teokset, joiden pääteemana on ongelman ratkaisu aina samanlaisen peruskaavan mukaan, kuitenkin niin että saavutettu lopputulos on joka kerta erilainen. Osa ajatusmallin laajentamista on myös tietyn epävarmuuden lisääminen, jota Ulrich ja Tung (1991) esittävät. Heidän mukaansa modulaarisuus voi olla suhteellista, siis tuote voi olla enemmän tai vähemmän modulaarinen.

Lehtonen (2003) mainitsee Ulrichin ja Tungin määritelleen komponenttien riippumattomuuden oleelliseksi osaksi modulaarisuutta, sillä riippumattomuus on standardoinnin ja vaihtokelpoisuuden peruste. Vaihtokelpoisuuden edellytys on, että komponenteilla on yhteinen määritelty tapa liittyä toisiinsa, siis rajapinta, jota ilmennetään yleensä standardien avulla. Riippumattomuus on vaikeammin määriteltävä laadullinen ominaisuus, joka kuvaa komponentin itsenäisyyttä, Sitä voi parhaiten hahmottaa kysymällä, voiko osan korvata toisella muuttamatta mitään muuta (liityntää). Mitä alhaisempi on ulkopuolisten muutosten määrä sitä itsenäisempi komponentti. Optimaalisessa tilanteessa muutoksia ei tarvita.

Modulaarisuus ei välttämättä rajoitu yksittäiseen tuotteeseen vaan yleistä ratkaisua voidaan käyttää useampaan eri sovellukseen. Alusta, platform, voi olla perusta tuote-

perheelle, jonka jäsenillä voi olla hyvin erilaiset käyttötarkoitukset (Juuti ja Lehtonen 2004, s. 104-113). Näitä toteutuksia voi havaita hyvinkin arkipäiväisissä tuotteissa: esimerkiksi Hölttä (2004, s. 318) mainitsee samaan akkuun perustuvat työkoneet, ja Juuti ja Lehtonen (2004, s. 104) samaan piirikorttiin perustuvat matkapuhelimet. Näissä variointi tapahtuu joko ulkoisen rakenteen (porakone, pyörösaha) tai käyttötarkoitukseen (käyttöjärjestelmän: perus- tai monitoimipuhelin) mukaan. Laajemmin platformeja käsitellään luvussa 3.

Modulaarisuus keskittyy helposti tuotteen komponentteihin ja fyysisiin ominaisuuksiin, mutta myös tuotteen ulkopuolisia elementtejä, kuten työtapoja ja -menetelmiä sekä näiden kehittämistä, voidaan käsitellä osana modulaarisuutta, varsinkin, jos modulaarisuus mielletään koko yrityksen kattavaksi toimintatavaksi. Lehtonen (2007, s. 206) tarkastelee väitöskirjassaan modulaarisuuden lähtökohtia ja pitää mahdollisena, että fyysisiin rajauksiin perustuvalla moduulijärjestelmällä ei välttämättä saavuteta kaikkia hyviä ominaisuuksia, joita modulaarisuuteen usein liitetään.

2.1.2. Muunneltavuus välineenä

Muunneltavuuden tai muuntelun ja modulaarisuuden tärkein ero on mielestäni siinä, että modulaarisuus on muuntumisen ideologiaa ja muunneltavuus on sen ilmaisua, jonka voisi yksinkertaistaen tulkita myös näin: modulaarisuus mahdollistaa muuntelun, jota myös Sarinko (1999, s. 33) toistaa. Tässä on syytä korostaa, että muuntelu viittaa nimenomaan hallittuun muunteluun, jossa vaihtoehtoisia ratkaisua luodaan käyttämällä ennalta suunniteltuja toimintoja ja tunnettuja reunaehtoja.

Muuntelun lähtökohta valitaan tilanteen ja käyttötarkoituksen mukaan. Siksi sen luonne voi olla merkittävästi erilainen eri tilanteissa. Ulrichin ja Eppingerin (1995) mukaan yrityksen sisäinen muuntelu saattaa merkitä esimerkiksi valmistuksen ja jakelun toimintoja ja vastaavasti asiakkaalle se voi tarkoittaa vaihtoehtoisia toimintoja.

Ongelmat, joiden välistä kuilua muuntelulla yritetään kaventaa liittyvät projektoivan asiakaslähtöisen suunnittelun vähäiseen toistettavuuteen ja massatuotteen muutosjäykkyyteen. Projektoiva asiakaslähtöinen toiminta antaa rajattoman joukon vaihtoehtoja käytettäväksi asiakasvaatimusten täyttämiseksi, mutta toistuvien suoritusten määrä jää vähäiseksi ja määrästä saatava hyöty saavuttamatta. Vastakohtaisesti massatuote ei anna mahdollisuuksia muuntelulle, mutta se on edullinen kustannuksiltaan.

Yksittäisen tuotteen kohdalla muunneltavuus on ratkaisu, keino rationalisoida tuotantoa. Muunneltavuudella luodaan tuotteelle lisäarvoa täsmentämällä sen ominaisuuksia vastaamaan kulloinkin kyseessä olevaa tilannetta. Sarinko (1999, s. 31) tunnistaa, viitaten Tiihosen tutkimukseen, että yritykselle lisäarvoa tuo muun muassa parantuneet mahdollisuudet reagoida asiakkaiden vaatimuksiin, lyhentyvät toimitusajat ja sitä kautta pienemmät varastot sekä tuotannon parempi hallittavuus. Asiakkaalle lisäarvoa voi olla esimerkiksi valikoidut toiminnot ja ominaisuudet tai vähentynyt toimitussuunnittelu.

Asiakkaalle modulaarisuus ei ole lisäarvo, joissakin tilanteissa se voidaan nähdä jopa tuotteessa negatiivisena ominaisuutena Lehtonen (2007, s. 207) esittää.

Kun yritys soveltaa muunneltavuutta läpi organisaation ja toiminnan, tulee siitä toimintatapa. Saringon (1999) mukaan motivaatiot siirtyä massaräätälöintiin voivat olla moninaisia: pitkän aikavälin hallittu muutos, nopea strategia muutos tai uuden liiketoiminta-alueen luominen. Oleellista on, että yrityksen henkilöstö ymmärtää muutoksen tuomat edut. Pitkän aikavälin muutoksessa toimintatapoja voidaan muuttaa pienissä erissä, niin että muutosvastarinta pysyy vähäisenä. Nopean muutoksen takana on yleensä yrityksen toimintaedellytysten tai kilpailutilanteen vaikeus, missä henkilöstönkin on helpompi hyväksyä muutos. Uuden liiketoiminta-alueen kanssa voidaan uudet toimintatavat luoda heti suunnitteluvaiheessa ja välttää aikaisempien tapojen tai organisaation tuomia ongelmia. (Sarinko 1999, s. 15–16.)

Muunneltavuus, mutta myös modulaarisuus, voi olla myös rajoite, se saattaa olla kompromissi erilaisten painotusten välillä. On todennäköistä, että asiakasmuunneltu tuote on kalliimpi kuin vastaava massatuotettu, ja että moduloinnin myötä tuoteidentiteetti saattaa kärsiä joko tuotteiden liiallisesta samankaltaisuudesta tai arkkitehtuurin tuomasta muutosjäykkyydestä Lehtonen (2003) muistuttaa.

Sarinko (1999, s. 21) kuvaa liian samankaltaisten tuotteiden ongelman sellaiseksi, missä asiakkaiden näkökulmasta tuotteet eivät erotu toisistaan ja valinnanmahdollisuudesta ei ole mitään hyötyä itselleen. Nielsen edellyttää, että vaihtoehtojen välillä pitää olla vaihtoehtoisuutta, että ne ovat keskenään elinkelpoisia (Nielsen 2009, s. 23). Tuoteohjelmalta saatetaan edellyttää keskenään ristiriitaisia vaatimuksia kattavuuden ja tarjonnan suhteen.

Toisaalta, poikkeava tuote saattaa jopa järkyttää asiakaskuntaa. Lehtonen (2007, s. 121–122) esittelee tapauksia (mm. Büssing decklaster kuorma-autokonsepti 1960-luvulta ja Hollmingin itsekantavaan säiliöön perustuva säiliöauto 1980-luvulta), joissa asiakasten odotuksiin ei ole vastattu. Monet näistä tuotteista ovat olleet teknisesti lähes ylivertaisia, mutta niiden valtavirrasta poikkeava ulkonäkö ei ole saanut asiakaskunnan hyväksyntää.

Modulointi voi olla myös riski tai siihen voi sisältyä muita riskejä. Lehtosen (2007) näkemyksen mukaan modulaarisuus voi olla myös muuta kuin tuotteen fyysisiin ominaisuuksiin tai toimintoihin perustuva moduulijärjestelmä. Lehtosen kokemukset muista lähestymistavoista ovat rohkaisevia ja mahdollistavat moduulijärjestelmään liittyvien haittojen, kuten suorituskyvyn tai tehottoman tilankäytön, vähentämisen.

Uusi teknologia on aina riski. Sen mukana saattaa olla epävarmuutta, joka voi olla hankkeelle ja tai yritykselle kohtalokas. Esimerkkinä tällaisesta hankkeesta Lehtonen esittelee Valmetin (nykyisin Transtech) M-sarjan diesel-veturin kehityksen (Lehtonen 2007, s. 125–130).

Muunneltavuuden haittoja voidaan varmasti vähentää, mutta millä keinoin? Yksi keino on sitoutua toimintatapaan pitkäjänteisesti, sillä Lehtosen (2003) mukaan tuotteeseen kohdistuva lisäsuunnittelun määrä voi olla suuri ja sitä kautta järjestelmän perustamiskustannukset voivat nousta korkeiksi. Sarinko (1999, s. 32) mainitsee keinoksi perusteellisen selvityksen asiakaskunnan vaatimuksista ja minkä ominaisuuksien suhteen muunnellaan. Peltosen ja muiden (2002, s. 79–80) keino on lähteä tavoitteesta, että massa- ja erillistuotannon etuja yhdistämällä löydetään sekä toistettavuutta että joustavuutta eli niitä keinoja, joilla tuotteelle kohdistuvat kiinteät kustannukset jäisivät mahdollisimman alhaisiksi.

Muunneltavalla tuotteella pitää olla vielä elinikää jäljellä, jotta siihen sijoitettu panos maksaisi itsensä takaisin, koska kustannukset sijoittuvat alkuvaiheen suunnitteluun. Useat lähteet tunnistavat sen tosiasian, että suurin hyöty muuntelulla saadaan, kun tuotteella on riittävä toiminnallinen ja valmistustekninen kypsyys, eli tuotteesta on aiempaa kokemusta kuten aikaisempia versioita (Sarinko 1999, s. 39, Lehtonen 2003, Rask 2004a, Nielsen 2009, Lehtonen 2007).

Kannattavuuden rajat asettuvat tuotteen ja volyymin mukaan, mutta Saringon (1999, s. 51) nyrkkisäännön mukaan muunneltavia tuotteita tulee vuodessa olla muutamista kymmenistä muutamiin satoihin, koska vähämenekkisiin tuotteisiin kohdistuu liian suuret kiinteät kustannukset ja toisaalta suurimenekkisellä massatuotteella ei asiakkaalle saavuteta lisäarvoa.

Tuotteelta edellytetään soveltuvuutta muunteluun, joka tarkoittaa, että ominaisuuksien ja komponenttien välisien sidoksien pitää olla määrällisesti rajallisia. Suuri keskinäisten riippuvuuksien määrä vaikeuttaa riippumattomien kokonaisuuksien muodostamista, jota Lehtonen (2003) pitää moduloinnin edellytyksenä.

2.2. Hallitusti suunniteltu hallittuun muunteluun

Hallittu muuntelu konkretisoituu tuoteyksilöksi eli variantiksi, kun asiakasvaatimukset konfiguroidaan tuotemalliin. Tätä yksilöintiprosessia Sarinko (1999, s. 24) toteaa kutsuttavan konfiguroinniksi ja tuoteyksilön määrittelyä konfiguraatioksi. Konfiguroinnin oleellinen elementti on, ettei siinä enää suunnitella uutta, vaan valitaan olemassa olevasta valikoimasta haluttu vaihtoehto.

Tavoitteista riippuen konfigurointi ei aina tuota täydellistä tuotemäärittelyä vaan esitietoja seuraavaa vaihetta varten. Lehtonen (2003) toteaa, että konfigurointia voidaan suorittaa usealla eri tasolla ja vaiheessa, jolloin aikaisempien konfiguraatiovaiheiden tiedot ovat seuraavan lähtötietoja. Näin portaittain edeten konfigurointia voidaan käyttää sekä ulkoiseen että sisäiseen ohjaukseen. Ulkoinen ohjaus saadaan esimerkiksi keräämällä myyntikonfiguroinnilla asiakasvaatimukset ja toivotut ominaisuudet, jolloin tuotannon valinnoilla ei ole eikä saa olla merkitystä. Sisäinen ohjaus vuorollaan tarkentaa

ne tuotantomenetelmät ja -tavat, jotka sopivat parhaiten valittuun (myynti)konfiguraatioon.

Koska konfigurointi vaikuttaa yrityksen toimintamalleihin, valittavaan tuoterakenteeseen ja käytettävään konfigurointimalliin, pitävät Pulkkinen ja Bonguelielmi onnistuneen käytännön toteutuksen edellytyksenä näiden välistä tasapainoa ja huomiointia (Pulkkinen & Bonguelielmi 2004). Tuote ei ole automaattisesti konfiguroituva, vaan se vaatii hyvin systemaattisen ja analyyttisen tuoteperheen kehityksen, jossa konfigurointi on mukana suunnittelussa heti alkuvaiheista lähtien.

Schomburgin (1980) jaottelun mukaan tuotteet voidaan jakaa neljään eri ryhmään tuoteattribuuttiensa mukaan. Pulkkisen ja Bonguelielmi (2004, s. 361) mukaan hallittu muuntelu sopii sellaisille tuotteille, joilla on Schomburgin jaottelussa joko yrityksen tai asiakkaan määrittelemät variantit. Tällaisia ovat esimerkiksi kannettava tietokone, johon voidaan valita vain valmistajan komponentteja tai työstökone, johon vakio-osien lisäksi voidaan suunnitella myös muita komponentteja. Kiinteät massatuotettavat vakiotuotteet (esimerkiksi kodinkoneet) tai kertaluonteiset projektit (voimalaitostoimitus) eivät ole konfiguroinnin kannalta mielekkäitä muunneltavia, koska muunneltavuus ei tuo niihin lisäarvoa.

Muunneltavan tuoteperheen suunnittelussa Pulkkinen ja Bongulielmi (2004, s. 362–363) lähtevät liikkeelle prosessien taloudellisten vaatimusten määrityksestä ja niihin vaatimuksiin sopeutuvan muuntelusystematiikan ja mallinnusmenetelmän valinnasta. Rask (2004b) antaa helppotajuisemman ja käytännönläheisemmän ohjenuoran tuotesuunnittelulle, Scanialla pitkään käytetyn työkalupakin – toolboxin – johtoajatuksen: pääkomponenteilla tulee olla vakioidut rajapinnat, huolellisesti mitoitetut suorituskykyalueet eri mallien välillä tyydyttämään laajan käyttöalueen tarpeet ja samat ratkaisut samoihin vaatimuksiin.

Tuotteen moduulien määrittämiseen ei ole yhtä oikeata ratkaisua vaan modulointiperuste on subjektiivinen valitulle tuotteelle ja tavoitteille. Hölttä (2004, s. 381–389) on vertaillut kirjallisuudessa esiteltyjä moduulien määritystapoja ja niiden soveltuvuutta tuoteperheille. Tavoille on ominaista, että ne ohjaavat määritystä omien lähtökohtiensa mukaisesti: ehdotetut moduulit ovat menetelmästä riippuen osin yhteneviä, toisin sanoen ne tunnistavat samoja elementtejä, mutta rajaukset ovat erilaisia. Näitä varmasti voidaan käyttää apuvälineenä, kun tuotteelle suunnitellaan moduulirakennetta, mutta viime kädessä modulointiratkaisut pitää ratkaista valitsevan tilanteen tavoitteiden, tiedon ja kokemuksen mukaan. Tätä johtopäätöstä tukee Höltän havainto, ettei mikään vertailluista menetelmistä sovellu sellaisenaan tuoteperheen modularisointiin. (Hölttä 2004, s. 387–388.)

2.3. Voidaanko tuote moduloida?

Modulaarisuuden sovittaminen tuotteeseen voi olla hyvin monisyistä ja toimintatapa soveltuu joillekin toisia paremmin. Siksi yrityksen on tärkeätä arvioida tilannetta omasta näkökulmastaan ja tilanteestaan kokonaisvaltaisesti, koska on harhaanjohtavan helppoa tehdä virheoletus, että tuotteiden näennäinen yhdenmukaisuus on riittävä peruste moduloinnille ja modulaarisuudelle. Seuraavaksi tarkastelen kahden eri yrityksen valintoja ja näkökulmia sekä omaan tuotantoon että tuotteeseen. Näistä vain toinen on päätynyt moduulijärjestelmäpohjaiseen sarjatuotantoon ja toinen pitäytynyt erillistoimituksissa.

Volvo Bus Finland Oy (entinen Carrus Oy) ilmoittaa toimittavansa vuositasolla noin 450 – 500 eritasoisesti varusteltua linja-autoa lähi-, kauko- ja matkailuliikenteeseen (Volvo Bus Finland 2002). Tervolan (2004) mukaan lähiliikenteeseen alustavaihtoehtoja on kuusi erilaista, joissa osassa voidaan tehdä valintoja kaksi- tai kolmeakselisen sekä jäykän tai erillisjousitetun etuakseliston väliltä. Moottorivaihtoehtojen lisäksi muuttujia ovat moottorin sijoitus keskelle tai taakse ja lisäksi takamoottorisista on mahdollista valita vielä matalalattiaversio. Turistibusseissa alustavaihtoehtoja on kolme ja kaikki alustat on varustettu levyjarruilla ja ilmajousituksella. (Tervola 2004, s. 11.)

Scania AB valmistaa raskaan liikenteen kalustoa, joista Eurooppaan suuntautuvan kuormaautokaluston määrä ensirekisteröinteinä on lähes 50 000 kappaletta (Scania 2005). Ajoneuvovalikoima kattaa kauko- ja jakeluliikenteen, maansiirtoautot sekä erikoisajoneuvot. Ajoneuvoihin on saatavissa kolmea eri ohjaamotyyppiä, joissa viisi erilaista makuuohjaamoa, kolme erilaista päiväohjaamoa ja kaksi lyhyttä ohjaamoa. (ScanAuto 2002.)

Tässä yhteydessä tarkastelen molempien ajoneuvojen korirakenteen räätälöintiä valmistuksen, suunnittelun ja toimintatavan näkökulmasta. Valmistajilla on perusratkaisut, joita muunnellaan asiakkaan toivomusten mukaisesti: linja-autossa muunneltava kori kattaa koko ajoneuvon ja kuorma-autossa ohjaamon. Siksi myös ratkaisumallit eroavat merkittävästi toisistaan: Volvon muuntelu on erillisratkaisuja ja Scanian muuntelu on massaräätälöintiä. Seuraavien lukujen, 2.3.1 ja 2.3.2, tarkastelut toistuvuuden merkityksestä ja suurien tuotantomäärien muuntelusta kertovat niistä näkemyksistä, joista näihin ratkaisuihin on päädytty.

2.3.1. Toistettavuuden merkitys

Toistettavuus nousee avainasemaan, kun tarkastelen Volvon tapausta. Siksi moduulijärjestelmä ei ole vaihtoehto. Volvon arvio alhaisesta toistettavuudesta on perusteltu, kun jo pelkkiä alustaratkaisuja ilman muita variaatioita on lähes sata ja yksinkertaisella laskutoimituksella voi todeta samankaltaisten toimitusten jäävän yhden käden sormilla laskettavaan lukumäärään. Kuin jatkoksi Tervola (2004) lisää, että merkittävä osuus muuntelusta on asiakkaan määräämiä valintoja ja variantteja varustelutason, säädösten, kulkureittien ja istuinten sijoittelun sekä matkustajapaikkojen sovittamista. Kun vielä koke-

muksesta tiedetään, että yksittäinen asiakaskin tilaa harvoin kahta täsmälleen samanlaista ajoneuvoa ei toistuvuus ole tuotannossa merkittävä tekijä. (Tervola 2004, s. 11.)

Tervola (2004) toteaa erillisratkaisujen merkityksestä Volvon liiketoiminnassa, että se on tärkeä osa heidän toimintatapaansa. Myyntiorganisaatio on vakuuttunut, että asiakkaan toivomusten mukainen joustava räätälöinti, on tärkeä kilpailuvaltti. Toisaalta erillisratkaisu on myös seuraus, sillä nykyisen mallin vakiointia suuresti hankaloittava asia on sen kantava korirakenne, jossa jokainen muutos ovien lukumäärään, sijaintiin tai korin pituuteen muuttaa korikehikon rakennetta. (Tervola 2004, s. 12 - 13.)

Tervola (2004, s. 11 - 15) näkee, ettei linja-auto ole moduloitavissa. Näkemys pitää paikkansa, jos tarkastelun kohteena on moduulijärjestelmä. Tämä perustuu siihen, että valmiin linja-auton integroidusta rakenteesta ei löydy erillisiä moduuleja. Tässä voin hyvin yhtyä Lehtosen (2007, s. 206) näkemykseen, että toimintoperusteinen modulointi tai moduulijärjestelmä ei vastaa reaalimaailman tavoitteita. Tarkastelemalla toisin, esimerkiksi noudattaen Lehtosen (2007) näkemyksiä modulaarisesta järjestelmästä, on hyvin todennäköistä, että toiminnasta voisi helposti osoittaa piirteitä vähintään osittain modulaarisista elementeistä, esimerkiksi suunnitteluperiaatteiden uudelleen käytöstä (Baldwin ja Clark 2000). Niiden tarkastelu jätetään tämän ulkopuolelle.

2.3.2. Harkittu modulaarisuus

Scanialla päinvastoin kuin Volvon tapauksessa tuotantomäärät ovat suuret (Scania 2005), ja yksilölliset asiakastoiveet ovat käytännössä mahdottomia. Siksi Scania toteuttaa massaräätälöintiä tarjoamalla sarjatuotteeseen rinnakkaisia, mutta tarkasti rajattuja valintavaihtoehtoja, joista asiakas voi valita mieleisensä.

Huolellisella suunnittelulla on massatuotteesta tehty tehokkaasti muunneltava, varioituva tuote. Tämä on edellyttänyt, että modulaarisuuden vaatimukset on tunnistettu heti suunnittelun alkuvaiheessa ja moduulijärjestelmää on määrätietoisesti ja pitkäjänteisesti kehitetty muunteluun sopivaksi. Ajoneuvo on toisaalta kiitollinen moduloitava, koska tietty yhdennäköisyys sekä edeltäjien että rinnakkaismallien välillä on tärkeä identiteettitekijä, ja siksi tuotteiden samankaltaistuminen ei ole ongelma. Muunneltava tai modulaarinen tuote ei kuitenkaan tarkoita, ettei se voisi olla ulkonäöllisesti houkutteleva, sillä muotoilukin voi olla moduuli, kuten Sarinko (1999, s. 40) esittää. Rask (2004a) jatkaa, että Scanian pääsarjojen muotoja on jo pitkään haettu huippumuotoilijoiden avustuksella.

Toisaalta lähtökohdat ovat myös hyvin erilaiset kuin linja-autovalmistuksessa, koska Scanian tuotantomäärät ovat merkittävästi suuremmat ja hytin rakenteellinen itsenäisyys antaa paremman lähtökohdan modularisoinnille. Käytännössä ratkaisu on massatuotannon muuntamista massaräätälöinniksi, jossa variointi tapahtuu valmistajan määrittämillä komponenteilla. Lisäksi kuorma-autoissa ohjaamo on perinteisesti muusta rakenteesta erillinen, jolloin rakenneratkaisujen merkitys on paljon vähäisempi kuin linja-auton korissa.

Scaniallakaan tavoitteena on ollut kilpailuedun hakeminen oman toiminnan tehostamisella. Moduloinnin vaikutukset näkyvät esimerkiksi nimikkeistön supistumisena, joka parantaa suurtuotannon edellytyksiä ja edelleen vähentää tuotannon ohjausta. Pitkäjänteisessäkin kehitystyössä voidaan saavuttaa merkittäviä tuloksia, joista Lehtonen (2003) mainitsee esimerkkinä Scanian hyttiin käytettävien nimikkeiden lukumäärän vähentymisen lähes puoleen 3- ja 4-sarjojen välissä. Tätä tulosta voidaan pitää yllättävänä, koska modulointia on Scanialla tehty jo vuosikymmenien ajan: Raskin (2004a) mukaan modulointi on aloitettu Scanialla jo 1970-luvun puolivälissä.

2.3.3. Muuntelu toimintatapana

Siinä missä modulointi tuottaa vaihtokelpoisia tuotepalikoita, on modulaarisuus muuntelun ideologia ja muuntelu väline toteuttaa sitä. Muuntelu on liiketoimintaidea ja toimintatapa, missä voidaan vastata asiakaskunnan alati kasvaviin vaatimuksiin hallitusti, joustavasti ja nopeasti. Hallitun muuntelun avulla voidaan tarjolla olevista ominaisuuksista kerta toisensa jälkeen tuottaa erilainen, mutta kuitenkin ennalta suunniteltu tuote.

Hallitun muuntelun perusedellytys on, että toistettavuus on etu suunnittelussa ja tuotannossa. Tällöin suunnittelu voi tukeutua toimivaksi tunnettuun ratkaisuun, asiakasmuuntelu voidaan toteuttaa ilman lisäsuunnittelua ja tuotanto hyödyntää suurempia sarjoja. Muunneltaville tuotteille on ominaista, että ne perustuvat modulaariseen rakenteeseen ja muuntelu tapahtuu ennalta määritellyissä rajoissa.

Parhaimmin muunteluun sopivat sellaiset tuotteet, jotka ovat joko valmistajan tai asiakkaan määritysten mukaan varioituvia. Toimivan kokonaisuuden kannalta on tärkeätä, että tuote on muunteluun rakenteellisesti sopiva ja että samankaltaisten tuotetoimitusten määrä on tuotantoon nähden sopiva: ei liian pieni eikä liian suuri. Tarkastellut linja- ja kuorma-auton korivalmistuksen ratkaisut toivat nämä ongelmat selvästi esille. Linja-auton koko alustan kattava kori osoittautui ongelmalliseksi useista syistä, joista keskeisimmiksi nousivat korirakenteen jäykkyyteen vaikuttavat monimutkaiset riippuvuussuhteet, suuresta variaatioiden määrästä johtuva alhainen toistuvuus ja joustavuuden korostunut asema. Massatuotantoa muistuttavassa kuorma-auton valmistuksessa ongelmat oli vältetty osaksi pitkään tapahtuneen järjestelmällisen moduloinnin vuoksi, mutta myös ohjaamon korirakenteen itsenäisyys muusta rakenteesta sekä tuotannon riittävä volyymi olivat tärkeitä onnistumisen rakenteita. Näiden esimerkkien valossa on oletettavaa, että keskimääräiset onnistumisedellytykset ovat suuremmat siirryttäessä massatuotannosta massaräätälöintiin kuin yksittäistuotannosta edeten.

Yksiselitteistä ohjetta tai metodiikkaa toimivan muuntuvan järjestelmän luomiseen ei ole tarjolla, vaikka erilaisia teorioita onkin esitetty. Niiden on osoitettu soveltuvan huonosti modulaaristen tuoteperheiden määrittelyyn ja siksi sopivan ratkaisun löytyminen on aina riippuvainen yrityksen omista lähtökohdista ja tavoitteista. Ensimmäinen yritys ei välttämättä ole loppuratkaisu vaan järjestelmän kehittäminen on pitkäjänteistä,

jopa useille vuosikymmenille ulottuvaa kehitystyötä, missä aiemman vaiheen tulokset toimivat seuraavan vaiheen lähtökohtana.

3. Platformeista

Platform-pohjaisen tuotekehityksen ydinajatus on jakaminen (Nielsen 2009, s.14).

Kirjallisuudessa platformeja nähdään monella tavalla, joiden määrittely vaihtelee näkökulman ja tavoitteen mukaan. Näistä yleistämällä Kristjansson määrittelee platformin kokoelmaksi yrityksen ydin assetteja, joita uudelleen käyttämällä saavutetaan kilpailuetua. (Lehtonen 2007).

Termiä platform käytetään eri yhteyksissä eri merkityksissä. Yksinkertaisimmillaan se voi tarkoittaa systemaattista standardointia. Edistyneempi tulkinta sisältää kokoonpano- tai toiminnallisia moduuleja. Eroavaisuuksia löytyy myös siitä ovatko elementit suunnitteluperusteita, valmiita moduuleja vai jotakin siltä väliltä. Lehtonen kuvaa hyvin toimivan platformin mahdollistavan tuoteinstanssien kokoamisen asiakkaalle systemaattisen muuntelun avulla valmiiden moduulien avulla noudattaen määriteltyä tuotearkkitehtuuria. (Lehtonen 2007).

Platformin, alustan, avulla pyritään tietoisesti käyttämään tuoteperheen tuotteissa asioita, *things*, joilla saadaan aikaiseksi (myönteinen) vaikutus, *effect*, yrityksen toiminnassa. Tyypillisiä asioita ovat tuotteen komponentit ja tuotearkkitehtuurit, mutta myös tuotantovälineet, prosessit ja toimitusketjut. Tyypillisiä vaikutuksia ovat tilaus-toimitusajan lyhentyminen tai tuotekustannusten alentumiset. (Nielsen 2009, s. 14.)

Keskeisiä osia tuotekehityksessä ovat itse platformin ja tuoteperheen kehitys. Tuoteperheen muodostavat tuotteet, joilla on jokin yhteinen tekijä. Kirjallisuudessa pidetään yhteistä piirrettä tekijänä, jolla tuoteperheet erottuvat perinteisistä tuote portfolioista. (Nielsen 2009, s. 14–15).

3.1. Tuoteplatformeista

Yleinen käsitys on, että platformit jakaantuvat kolmeen ryhmään. *Building block platform*, yhteisiin rakenneosiin kuten komponentteihin, moduuleihin ja osiin perustuva. *Cornerstone platform*, jossa koko tuoteperhe perustuu samaan alustaan. *Skeletal platform*, jossa rakenteen perustana on yhteinen arkkitehtuuri. Merkittävin ero näiden mallien välillä on näkökulma, josta platformia lähestytään. Lähestymistavalla on merkitys siihen, kuinka platform komponentteja hallitaan. (Nielsen 2009, s. 15).

Alkuperäisessä building block -ajattelussa tuoteperhe on joukko tuotteita, jotka voidaan kehittää ja tuottaa tehokkaasti käytettävissä olevista komponenteista (Meyer Nielsenin 2009 mukaan). Robertson (Nielsen 2009 mukaan) laajentaa käsitettä abstraktim-

maksi kokoelmaksi yrityksen assetteja. Kirjallisuudessa näitä kutsutaan yleisesti moduuleiksi. Koska kyseessä ovat muunteluun tarkoitetut moduulit, ovat ne myös Lehtosen M-moduuleja (luku 2). Tässä kontekstissa kyse on standardoiduista osista. (Nielsen 2009, s. 15).

Teollisuus on yleisesti omaksunut cornerstone platformin osaksi toimintaansa, tilanteissa tuotteet perustuvat samaan lähtökohtaan. Luvussa 2.1.1 mainitut teollisuusratkaisut, akkujärjestelmä ja piirilevy, ovat näistä esimerkkejä.

Abstraktein platformeista on skeletal platform. Sen perustana on geneerinen arkkitehtuuri, joka toimii alustana tuoteperheen tuotteille (Nielsen 2009, s. 17). Arkkitehtuurin komponentit ovat *placeholdereita*, mustia laatikoita, kuvaamassa vaihtokelpoisia elementtejä. Esimerkkinä tällaisesta voi pitää esimerkiksi ohjelmistovalmistajaa, joka toimittaa eri käyttötarkoituksiin tietokantapohjaisia verkkosovelluksia. Samat yleiset rakenteet voidaan toistaa tuotteesta toiseen vaikka itse sovellusalue muuttuu. Tässä työssä suoritetut ratkaisut ovat esimerkkejä skeletal platform toteutuksesta.

3.2. Samankaltaisuuden lisäämisestä

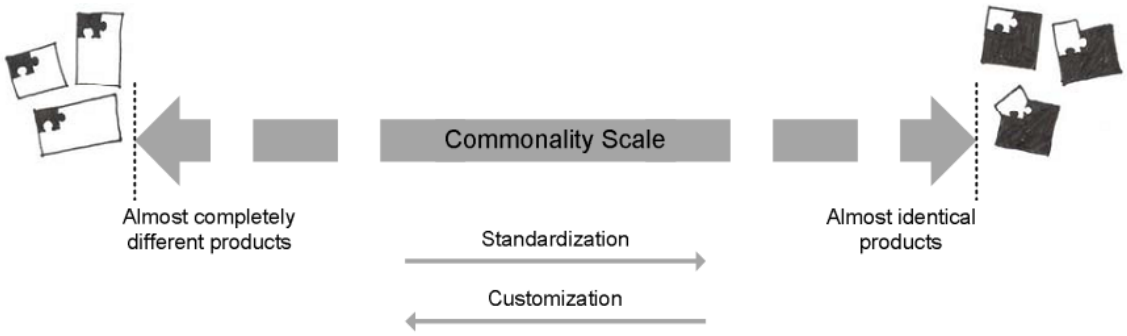
Tavanomainen tapa lisätä samankaltaisuutta tuotteiden välillä on luoda niiden välille modulaarista arkkitehtuuria. Ulrichin (1995) mukaan tuotearkkitehtuurit ovat yleensä määriteltyjä toiminnallisia elementtejä, kuvauksia toiminnallisista elementeistä fyysisiksi komponenteiksi, ja määriteltyjä rajapintoja komponenttien välillä.

Modulaarisessa arkkitehtuurissa tuote pohjautuu tarkasti rajattuihin (itsenäisiin) moduuleihin, joita yhdistää määritellyt rajapinnat. Integroiduissa ratkaisuissa komponentit ovat sidosteisiä ja vaikeasti rajattavia. Komponentin muutokset modulaarisessa arkkitehtuurissa sallii paikalliset muutokset ilman vaikusta muuhun tuotteeseen. Integroidussa ratkaisussa yksittäisenkin muutoksen vaikutukset voivat olla laajat. (Ulrich ja Eppinger 1995). Reaalimaailmassa harvoin päästään täysin ideaalisiin ratkaisuihin ja siksi Ulrich ja Tung (1991) esittävät modulaarisuuden olevan suhteellista.

Tuotetasoiset kustannusten, suorituskyvyn tai laadun parantamiseen tähtäävät toimenpiteen johtavat yleensä integroiduimpiin ratkaisuihin. Modulaarisen arkkitehtuurin joustavuus yleensä nostaa kustannuksia ja siksi ominaisuuksien tulee olla hyvin perusteltuja. (Nielsen 2009, s. 18).

Lehtonen esittää, että on myös muita perusteita luoda modulaarisuutta kuin pelkkä perinteinen, fyysisiin ominaisuuksiin perustuva jaottelu. Perinteinen lähestymistapa johtaa usein moduulijärjestelmään, jossa edut saavutetaan ensisijaisesti moduulijärjestelmän kuin moduulien kautta. Modulaarisuutta voi olla myös ilman moduulijärjestelmää. Lehtosen mukaan toiminnallisuutta ei voida pitää automaattisesti lähtökohtana moduulijaottelulle. (Lehtonen 2007, s. 206.)

Modulaarista arkkitehtuuria yleensä edeltää jokin vaatimus, joka voi olla niin yhtenäisyyttä kuin toiminnallisuutta edistävä (Erixon 1998 Nielsenin 2009 mukaan). Jälkimmäisestä esimerkkeinä Nielsen (2009) mainitsee ylläpidon, huollettavuuden, päivitettävyyden ja kokoonpanon. Nielsen (2009) havainnollistaa standardoinnin ja räätälöinnin välistä yhteyttä samankaltaisuuden asteikolla (kuva 1). Eri teollisuuden aloilla samankaltaisuuksien määrä vaihtelee täysin erilaisista lähes identtisiin tuotteisiin ja siten moduulien käyttötarkoitus vaihtelee riippuen siitä miten yrityksen toiminta sijoittuu tällä asteikolla.



Kuva 1. *Tuotteiden samankaltaisuuden aste Nielsen (2009, s. 19) mukaan.*

Yritykset, joiden toiminnassa samankaltaisuus on alhainen, pystyvät yleensä hyötymään platform-pohjaisesta tuotekehityksestä. Komponenttien standardointi, prosessit ja uusien tuoteperheiden luominen voivat avata jopa uusia toimintatapoja. Yritykset, joiden tuotteet perustuvat korkeaan samankaltaisuuteen, kokevat ongelmia siinä kuinka samankaltaisuutta voidaan lisätä, kun kaikki mikä on jaettavissa, on jo jaettua. (Nielsen 2009, s. 19–20.) Jälkimmäisten osalta lisäarvoa haetaan siitä, että tuotteita voidaan räätälöidä. Edellytyksenä on, että löytyy käyttäjiä, jotka haluavat siitä maksaa.

Potentiaalinen ongelma, joka liittyy räätälöinnin lisäämiseen, on että tuotteisiin muodostuu lisäarvotonta varioituvuutta, *non-value-added variety*, jota suurin osa käyttäjistä ei tunnista, ei välitä tai ei halua maksaa lisää (Ramdas 2003, Nielsen 2009 mukaan). On tunnistettu (Desai et al. 2001 Nielsen 2009 mukaan), että lisäarvotonta varioituvuutta aiheutuu myös huonosta tiedonkulusta.

3.3. Varianteista

Modulointi antaa mahdollisuuden monipuolisiin variantteihin. Loppukäyttäjän näkökulmasta tuotteiden välillä pitää olla vaihtoehtoisuutta, *trade off*. Tämä vaihtoehtoisuus voi muodostua useasta eri tekijästä. Näitä tekijöitä on havainnollistettu taulukossa 1. Mikä on kannattavaa variointia pitää tapauskohtaisesti arvioida esimerkiksi tuotemäärien ja varianttien tuomien ominaisuuksien avulla. Esimerkiksi suuri varianttien määrä on

käytännössä vaikeasti hallittavissa. Siksi määrää ei kannata pitää ensisijaisena tavoitteena, koska tuoteperheen liian samankaltaiset tuotteet kilpailevat keskenään ja ominaisuuksiltaan parempi tuote tekee kilpailevasta tarpeettoman. Uusi tuotteen versio voi olla korvaava tai variantti, jos saatavuuteen tai hintaan liittyy rajoituksia. Vaikka tuotteet voivatkin olla teoreettisessa mielessä variantteja, niin todellisuudessa variantin luomat kustannukset saattavat olla peruste sen hylkäämiselle. (Nielsen 2009, s. 23.)

Taulukko 1. *Version ja variantin eroista.*

Ominaisuus	**Versio**	**Variantti**	**Vaikutus**
Eri tuotenimillä markkinoitava teknisesti samankaltainen tuote.		X	Tuotteiden välillä on eroavaisuutta, jos tuotemerkillä on vaikutusta.
Ominaisuuksiltaan päivitetty uusi tuote.	X		Uusi tuote korvaa vanhemman. Vanhempi tuote tulisi poistaa.
Ominaisuuksiltaan päivitetty uusi tuote.		X	Jos vanhemmalla tuotteella on kysyntää, hinnan, saatavuuden tai muun tekijän johdosta.

3.4. Lähtökohdat ja motivaatio

Milloin ja miten yritykset löytävät platform-ratkaisut? Nielsenin (2009, s. 92–94) näkemyksen mukaan siihen tarvitaan sopiva aika ja motivaatio. Aika on tässä yhteydessä hyvin suhteellinen käsite. Se tarkoittaa vaihetta, jolloin yrityksen tuotteet ovat riittävän matuureja – konseptit ja teknologia ovat riittävän vakiintuneita stabiilin platformin kehittämiselle ja niillä on vielä elinkaarta jäljellä. Mutta, myös yrityksellä tulee olla valmius pitkäaikaiseen investointiin ja luottamusta siihen, että investoinnin tuomia etuja pystytään soveltamaan.

Kokemuksella on vahva osuus siihen kuinka platform-projekti-ideat löytyvät. Nielsen (2009, s. 92–102) on luokitellut lähtöideoille tekijöitä ja niiden taustoja (taulukko 2). Kun yrityksestä löytyy riittävää tuotekokemusta ja -näkemystä toiminnan eri tasoilta, ja pystytään tukemaan uudella tiedolla, edellytykset ideoille ovat olemassa.

Taulukko 2. *Platform-projekti-ideoiden muodostuminen (Nielsen 2009, s. 92–102).*

Idean lähde	**Kuvaus**
Tutustuminen	Nielsenin (2009, ss. 92-102) mukaan yritykset kykenevät tuottamaan omia platform-ratkaisuja, mutta ne eivät välttämättä tunnis-

	ta sellaista ilman apua. Kun yritykset ovat saaneet tietoa tai ymmärrystä siitä, mitä platformit ovat, ovat he pystyneet tunnistamaan omasta toiminnastaan vastaavia mahdollisuuksia. Nämä ilmeiset, niin sanotut *low-hanging fruits*, projektit, ovat yleensä ratkaisuja ongelmiin, joita on yrityksessä tunnistettu ja siksi esimerkiksi työntekijät ovat vastaanottavaisempia erilaisille ratkaisuille.
Ajoitus	Platform projektin onnistuminen on kiinni ajoituksesta, jossa ulkoisten tekijöiden tulee olla soveltuvasti kohdallaan. Ulkoinen paine ohjaa yrityksen toimintaa, tuotteen konsepti ja teknologia ovat vakiintunutta sekä tuotteella on vielä elinkaarta jäljellä. Lisäksi yrityksen kypsyys aloittaa, tuottaa ja tukea projektia ovat menestyksen merkittäviä tekijöitä.
Osastot	Kaupalliset näkökulmat, toiminnalliset ja tekniset konfiguraatiot sekä spesifikaatiot antavat myynnin ja marketoinnille mahdollisuuden havaita ja ehdottaa uusia näkökulmia tuotesuunnittelulle. Jossain määrin rajallinen tekninen tietämys jopa toimii ovien avaajana uusille ajatuksille. Tuotanto-osastoilta tulevat ideat usein keskittyvät tuotannollisten – valmistuksen, jakelun tai kapasiteetin – ongelmien ratkaisuihin. Tyypillistä ideoille on, että ne avaavat edullisia variaatiomahdollisuuksia tai luovat jopa edellytyksiä innovatiivisille tuotteille. Näihin riskinä liittyy se, että tuotantoperusteiset muutokset eivät välttämättä ole kuitenkaan linjassa asiakasvaatimusten tai -arvojen kanssa. Tuotekehityksestä on luonnollinen paikka, josta ideoita syntyy. Tekninen osaaminen tuotteista ja niiden rakenteista antaa hyvän pohjan erilaisille esityksille. Ja usein osallistuminen useampaan projektiin antaa mahdollisuuden käyttää tehtyjä ratkaisuja uudelleen. Tähän uudelleen käyttömahdollisuuteen liittyy riski, jossa vain omaa suunnittelua käytetään uudelleen käytettävistä teknisistä apuvälineistä huolimatta.
Ratkaisujen ehdottaja	Toisinaan ehdotukset voivat tulla myös yksittäisiltä henkilöiltä. Sellaisia voivat olla esimerkiksi kokeneet työntekijät, joilla on näkemystä kehittämisestä, muttei välttämättä siitä mitkä ovat platformin vaikutukset. Usein näiden henkilöiden kohdalla on kuitenkin ongelmana, ettei heillä ole mahdollisuutta ehdottaa tai saada ideoitaan läpi organisaatiossa. Managerit osallistuvat useampaan projektiin ja siten heille usein muodostuu työntekijöitä laajempaa näkemystä siitä, mitä yhteisiä ja varioituvia elementtejä on. Hyvin usein heillä on myös ym-

	märrystä yrityksen tavoitteista ja pystyvät sovittamaan näkemystään myös pidemmälle aikavälille. Platform asiantuntijat yleensä pystyvät tunnistamaan mahdollisen idean helpommin tai ovat aktiivisempia sellaisen soveltamiseen eri tilanteissa. Toisaalta, riski heidän kohdallaan on, että innostus platformin soveltamiseen ohittaa asiakastarpeen.

Ideoiden lisäksi tarvitaan vielä motivaatiota. Platformin kehitys on aina investointi. Kehityspäätöstä yleensä edeltää jokin sortin pakko, joka voi johtua markkinatilanteessa, tuotehallinnasta tai muusta vastaavasta syystä. Yrityksen toimintaan haetaan muutosta ja siten keskeinen motivaatio platform ratkaisujen etsimiselle on tehokkuuden kehittäminen, kuten kustannussäästöjen hakemista yhtenäisistä työkaluista tai läpimenoajan lyhentämistä valmista suunnittelua hyödyntämällä.

Oleellinen osa platform hankkeen onnistumiselle on koko yrityksen sitoutuminen siihen. Toiminnalla tulee olla yritysjohdon tuki ja osallistuvilla riittävä motivaatio tehdä muutoksia omaan toimintaansa. Käytännön esimerkkinä Nielsen (2009) esittelee LEGOlla toteutetun Wheels-projektin, jossa pyrkimyksenä oli rajata käytettävien pyöräkomponenttien määrää. Samansuuntaisia projekteja oli yrityksessä yritetty tehdä jo useampia, mutta vasta vuoden 2006 projekti onnistui, kun koko organisaatio kiinnostui siitä. Se kuitenkin edellytti taustoittavaa toimintaa: sisäistä markkinointia, jolla säännöllisesti esitettiin ja korostettiin järjestelmän etuja, kerätyn tiedon analysointia ja toiminnan, myös platformin, monipuolista kehittämistä sekä toimintaa tukevia aliprojekteja. (Nielsen 2009, ss. 48–67, 102). Sekä Lehtonen (2007) että Baldwin ja Clark (2000) ovat havainneet, että myös tuotekehitysorganisaation pitää rakentua modulaarisen rakenteen mukaisesti toimiakseen tehokkaasti.

Eräs riskeistä on, kuinka investointi saadaan kannattamaan. Nielsenin (2009, s. 25) mukaan platform-kehitys on investointi, joka maksaa takaisin vain jos moduuleja käytetään. Tehokas käyttäminen edellyttää järjestelmän uusiutumista määräajoin, siten että se vastaa ajan tuomiin haasteisiin. Platform jää ajasta jälkeen, jos siitä ei pidetä jatkuvasti huolta (Nielsenin 2009, s. 88). Kantava ajatus on, että tehtyä investointia ylläpidetään säännöllisesti esimerkiksi alustaa päivittämällä, generaatiot, ja tuoteohjelmaa tarkastamalla, *spring cleaning*. Siihen saattaa lisäksi liittyä erilaisia tuotekehityshankkeita, jotka luovat tarvetta uusille ominaisuuksille.

3.5. Kehittämisestä

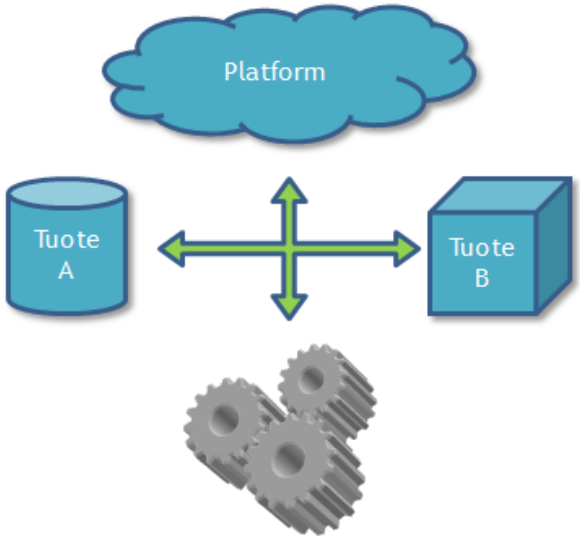
Kehitystyö voidaan jakaa kahteen pääosaan: platform- ja tuotekehitykseen. Nielsen (2009, s. 38) jakaa nämä kehitysvaiheet valmistavaksi, *preparation*, ja toteuttavaksi, *execution*. Valmistava vaihe luo ne puitteet, jolle koko järjestelmä perustuu. Toteuttava tuotekehitys-vaihe käyttää järjestelmää hyväkseen luodakseen tuoteyksilöitä. Lähtökohtaisesti valmistavan vaiheen tulee edeltää toteuttavaa. Todellisuudessa tämä ei aina rea-

lisoidu, vaan dynaamisessa kehityksessä platform-komponentitkin saatetaan kehittää projektin aikana. Nielsen (2009, s. 38) ei pidä tätä ongelmana, mutta korostaa samassa yhteydessä, että platform ja tuotekehityksen pitää olla keskenään hyvin synkronoitua, jotta komponentit ovat valmiina silloin, kun niitä tarvitaan.

Kehitys kohtaakin usein aikataulupaineita. Ongelmaksi platform-kehityksessä muodostuu se, että platform kehitys pitää aloittaa vasta sitten, kun on riittävästi tietoa tarpeista ja mahdollisuuksista. Varhainen aloitus helposti johtaa tilanteeseen, missä komponentit eivät ole riittävän määriteltyjä ja siten irrelevantteja tai laadullisesti huonoja lopputuotteelle. Mahdollisimman myöhälle siirretty aloitus korjaa näitä ongelmia, mutta vastaavasti saattaa aiheuttaa turhaa odotusta ja viivästystä niiden käytölle. (Nielsen 2009, s. 135). Kuva 2 havainnollistaa platform-kehityksen ajastamisen reunaehtoja.

Kuva 2. *Platform projektin ajastaminen (Nielsen 2009, s. 135).*

Vaikka platform- ja tuotekehitys erotetaankin eri toiminnoiksi, ei se tarkoita sitä, että ne ovat toisistaan riippumattomat. Tiedonsiirron tulee olla kahdensuuntaista: platform-kehityksen pitää on tietoinen tuotekehityksen suunnista ja toisaalta tuotekehitys saa jatkuvasti tietoa platformeista (Nielsen 2009, s. 133–134). Tuote- ja suunnittelutiedon tulee välittyä myös tuotteiden ja toimintojen välillä (kuva 3). Nielsen (2009, s. 98) on havainnoinut, että tiedon välittyminen voi olla ongelmallista. Suunnittelijat mielellään käyttävät jo tehtyä uudelleen, esimerkiksi muottien CAD komponentteja, mutta helposti rajoittuvat vain itse suunnittelemiensa osien ja komponenttien uudelleenkäyttöön. Tai tieto ei siirry eri toimintojen välillä: vaikka suunnittelussa käytetään komponentteja uudelleen, ei tieto välity valmistukseen, jossa CAM-työstöradat saatetaan suunnitella uudelleen alusta asti hyödyntämättä aiempaa vaikka muutokset ovat olla hyvinkin vähäisiä.



Kuva 3. *Tiedonjakaminen platformin, tuotteiden ja eri toimintojen välillä.*

Tilanteissa, joissa arkkitehtuurin tai komponenttien jakaminen ei sovellu, Nielsen (2009, s. 98) näkee mahdollisina lähestymistapoina tietoon, *knowledge platform*, ja/tai puolivalmisteeseen perustuvan platformin. Tietoon pohjautuvassa mallissa tieto on konseptin keskeinen abstrahointiväline ja tuotekehitykselle pienimuotoiset muokkaukset ovat sallittuja. Puolivalmisteeseen perustuva platform ei välttämättä abstraktoi perustaa vaan luo yhteisen kiinteän perustan, jota muokataan asiakastarpeen mukaan. Korvaamalla abstraktio puolivalmisteella tuotealustassa, muut jatkovaiheet kuten valmistus pystytään perustamaan aina samoihin lähtökohtiin.

3.6. Platform toimintamallina

Platform-pohjaisen tuotekehityksen perusta on jakaminen, jossa yrityksen keskeisimmät assetit uudelleen käyttämällä saavutetaan kilpailuetua. Siinä pyritään tietoisesti käyttämään tuoteperheen tuotteissa asioita, joilla saadaan vaikutus yrityksen toimintaan. Platformin muoto vaihtelee tarpeen mukaan, yksinkertaisesta systemaattisesti standardoidusta järjestelmästä edistyneempiin kokoonpano- tai toimintamoduuleihin toteuttaen määriteltyä tuotearkkitehtuuria. Tyypillisiä vaikutuksia ovat tilaus-tuotannon läpäisyn nopeutuminen tai tuotekustannusten alentuminen.

Platformilla on lähtökohdat ja motivaatio. Yleensä lähtökohdat määrittyvät olemassa olevasta tuotevalikoimasta ja sen vaatimuksista. Modulaarinen arkkitehtuuri perustuu vaatimukseen. Vaatimus voi olla yhtenäisyyttä tai toiminnallisuutta edellyttävä. Mitä korkeampi on samankaltaisuuden vaatimus, sitä korkeampi on standardoinnin aste. Motivaatiota luo pakkotilanne: yrityksen on mukautettava toimintatapansa säilyttääkseen toimintaedellytyksensä rationalisoimalla toimintaansa.

Platform arkkitehtuuri on kompromissi erilaisten painotusten välillä. Sillä pyritään ratkaisuihin, joilla aikaansaadaan moduulien välille alhainen määrä keskinäisiä sidoksia.

Toisaalta tuotetasoiset optimoinnit johtavat yleensä integroiduimpiin ratkaisuihin. Koska joustavuus on moduulijärjestelmässä kallis ominaisuus, on perusteltua tutkia myös muita lähestymistapoja kuin toiminnalliset tai valmistukselliset moduulit. Siksi on kyseenalaistettu toiminnallisuuden lähtökohtaisuus moduulijaottelulle. Esitettynä on näkemys, että modulaarisuutta voi olla myös ilman moduulijärjestelmää.

Platformeja ryhmitellään niiden lähtökohtien mukaan. Building block-platform perustuu tarkkaan standardointiin, jossa niiden keskinäinen rakenne ei ole tarkastelun kannalta merkitsevä. Cornerstone platform perustuu yhteiseen perusratkaisuun ja skeletal platform tuotearkkitehtuurin määrittelyyn. Lisäksi on esitettynä sovellus cornerstone platformista, jossa lähtökohtana on tuotteen puolivalmiste, josta räätälöidään asiakasvaatimusten mukainen tuote.

Tuotekehityksen kannalta perusedellytys on, että kehitys jakautuu valmistavaan ja toteuttavaan vaiheeseen. Valmistava kehitys luo järjestelmän perusteet ja toteuttava vaihe realisoituu järjestelmään perustuvina tuotteina. Onnistuneen platformin pohjana on näiden toimintojen keskinäinen suhde, jossa vaatimusten, ajan ja tiedonhallinta ovat keskeisessä roolissa.

4. Toimialasta

Logistiikan alalta löytyy erilaisia kuljetusliikkeitä alkaen suurista valtakunnallisestikin merkittävistä toimijoista, keskisuuria paikallisen tason operaattoreita sekä pieniä yksityisiä kuljetusyrittäjiä. Yritysten tarpeet vaihtelevat teknologisen kiinnostuksen, palvelutarpeen määrän ja kustannustason mukaan. Edelleen jokaisella yrityksellä on oma toimintamallinsa ja -strategiansa; nyrkkisääntönä on kuitenkin että isot kuorivat kerman päältä ja pienet pärjäävät vain erikoistumalla joka näkyy toimialalla vahvana verkottumisena ja alihankintaketjuina.

Yksiselitteisen kaiken kattavan määrittelyn löytäminen on erittäin haasteellista. Tarpeet ovat erilaisia: isot toimijat haluavat hallita kokonaisuutta, pienemmille riittää työaikojen ja paikkatiedon seuranta. Asiakaskohtaisesti hajontaa lisää kuljetusyrityksien sisäiset, vakiintuneet toimintatavat ja käytännöt.

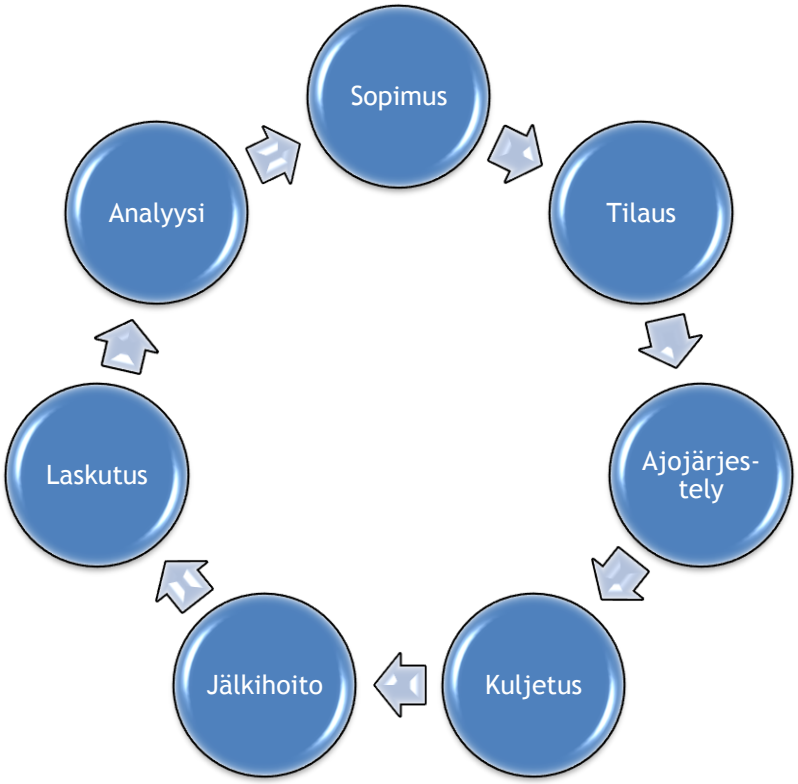
Selkeän ratkaisun kannalta on ongelmallista, että alaa koskeva lainsäädäntö tai vallitsevat käytänteet ovat epämääräisiä. Esimerkkinä voidaan mainita rahtikirja, joka lain mukaan pitää olla jokaisen kuljetuksen mukana. Säädös määrittelee mitä siinä pitää vähintään olla, mutta jättää vielä paljon pelivaraa: "asianosaiset voivat merkitä rahtikirjaan muitakin tarpeelliseksi katsomiaan tietoja". (Tiekuljetussopimuslaki 23.3.1979/345 6§) Sääntely tapahtuukin pääasiassa valitun kuljetusyrityksen toimintamallin puitteissa joko yrityksen omasta aloitteesta, yhteistyökumppanin ja sidosryhmien vaatimuksista. Toimialan sisäisen standardoinnin merkitys on vielä vähäinen. Joitakin yrityksiä tämän saavuttamiseksi on meneillään, mutta työ on vielä alkuvaiheissaan.

Oma lukunsa on sitten kuljetusalan terminologia, joka vaihtelee vahvasti yrityksestä toiseen ja sidoksissa yrityksen toimintamalliin ja -kulttuuriin. Karrikoivasti voidaan sanoa, omakohtaiseen kokemukseen perustuen, että eri asiakkaiden kanssa keskustellessa on ilmennyt tarvetta suomi-kuljetusliike-suomi sanakirjalle. Keskeisimpiä käsitteitä on selitettynä luvussa 4.1.

Kuten edellä todettiin, ei eri yritysten välille voida löytää mallia, joka sopisi kaikille. Siksi yhteisten piirteiden löytämiseksi on tarpeen nostaa tarkastelun abstraktiotasoa. Tarkasteltaessa yritysten tuotantoprosesseja voidaan löytää selkeitä työvaiheita, jotka toistuvat yrityksestä toiseen (kuva 4) (Leinonen 2008, Mäki-Mikkilä 2008).

- Kuljetusliikkeen ja kuljetustarpeen omistajan välillä on olemassa sopimus,
- kuljetustarpeen omistaja tilaa sopimuksen mukaista kuljetuspalvelua,
- ajojärjestelyssä tilaukselle osoitetaan kuljetusresurssi,
- varsinainen kuljetustapahtuma,

- jälkihoito, jossa tarkistetaan poikkeamat tilauksen ja toteutuman välillä,
- laskutus sekä
- valveutuneimmissa yrityksissä kerätyn datan analysointi uusien sopimusten pohjaksi.



Kuva 4. *Tuotantoprosessin eri vaiheet.*

Prosessien sisäiset tapahtumat ja aliprosessit vaihtelevat yritysten mukaan. Tätä projektia tehdessä oli yllättävää – alalle ulkopuolisena – havaita, että yrityksestä riippumatta tuotantoon liittyy paljon käsityövaltaisia ja tarkkuutta vaativia työvaiheita.

Perinteinen kuljetuspalvelu, toisin sanoen kuljetustarpeen tyydyttäminen tavaran siirtämisellä kohteesta A kohteeseen B ei enää riitä, vaan kuljetusyritykset hankkivat kilpailuetua tehostamalla omaa toimintaansa, tarjoamalla sekä lisäpalveluita että tarkempaa tietoa suoritetuista kuljetuksista. Tätä Leinonen (2008) kutsuu palveluketjuksi. Keskusteluissa eri kuljetusliikkeiden kanssa on käynyt ilmi, että myös poikkeamien hallinta on oleellinen osa palveluketjua. (Leinonen 2008, Mäki-Mikkilä 2008)

4.1. Termistöstä

Tässä luvussa käsitellään ja kuvataan kuljetusalan keskeisimpiä termejä siten, kun ne on tarkastelun alla olevassa järjestelmässä käsitetty.

4.1.1. Auto

Auto on ajoneuvon, mahdollisen perävaunun ja kuljettajan muodostama kokonaisuus.

4.1.2. Ajoneuvo

Ajoneuvo on kuljetuksen suorittamiseen käytettävä kulkuneuvo.

4.1.3. Reitti tai lenkki

Reitti ja lenkki ovat tilauksesta realisoituneita ajojärjesteltyjä ajoja, jotka välitetään kuljetuksen suorittavalle autolle. Runkoliikenteessä käytetään termiä *reitti* ja jakeluliikenteessä termiä *lenkki*.

4.1.4. Oletusreitti tai lenkki

Ennakolta suunniteltu reitti tai lenkki, jota käytetään ajojärjestelyrutiinien vähentämiseen ja automatisointiin. Runkoliikenteessä käytetään termiä *oletusreitti* ja jakeluliikenteessä termiä *oletuslenkki*.

4.1.5. Sopimusreitti

Runkoliikenteen kuljetussopimuksessa on määritelty asiakkaan kanssa sovittu reitti ja sen ehdot. Sopimusreitti määrittää vain alku- ja loppupaikan ja siten voi olla erilainen kuin tosiasiassa tapahtuva kuljetusketju.

4.1.6. Kuljetusprofiili tai -vaade

Kuljetuksen lisämääre, jolla huomioidaan kuljetukseen liittyviä erikoisjärjestelyjä. Kuljetusvaade voi olla esimerkiksi *pakaste* – jolloin tuotteet kuljetetaan pakastimella varustetulla ajoneuvolla – tai *ADR* – European Agreement concerning the international carriage of Dangerous goods by Road, vaarallisten aineiden kuljetus, jolloin kuljettajalta vaaditaan ADR-tutkinto ja ajoneuvolta ADR-varusteet (Ajoneuvohallintokeskus 2009).

4.1.7. Kuljetussopimus

Kuljetusliikkeen ja kuljetusta tilaavan asiakkaan, tilaaja, välille tehty sopimus, jossa asetetaan kuljetuksen ehdot. Kuljetussopimuksen rakenne on erilainen eri toimintomallien välillä. Yhteistä näille on, että sopimuksessa määritellään kuljetettavat tuotteet, kuljetusprofiilit ja lisäpalvelut sekä näihin sovellettavat hinnoitteluperusteet ja -säännöt.

Runkoliikenteen kuljetussopimuksessa määritellään ajettavat reitit ja niiden aikataulurajat. Jakeluliikenteen sopimuksessa määritellään jakelualueet.

4.1.8. Aikaikkuna

Lastauksen ja purun ajankohta, jonka aikana tapahtuva toiminta on sopimuksenmukainen.

4.1.9. Kuljetustarve

Kuljetustarve on tuote, joka pitää kuljettaa pisteestä A pisteeseen B.

4.1.10. Kuljetusalusta

Tuote kerätään alustalle kuljetusta varten. Alusta voi olla esimerkiksi fin- tai euro-lava, maito- tai tukkurullakko, liha- tai elintarvikelaatikko ja niin edelleen.

4.1.11. Kuljetusmääräys eli rahtikirja

Lakisääteinen dokumentti, jossa on määritelty kuljetuksen kohde, lähettäjä ja vastaanottaja.

4.1.12. Ajojärjestely

Logistiikanohjauksen työvaihe, jossa määritellään kuljetustarpeelle sen kuljetuksen hoitava resurssi.

4.1.13. Jälkihoito

Ajojärjestelyn loppuvaihe, jossa käsitellään tuotantovaiheen aikana havaitut poikkeamat ja tarkistetaan ajotapahtuma.

4.1.14. Stoppi- tai pudotuspaikka

Reitillä tai lenkillä oleva pysähdyspaikka, jossa lastataan tai puretaan kuljetustarve. Samasta pisteessä voidaan tarvittaessa tehdä sekä jakelu että nouto. Järjestelmä käsittelee ne kuitenkin kahtena eri operaationa.

4.1.15. Nouto

Kuljetustarve noudetaan lähtöpaikalta terminaaliin tai keräilypisteeseen keräilyä varten.

4.1.16. Jakelu

Kuljetustarve viedään terminaalista tai keräilypisteestä pudotuspaikalle.

4.1.17. Suorajakelu

Suorajakelu on jakeluliikenteen toimintamalli, jossa jaeltavia tuotteita ei kuljeteta terminaaliin tai keräilypisteeseen vaan toimitetaan suoraan tilaajalta vastaanottajalle. Käytetään usein tilanteissa, joissa käynti terminaalin tai keräilypisteen kautta aiheuttaisi turhan kierron, kun vastaanottaja on kohtuullisesti saavutettavissa ajettavan lenkin puitteissa.

4.1.18. Sopimusmalli

Sopimusmalli on sääntökokoelma, jota sovelletaan kuljetustilaukseen. Säännöt määrittävät esimerkiksi, kuinka useampia reittitilauksia voidaan käsitellä yhdessä siten että kapasiteettia voidaan käyttää kriteerinä.

Järjestelmässä voi olla samanaikaisesti useita erilaisia sopimusmalleja. Sopimukselle määritellään siihen sovellettava malli. Sopimusmallit ovat tyypillisesti sellaisia, jotka sisältävät liikesalaisuuksiksi luettavaa tietoa. Yleistoimituksessa järjestelmässä toimitetaan vain yksinkertaiset perusmallit.

4.1.19. Sopimuksenmukaisuus

Tilaus on sopimuksenmukainen silloin, kun se täyttää asiakkaan kanssa sovitun sopimusmallin. Tilaus, joka ei ole sopimuksen mukainen aiheuttaa poikkeaman.

4.1.20. Poikkeama

Poikkeama on ennakoimaton muutos tuotantoon. Poikkeamien käsittely on merkittävä osa liiketoimintaa, sillä usein poikkeamasta aiheutuu lisäkustannuksia jollekin tai jopa kaikille kuljetuksen osapuolille. Taulukko 3 esittää esimerkinomaisesti mahdollisia poikkeamia ja niiden vaikutuksia.

Taulukko 3. *Esimerkkejä erilaisista poikkeamista ja niiden mahdollisista seurannaisvaikutuksista.*

Osapuoli	**Poikkeama**	**Mahdollinen vaikutus (esimerkiksi)**
Tilaaja	Lastattava määrä ei vastaa tilattua.	Määrän ylitys voi aiheuttaa kiireellisen lisäkapasiteetin tarpeen. Alitus saattaa aiheuttaa ylikapasiteetin, joka välillisesti voi olla pois toisesta kuljetuksesta ja siten aiheuttaa lisäkapasiteetin tarvetta.
	Tuotanto on viivästynyt.	Ajoneuvo ei pysty lastaamaan sovitussa aikataulussa, voi aiheuttaa kumuloituvaa viivästystä tai lisäkapasiteettitarvetta.
Kuljetusliike	Kuljetus on myöhästynyt.	Ajoneuvoa ei voida lastata/purkaa suunnitellusti. Voi aiheuttaa tarvetta lisäkapasiteetille tai tuotannon viivästymistä.
	Kuljetus ja/tai ajoneuvo on vaurioitunut.	Tuotteita ei ole voitu toimittaa perille sovitusti.
Vastaanottaja	Lastausta ei voida purkaa tai vastaanottoa ei ole järjestetty.	Tuotteita ei saada puretuksi suunnitellusti vaan ne täytyy tuoda välivarastoitavaksi esim. terminaaliin, ja niille pitää järjestää uusi kuljetus.

Poikkeamat käsitellään aina tapauskohtaisesti käsin, koska osa poikkeamista voi olla seurannaisvaikutusta edellisestä ja syntyneen kustannuksen jaosta voi olla epäselvyyttä. Poikkeama ei välttämättä ole negatiivinen toimenpide vaan joissakin olosuhteissa voidaan tilaajan kanssa sopia, että tämä kuljetus on sovitusti sopimuksesta poikkeava. Siitä tehdään poikkeama siksi, että sen jatkokäsittely tapahtuu käsin sovitut muutokset huomioiden.

4.2. Toimintamalleista

Tämän työn näkökulmasta kuljetusalalta voidaan löytää kaksi pääasiallista toimintastrategiaa: runko- ja jakeluliikennöinti. Yksittäinen kuljetusliike luonnollisesti valitsee

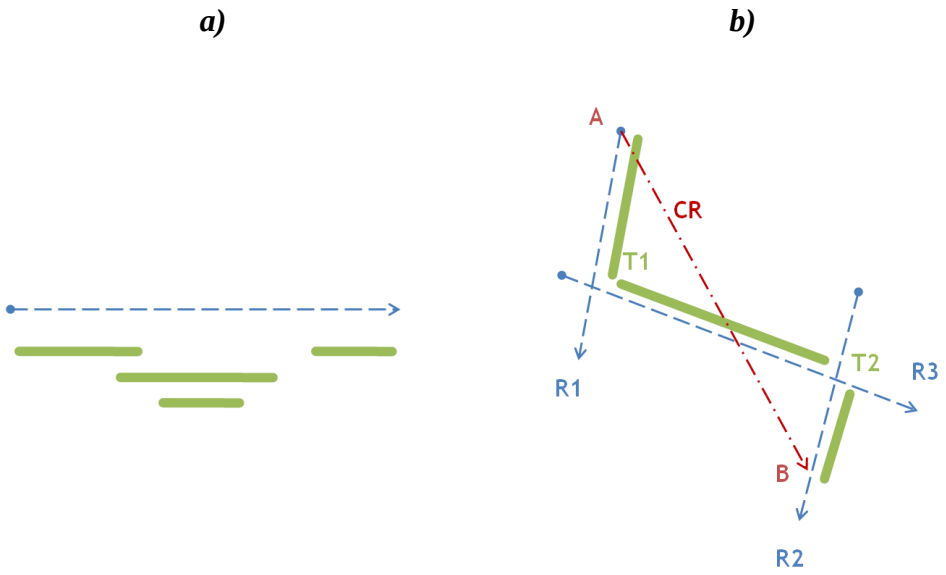
omansa, mutta valinta ei välttämättä ole mustavalkoinen vaan siihen saattaa sisältyä molempia malleja. Usein kuitenkin niin, että yritys keskittyy omaan ydinosaamiseensa ja alihankkii puuttuvan osan.

Runko- ja jakeluliikenteen ominaispiirteitä tarkastelen seuraavissa luvuissa.

4.2.1. Runkoliikenne

Puhtaimmillaan runkoliikenne on suurehkojen tuote-erien, jotka voivat olla mitä tahansa kuljetettavaa tavaraa, kuljettamista terminaalista toiseen: valmistajalta jakelutukkuun, liikennöitsijän terminaalien välistä siirtoa tai valmistajalta (teolliselle) loppukäyttäjälle. Kuljetukselle ominaista on suurehko kuljetuserä joka viedään pisteestä A pisteeseen B kokonaisuudessaan yhdelle vastaanottajalle.

Usein kuljetukset ovat säännönmukaisia ja sidotaan osaksi laajempaa ja/tai pidempikestoista reittiä. Yhdellä reitillä voi kuitenkin olla useampia tuote-eriä eri lähettäjiltä eri vastaanottajille. Runkoreitti sisältää yhden tai useamman sopimusreittiosuuden. Sopimusreitin mukainen kuljetus saatetaan viedä jopa eri runkoreittien mukana (kuva 5 a ja b).

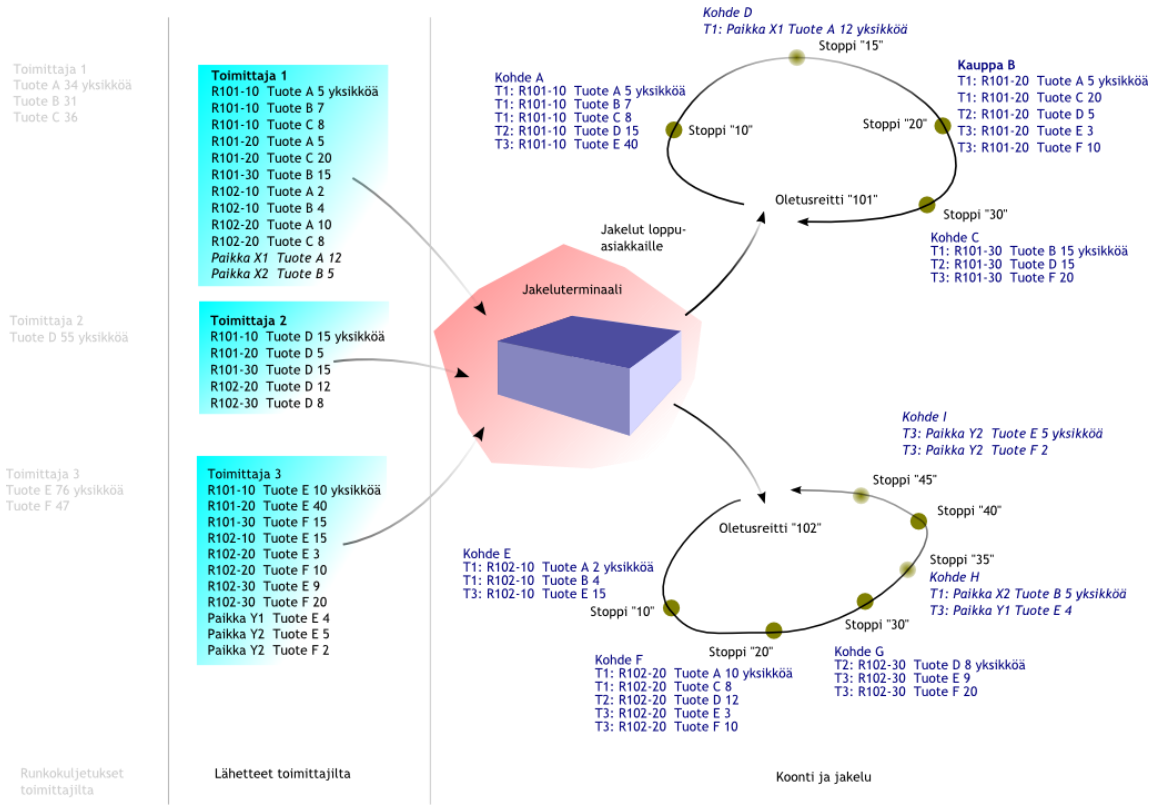


Kuva 5. a) *Reitin muodostuminen sopimusreittiosuuksista.* ***b)*** *Sopimusreitin (CR) toteutuminen useamman reitin kautta, missä R*n *on reitti ja T*n *terminaali.*

Runkoliikenteelle on edelleen ominaista, vaikkei välttämätöntä, valtakunnallisuus. Kuljetettavat matkat ovat pitkiä yleensä kaupungista toiseen, mutta mahdollisesti myös valtiosta toiseen ja kalustona käytetään pääosin rekkakalustoa.

4.2.2. Jakeluliikenne

Jakeluliikenne on pääsääntöisesti paikallista toimintaa, jossa liikennöitsijän terminaalista tai jakelijan tukusta käsin jaellaan useammasta eri lähetyserästä koottu lähetys yksittäiselle vastaanottajalle (kuva 6). Vastaanottajalle koottavat erät ovat pääsääntöisesti selvästi runkokuljetuksia pienempiä.



Kuva 6. *Terminaalista tai jakelijan tukusta käsin jaellaan useammasta eri lähetyserästä koottu lähetys yksittäiselle vastaanottajalle.*

Jakeluliikenne on säännönmukaista, yleensä paikallista ja perustuu kestoltaan maksimissaan vuorokauden kestäviin lenkkeihin. Käytettävä kalusto muodostuu pääosin paketti- ja kuorma-autoista.

Yksittäinen lenkki koostuu yhdestä tai useammasta pudotuspisteestä, stopista. Terminaalissa suoritetun koonnin tuloksena voidaan yksittäiselle vastaanottajalle toimittaa kerralla lähetys, joka koostuu useiden lähettäjien toimituksista.

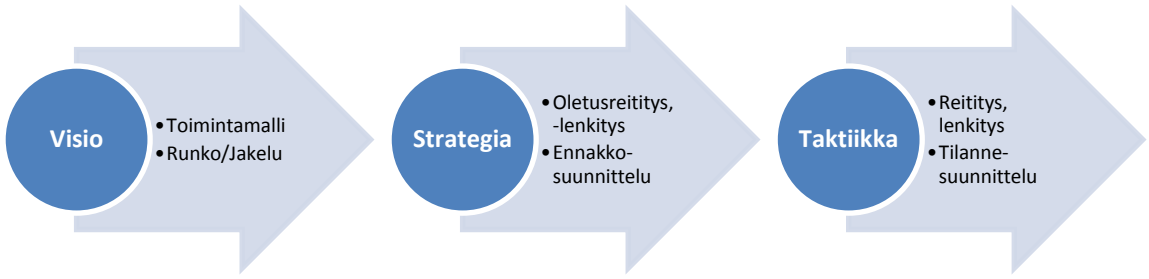
Terminä jakeluliikenne on hieman epämääräinen, sillä sen alle voidaan lukea myös nouto-, suorajakelu ja jakelutoiminta. Nouto tässä yhteydessä on pienimuotoista runkoliikennöintiä, jossa tuotteet noudetaan valmistajalta terminaaliin kerättäväksi. Jakelu on terminaalista suoritettavaa kerättyjen erien toimittamista. Suorajakelu on noudon ja jakelun yhdistelmä, jossa tuotteet noudetaan valmistajalta ja jaellaan loppuasiakkaille käyttämättä niitä lainkaan terminaalissa.

4.3. Suunnitelmallisuus

Mitä laajemmaksi kuljetusyrityksen toiminta kasvaa sitä tärkeämpää on hallita kokonaisuutta etukäteissuunnittelun kautta. Pieni kuljetusyrittäjä kykenee toimimaan ketterästi muutamien ajoneuvojen kanssa pitämällä tietoja päässään. Suurempi kuljetusyritys kymmenine ellei satoineen autoineen on vaikeuksissa ilman toimintaa tukevaa järjestelmää. Kokemusperäinen tieto on, että ammattitaitoinen ja kokenut ajojärjestelijä pysyy hallitsemaan noin kahtakymmentä autoa rutiininomaisesti päässään ilman ulkoisia apuvälineitä (Peltonen, P. 2008, Mäki-Mikkilä 2008).

Kun liikennöitsijän toiminta on vakiintunutta, alkaa reiteistä ja lenkeistä paljastua vakioituneita ajoja, joita hyväksikäyttämällä voidaan keventää päivittäistä ajojärjestelyn taakkaa.

Karkeasti jaotellen voidaan suunnitelmallisuus purkaa visioksi, strategiaksi ja taktiikaksi (kuva 7). Vision kuvatessa kuljetusyrityksen periaatteita ja toimintamallia, strategia on ennalta tehtyä suunnittelua ja taktiikka sitä miten puretaan päivän tilanne.



Kuva 7. *Tuotannon suunnittelun vaiheet.*

4.3.1. Etukäteissuunnittelu eli reittisuunnittelu

Etukäteissuunnittelu pohjautuu ennustettavuuteen. Tarkastelun alla olevassa järjestelmässä ennustettavuuden pohjana ovat kuljetussopimukset. Näiden lisäksi suunnittelussa voidaan hyödyntää järjestelmän ulkopuolista tietoa kuten tuotantolaitoksilta saaduilla ennakkoilmoituksilla ja esimerkiksi kuljetuspäällikön kokemusperäisellä tiedolla ajankohdan kuljetusmääristä ja reittimuutoksista Mäki-Mikkilä (2008) kertoo. Peltosen (2008) mukaan paikallistuntemuksen merkitys korostuu jakeluliikenteen suunnittelussa.

Suunnittelun tuloksena pyritään muodostamaan reitti, joka täyttää yrityksen optimointikriteerit mahdollisimman hyvin. Yksittäisestä reitistä käytetään nimitystä *oletusreitti*. Tiettyyn ajankohtaan, kauteen, sidotuista oletusreiteistä käytetään nimitystä *oletusreititys*. Selkeyttävänä analogiana voi tässä yhteydessä ajatella esimerkiksi linja-autojen aikatauluja; ne laaditaan tietylle aikavälille (kausi), niissä kuvataan ajettava linja (reitti) ja merkittävimmät pysähdysajat (sopimustiedot). Kuten linja-autoliikenteessäkin,

ennakolta ei tiedetä mille pysäkeille pysähdytään, vaan nämä selviävät, kun reittiä ajetaan. Ja liikennöitsijän ajamat linjat vastaavat reititystä.

Oletusreititys muodostaa ajojärjestelyn rungon, jota sitten täydennetään ajojärjestelyssä päivätason tiedolla. Oletusreitityksen perusajatuksena on poistaa ajojärjestelyyn liittyviä rutiininomaisia, toistuvia järjestelytehtäviä. Oletusreitti ei ole kuitenkaan ehdoton, vaan sitä voidaan tarvittaessa muokata tilanteen mukaan niin, ettei yksikään tilattu ajo jää tekemättä.

Oletusreititykseen voidaan myös liittää resurssienhallintaa; reitille voidaan nimetä oletuskalusto ja osoittaa kuljettajat.

Kuljetusalalla on tyypillistä erilaiset kausivaihtelut. Kausi on hieman epätarkka termi tässä, sillä siihen ei sisälly tarkkaa rajausta. Kausi voi olla päivissä mitaten lyhyt- tai pitkäkestoinen. Esimerkkinä lyhyistä ajanjaksoista on poikkeukselliset ajopäivät: pyhäpäivät kuten pääsiäinen, juhannus tai joulu, jotka tarvitsevat oman erikoishuomionsa. Yleensä näitä edeltävät ja seuraavat päivät ovat kiireisiä esimerkiksi elintarvikekuljetusten osalta toteavat Mäki-Mikkilä (2008) ja Peltonen (2008). Pitkäkestoinen kausi on esimerkiksi arjen normaaliaika ja hiljaisemman toiminnan lomakausi, jolloin ajoja suoritetaan samalla peruskaavalla pidempään.

4.3.2. Tilannesuunnittelu eli ajojärjestely

Ajojärjestelyssä päätetään viimekädessä kuinka ajot suoritetaan. Ajojärjestelijä muodostaa ajettavat reitit ja määrää niille ajoneuvot sekä kuljettajat. Ajojärjestelyssä kuljetustarpeet liitetään reittiin.

Ajojärjestely voi pahimmillaan olla hyvin hektistä toimintaa. Siksi siinä pyritään käyttämään ennakolta hyväksi havaittuja ratkaisuja uudelleen ja uudelleen. Toiminnan luonteen vuoksi on syntynyt konsepti oletusreiteistä ja reitityksestä. Ajojärjestelyn pohjana voidaan käyttää oletusreitityksen pohjalta muodostettuja reittejä, mutta ne voidaan muuttaa tai poistaa tarpeen mukaan.

Ajojärjestelyssä seurataan jatkuvasti kuljetustarpeiden määriä, sovittuja aikaikkunoita, toteutumia sekä sijaintia. Kuljetuksia voidaan siirtää myös ajoneuvojen ja reittien välillä tarpeen tullen. Tyypillisimmillään ajojärjestelyä tapahtuu siihen asti, kunnes tuotteet purettu vastaanottajalle. On normaalia, että kuljetusmäärät selviävät vasta viime hetkellä ja tyhjä kuljetustilaus on kutsu lähettää auto tekemään nouto. Tästä saattaa aiheutua lisäkapasiteettitarvetta, joka pitää ratkaista välittömästi. Ratkaisun tekeminen kuuluu ajojärjestelijälle, joka ammattitaitonsa ja kokemuksensa perusteella valitsee siihen parhaiten sopivan ratkaisun. Joskus se saattaa olla noutopyyntö lähellä olevalle toiselle ajoneuvolle, ja toisella kertaa esimerkiksi auton tilaus alihankkijalta. (Mäki-Mikkilä 2008)

Ajojärjestelyn tuloksena syntyy kuljetusmääräys eli rahtikirja, joka välitetään ajoneuvolle. Välitys suoritetaan sähköisesti, jos ajoneuvo on varustettu päätteellä tai paperilla rahtikirjana, jos ajoneuvossa ei ole päätettä.

Kun ajojärjestely toimii optimaalisesti, ajojärjestelijän työpäivästä ensimmäinen puoli kuluu edellisen päivän tapahtumien tarkistukseen ja jälkihoitoon ja jälkimmäinen puolisko seuraavan päivän suunnitteluun – ajojärjestelijän ei pidä elää tätä hetkeä (Leinonen 2008)!

4.3.3. Suunnittelun ja toteutumien seuranta

Seurannan tärkein tehtävä on ylläpitää oikeata tietoa järjestelmässä. Toisaalta se antaa tärkeätä historiatietoa liiketoiminnan kehittämiseen, mikä on se malli minkä pohjalta kannattaa toimia? Kertyneen tiedon pohjalta voidaan seurata sopimusten tarkkuutta, tarkoituksen- ja ajanmukaisuutta sekä ohjata liiketoimintaa oikeantyyppiseen toimintamalliin. Välimaa (2010) käsittelee laajemmin tiedon keruuta myöhempää analysointia varten. Varsinainen analytiikka ei sisältynyt tähän projektiin vaan on varattu tulevaisuuden kehitysoptioksi ja siten jää laajemman tarkastelun ulkopuolelle.

5. Ratkaisun perusteista

Järjestelmän kohdalla on kyseessä palvelukokonaisuus johon sisältyy useita erilaisia elementtejä kuten laitteistot, ohjelmistot, ylläpitopalvelut ja tuotekehitys. Isossa mittakaavassa on kysymys palvelukonseptin tuotteistamisesta. Tämän työn puitteissa tarkasteltavaksi otetaan vain ohjelmistot ja tuotekehitys. Näihin välillisesti liittyvät kuten ylläpitopalvelut jätetään tarkastelun ulkopuolelle.

Toimittajan valinnat perustuvat luonnollisesti yritysjohdon asettamiin tavoitteisiin, joiden pohjana on yrityksen visio, strategia ja liiketaloudelliset tavoitteet. Tavoitteena on selkeä pyrkimys liiketoiminnan kehittämiseen, jossa toimitussyklin nopeuttamisella, luotettavuudella ja toiminnan ennakoitavuudella on suuri merkitys. Ohjelmiston kannalta mielenkiintoisemmiksi kuitenkin nousevat ne keinot ja valinnat, joilla näihin pyritään vastaamaan. Kuinka näitä tavoitteita voidaan sitten saavuttaa? Miten toimintatapoja ja järjestelmää muutetaan vastaamaan tavoitteita?

Koska markkinat ovat osoittaneet, että Fleetlogisin kaltaiselle järjestelmälle on kysyntää, on luonnollista, että laajenevan asiakaskunnan tarpeet kuitenkin edellyttävät joustavampaa tapaa muunnella järjestelmää. Lisäksi lisäarvoa haetaan siten, että jo kertaalleen tehtyä työtä voidaan merkittävässä määrin hyödyntää tai uudelleen käyttää sellaisenaan.

Tuloksena on ollut hallitsemattomasti laajeneva kokonaisuus, jossa uudelleen käyttö on ollut vaikeata tai lähes mahdotonta ja eri osioiden keskinäiset riippuvuudet suuria.

Järjestelmä on perinteisesti vaatinut merkittävän määrän räätälöintejä ja ylläpitoa, jonka vuoksi järjestelmän käyttöönotto on ollut työläs toteutettava. Tehdyille ratkaisuille on ollut ominaista, että ne ovat olleet *ad hoc* tyylisiä eli juuri tähän tarpeeseen. Järjestelmään on luotu prototyyppinä uusia ominaisuuksia, joita sopivasti räätälöimällä on seuraava ongelma ratkaistu. Tämä ratkaisu on ollut toimiva, kun järjestelmää on kehitetty, muutokset on voitu tehdä osin nopeastikin mutta mittakaavasta saatava hyöty on jäänyt saavuttamatta. Lisäksi useamman asiakkaan kanssa toimiessa variaatioiden määrä on kasvanut ja toteutukseen liittyvän erikoistietämyksen määrä lisääntynyt.

Mittakaavaedun saavuttamiseksi järjestelmään tarvitaan yhteinen, vakioitu perusratkaisu johon voidaan tehdä muutoksia valittujen sääntöjen puitteissa. Perusratkaisu tai tässä tapauksessa tuoterunko toimii alustana eli eräänlaisena platformina. Platformin kautta voidaan vakioida työkierron rakenne ja rajapinta erilaisten toiminnallisuuksien yhdistämiselle.

Tahtotilana on muuttaa toimintatapaa kohti valmiiksi koottua palvelukonseptia, jossa asiakkaan vaatimusten mukainen kokoonpano, konfiguraatio, on keskeisessä roolissa. Konfiguraatio voidaan muodostaa mahdollisimman tehokkaasti olemassa olevista komponenteista ilman merkittävää asiakaskohtaista räätälöintiä korostamalla tehdyn työn uudelleenkäytettävyyttä. Nykymallissa vastaavaa on haettu muuttamalla lähinnä sopivinta ratkaisua, mutta yhtä kaikki, ratkaisu on aina rakennettu yhden asiakkaan tarpeisiin.

Räätälöinneille on tilauksensa esimerkiksi tilanteissa, joissa halutaan yksilöllinen ratkaisu asiakkaan oman kehitystyön tai toimintahistorian pohjalta. Näistä on löydettävissä lukuisa joukko esimerkkejä palveluiden hinnanmuodostuksessa lisäksi ne voivat sisältää liikesalaisuuksia tai niihin rinnastettavaa tietotaitoa, jolla haetaan liiketoiminnallista etua kilpailijoihin nähden. Tai yksinkertaisimmillaan kyseessä on yhteensopivuuden aikaansaaminen vanhojen sopimusten kanssa.

Kokonaisuuden kannalta on merkittävää tietää myös mitä tulevaisuus tuo tullessaan. Ennakoinnilla voidaan jäsentää sekä omaa toimintaa että lisätä uskottavuutta. Käytännön tasolle tulevaisuudet hankkeet voidaan jalkauttaa esimerkiksi toiminnallisuuksien ajoittamisella tuleville ajanjaksoille eli tiedetään ennakolta kuinka järjestelmää voidaan laajentaa riippumattomasti, kuinka siihen lisätään toiminnallisuuksia – vakio- tai räätälöityjä elementtejä – luomatta liiallisia sidoksia komponenttien välille.

5.1. Asiakkaan toimiala

Tässä työssä tarkasteltava Fleetlogis on yleispätevä ohjelmisto kuljetusalan tuotannonohjaustarpeisiin. Vaikka järjestelmä on sovelias sekä runko- että nouto- ja jakeluliikenteen hallintaan ei samat työkalut toimi kaikissa tilanteissa ja niiden välillä on tehtävä määrittelyjä ja rajauksia. Myös kuljetusyrityksen koko ja tarpeet määräävät millainen ratkaisumalli valitaan.

Toimialan yleisimpien tarpeiden mukaan voidaan karkeasti tehdä yritysten välillä jaottelu:

- Toimintastrategian mukaan; yritys keskittyy runko- tai jakeluliikenteeseen.
- Yrityksen koon mukaan; isommat yritykset tarvitsevat omia prosessejaan tukevia työkaluja.
- Seuranta- ja analyysitarpeen mukaan; pienemmille riittää tuntikirjaukset ja auton reaaliaikaisen sijainnin selvittäminen siinä missä isot yritykset haluavat tarkastella reitin tunnuslukuja ja tapahtumia.

Mitä tämä sitten merkitsee sovelluksen kannalta? Ensimmäiseksi runko- ja jakeluliikenteen lähtökohdat ovat tilausten vastaanoton kannalta erilaiset, mahdollisuudet asiakaskohtaisiin ratkaisuihin ja valmiudet niiden ylläpitoon olennaisesti riippuvat yrityksen

resursseista sekä laajempien resurssien myötä myös seurannan tarpeet kasvavat, kun hallinnasta ovat vastuussa useammat henkilöt.

5.2. Toimintamallin mukaiset variantit

Runkoliikenteessä liikennöitsijä saa tilauksen tuotteen viemiseksi valmistajalta jakeluterminaaliin ja toimittaa sinne massatoimituksen tuotteita yhdellä lähetteellä. Jakeluliikenteessä massatoimitus kerätään usean eri vastaanottajan toimituksiksi ja lähetteiden määrä moninkertaistuu edelliseen verrattuna. Siinä missä runkokuljetukset ovat lähes kiinteästi vakioituja pisteiden A ja B välillä saattaa jakeluliikenteessä olla jatkuvasti vaihdoksia vastaanottajien kohdalla.

5.2.1. Räätälöity variantti vai perustuote?

Edellä on todettu, on yrityksen koolla varsin suuri merkitys siihen kuinka paljon tuotetta voidaan räätälöidä yrityksen tarpeisiin. Yritys joutuu siis järjestelmää määritellessään ottamaan kantaa omiin valmiuksiinsa ylläpitää järjestelmää. Vaikka järjestelmä toimitetaankin loppuasiakkaalle palveluna avaimet käteen periaatteella, vaatii järjestelmän käyttö tietoteknistä osaamista perusmäärittelyjen tekemiseen ja normaaliin ylläpitoon. Mitä enemmän järjestelmään lisätään räätälöintejä ja integrointeja muihin järjestelmiin sitä enemmän vaatimukset kasvavat. On siis oletettavaa, ettei potentiaalisessa käyttäjäkunnassa ole merkittäviä resursseja järjestelmän vaativampaan ylläpitoon.

On siis kysyttävä riittääkö asiakkaalle päävariantin mukainen perusratkaisu vai tarvitaanko räätälöidympää ratkaisua?

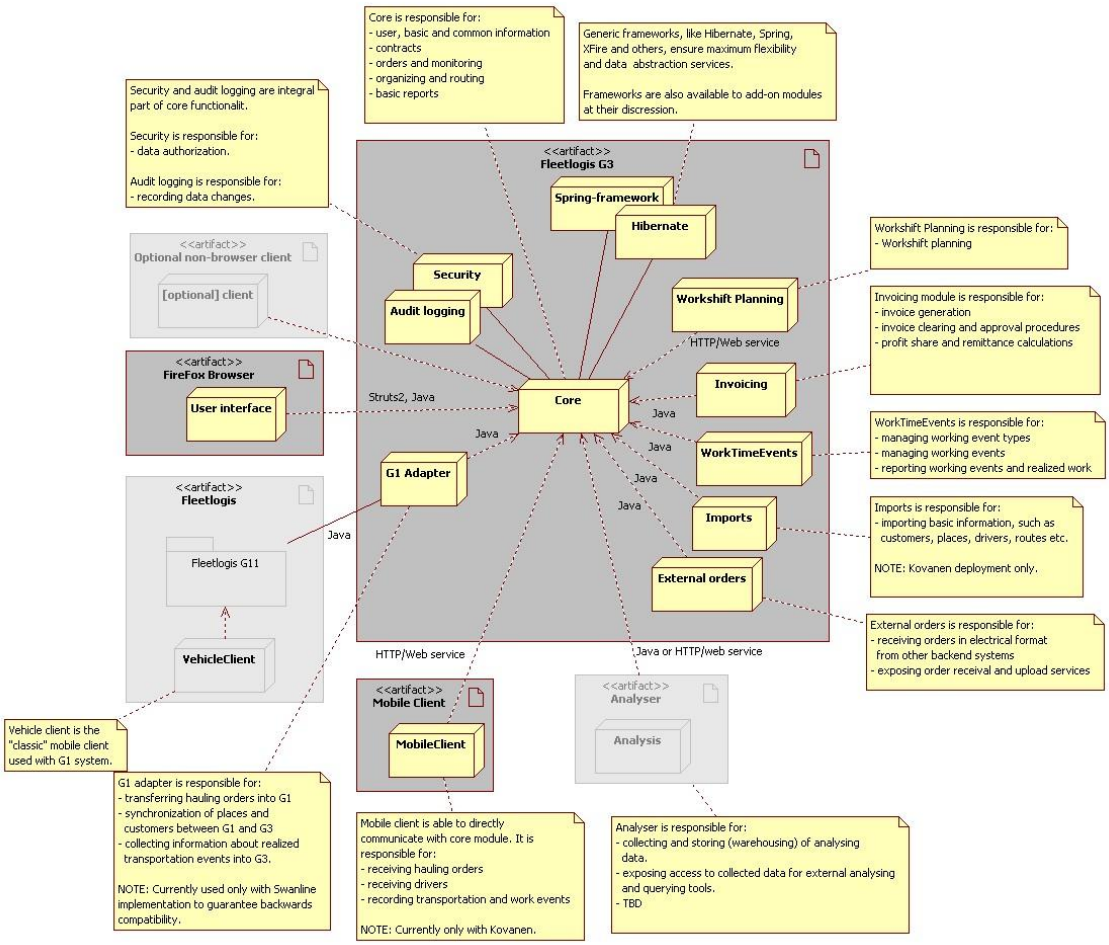
Perusratkaisuna järjestelmään toteutettiin päävariantit, joihin koottiin tärkeimmät ominaisuudet niin että ne palvelevat mahdollisimman suurta osaa käyttäjistä. Pääsääntöisesti ominaisuuksien valinta on tehty nojaten aiempaan kokemukseen järjestelmän toiminnoista ja asiakkailta saaduista palautteista.

6. Tuotteen modulaarisuuden kehityksestä

Tässä luvussa tarkastelen mitä projektilla tavoitellaan ja millaisin välinein työtä tehdään.

6.1. Tuotteen kannalta

Tuotteen kannalta tavoitetilana oli edetä tilanteeseen, jossa jokainen looginen toiminta on eriytetty omaksi moduulikseen (kuva 8).



Kuva 8. *Sovelluksen moduulirakenteesta uml deployment-kaaviona.*

Moduulit voidaan jakaa seuraaviin kategorioihin:

- toiminnalle välttämättömät,
- käyttöliittymä,
- valinnaiset moduulit,
- ajoneuvopäätteet,
- yhteydet ulkoisiin järjestelmiin ja
- räätälöidyt moduulit.

Toiminnalle välttämättömät ovat moduuleita, joita ilman järjestelmä ei pysty toimimaan. Loppukäyttäjän näkökulmasta tällaisia moduuleita ei edes ole olemassa vaan ne sisältyvät tuotteeseen. Esimerkkejä välttämättömistä moduuleista ovat ohjelmistoydin, core, ja käyttäjien ja käyttöoikeuksien hallinta, security.

Käyttöliittymä on oma kokonaisuutensa, sitä ei ole luettu toiminnalle välttämättömiin, koska sillä voi olla erilaisia toteutusvaihtoehtoja, jotka ovat toiminnalle välttämättömistä moduuleista riippumattomia. Käyttöliittymän vaihtoehtoinen toteutus on viimekädessä asiakkaan valittavissa, jos tarjolla ei ole heille sopivaa vaihtoehtoa olemassa tai se pitää integroida tiiviimmin osaksi jotain laajempaa sovelluskokonaisuutta. Nyt tehdyssä toteutuksessa käyttöliittymä perustui web-palvelimen ja -selaimen käyttöön.

Valinnaiset moduulit ovat sellaisia kokonaisuuksia, jotka voidaan liittää toiminnalle välttämättömien rinnalle niiden toimintaa kuitenkin häiritsemättä. Niiden avulla saadaan järjestelmään liitettyä lisäominaisuuksia, kuten resurssisuunnittelua tai työtapahtumaseurantaa.

Ajoneuvopäätteet ovat ajoneuvojen pääteohjelmistoja, jotka välittävät tuotantotapahtumia järjestelmästä ajoneuvoon ja päinvastoin. Tiedonsiirto tapahtuu on-line internet yhteydellä. Yleensä moduuliratkaisu sisältää sekä päätelaiteohjelmiston että palvelinpään moduulin, jolla käsiteltävä tieto sovitetaan ajoneuvopäätteelle paremmin sopivaksi.

Räätälöidyt moduulit ovat pääsääntöisesti sellaisia, joilla järjestelmää sovitetaan toimimaan muiden järjestelmien kanssa ja ne toteutetaan räätälöidysti vastaamaan asiakkaan tarpeita. Esimerkkeinä tällaisista ovat tilausten vastaanotto ulkoisesta järjestelmästä joko konekielisenä tai tiedostopohjaisena siirtona, tietoaineiston tuonti tai sovitus vanhan tuotannonohjausjärjestelmän välillä. Myös taloushallintoa koskeva, laskutus ja alihankkijatilitys, aineisto sovitetaan aina käytössä olevan järjestelmän tarpeisiin.

Taulukko 4 esittää moduulien sijoittumisen eri kategorioihin.

Taulukko 4. *Moduulikategoriat ja moduulien sijoittuminen niihin.*

Kategoria	**Moduuli**
Toiminnalle välttämätön	Core, Security, Audit logging, Spring framework, Hibernate.

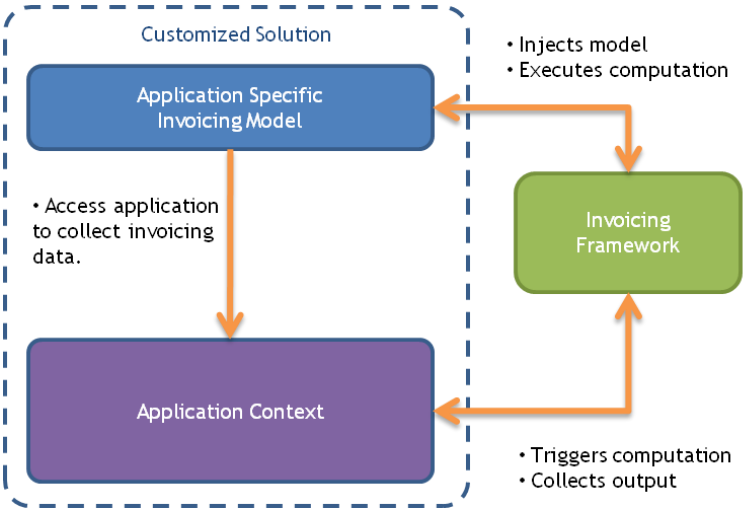
Käyttöliittymä	User interface.
Valinnaiset moduulit	Työvuorosuunnittelu, työtapahtumaseuranta.
Ajoneuvopäätteet	Mobile client.
Räätälöidyt moduulit	Laskutus- ja alihankkijatilitysaineiston vienti[1], ulkoisten tilausten vastaanotto, adapterit on-line tiedonsiirtoon eri järjestelmien välillä, tietoaineiston tuonti[2].

[1] *Aineisto kootaan eräajona siirrettäväksi taloushallinto-ohjelmistoon.*
[2] *Kertaluonteisesti käytettävä käyttöönottoon liittyvä moduuli.*

6.2. Arkkitehtuuri ja ylläpidettävyys

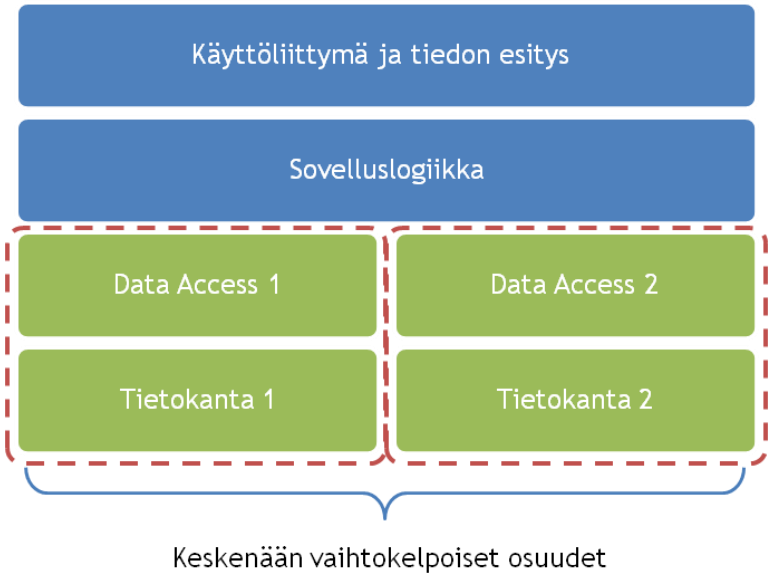
Kuten jo edellisessä luvussa (5.) todettiin, tuotteelle tarvitaan sellainen rakenne, joka kestää erilaisten asiakaskokoonpanojen muodostamisen ilman jatkuvaa perusratkaisuun muuttamista ja sallii mahdollisimman suuren uudelleenkäytön.

Niille kokonaisuuksille, moduuleille, jotka eivät välittömästi kuulu perussovellukseen käytettiin rajapintoihin perustuvaa ratkaisua. Rajapintojen avulla voidaan abstrahoida varsinainen sovellusratkaisu yhdenmukaisen liityntäpinnan taakse ja jättää varsinainen toteutus ulkoiseen osioon (kuva 9).

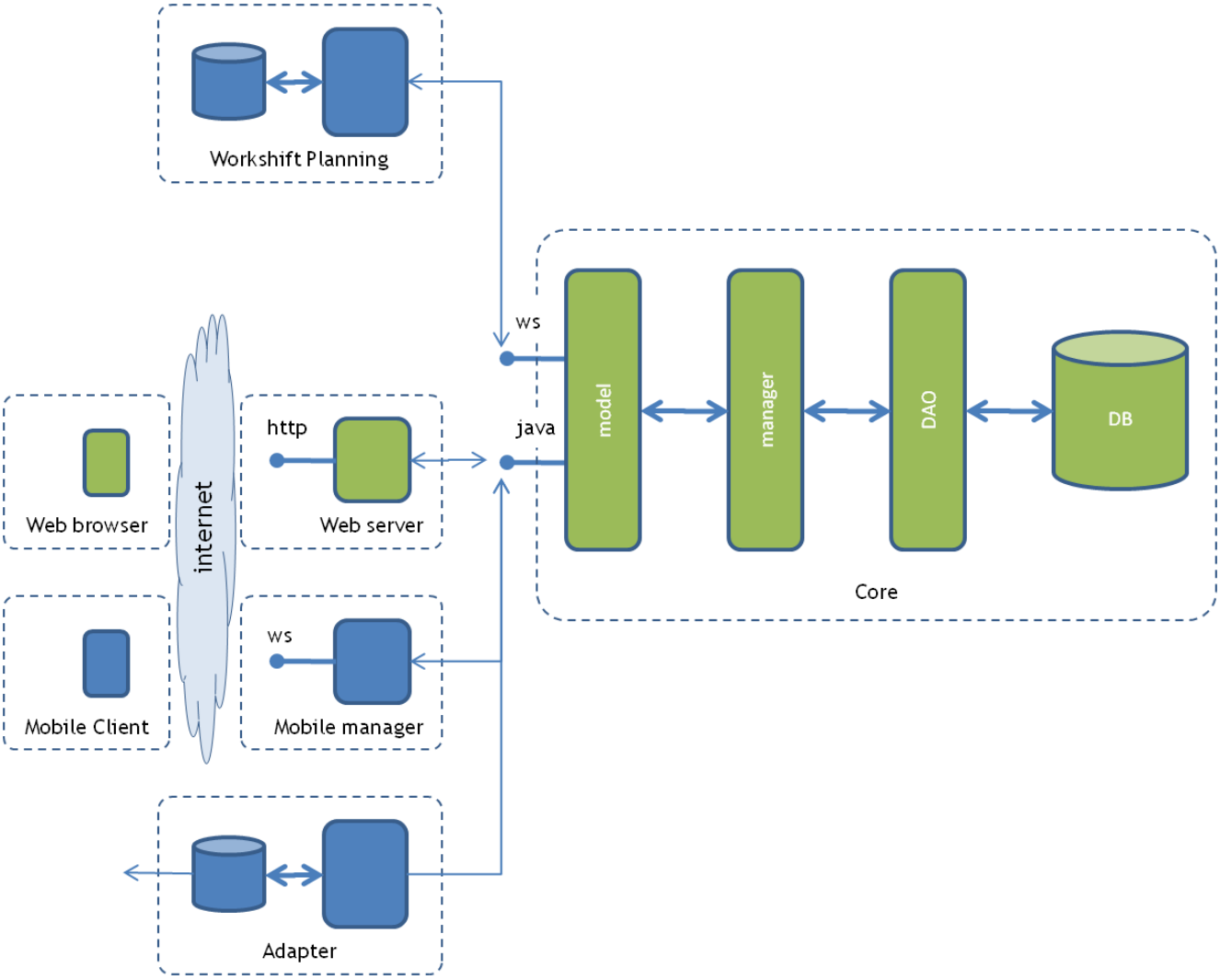


Kuva 9. *Moduulien välinen riippuvuus.*

Ohjelmistokehityksen puolelta katsottuna modulaarisuutta on voitu toteuttaa laajasti. Toolbox-ajattelun mukaisesti hyödyntämällä erilaisia olemassa olevia tunnustettuja tekniikoita, kuten *Spring* ja *Hibernate*, sekä vakioimalla eri arkkitehtuurikerrosten liityntäpinnat saadaan aikaiseksi sovellukselle luonteva arkkitehtuuri, jossa esimerkiksi käytettävä tietokantamoottori voidaan vaihtaa toiseen muuttamatta lainkaan sovelluslogiikkaa, kuten kuva 10 esittää. Kuva 11 esittää muodostunutta sovellusarkkitehtuuria.



Kuva 10. *Havainne sovellusarkkitehtuurista, jossa keskenään vaihtokelpoiset ratkaisut.*



Kuva 11. *Sovellusarkkitehtuurin keskeisimmät tasot ja moduulit.*

Seuraavaksi keskityn tarkastelemaan modulaarisuutta puhtaasti ohjelmoinnin kannalta. Varsinaisen työn toteuttamisen kannalta seuraavaksi esiteltävät teknologiat ovat työkaluja, jotka mahdollistavat erilaisia asioita. Ne ovat tässä yhteydessä nostettu esille sen vuoksi, että niiden avulla voidaan havainnollistaa ohjelmistoon tehtyä modulaarisuutta, sen etuja ja/tai haasteita. Ne esitellään seuraavaksi tämän työn kannalta keskeisimmiltä ominaisuuksiltaan, mutta niiden yksityiskohtaista käyttöä tai muita toiminnallisuuksia ei tämän työn puitteissa laajemmin käsitellä.

6.2.1. Java

Java on Sun Microsystems Inc:n kehittämä oliopohjainen alustariippumaton kehityskieli, jota käytetään laajasti erilaisissa internet-palveluissa (Sun 2009). Luonteenomaista Javalle on sen moduulijärjestelmän kaltainen rakenne, mm. rajapintojen laajamittainen

käyttö. Java on erityisen suosittu myös avoimen lähdekoodin yhteisöissä ja sillä on paljon palvelintoimintaan liittyviä sovelluksia ja ratkaisuja saatavissa.

Korkeamman tason oliopohjaisissa ohjelmointikielissä, kuten Javassakin, on useita yhteneväisyyksiä modulaarisuuden kanssa. Modulaarisuus, siinä muodossa kuin sitä tämän työn yhteydessä yleisesti käsitellään, on saanut paljon vaikutteita ohjelmistotekniikan kehityksestä.

6.2.2. Spring Framework

Spring Framework on suosittu avoimen lähdekoodin ohjelmointikehys yrityssovellusten, enterprise application, kehitystyöhön. Sen perusta on Rod Johnsonin 2002 julkaisemassa *Expert One-on-One J2EE Design and Development* –kirjassa. Keskeisin periaate on eriyttää ohjelmiston toiminnalliset kokonaisuudet ja liittää ne toisiinsa vakioidun menettelyn kautta. Eli Spring toimii tässä välittäjänä *inversion of control* -suunnitteluperiaatteen mukaan, jossa välittäjän avulla yhdenmukaisesti määritellään ja hallitaan Java objekteja *callback*-kutsujen avulla. Välittäjän velvollisuus on hallinta objektien elinkaarta: luoda, initialisoida ja määritellä objektien väliset yhteydet. (Wikipedia 2009a) Hyödyntäen vakioitua rajapintaa, josta ohjelmointitekniikassa käytetään perinteisesti nimitystä *interface*, voidaan yksittäisen toiminnallisuuden logiikkaa muuttaa vain korvaamalla kyseinen sovellusosa, Springiä käytettäessä muokkaamalla vain XML-muotoista määrittelytiedostoa ilman koko sovelluksesta uudelleen kääntämistä, *re-compilation*.

Vaikka vaihtokelpoisuus mahdollistaa esimerkiksi kokonaisen ohjelmistokokonaisuuden tai moduulin, kuten käytetyn tietokannan, vaihtamisen toiseen ovat suurimmat hyödyt siinä, että se helpottaa ohjelmiston ylläpitoa. Jos rajapinnat ovat muuttumattomat, ei muihin ohjelmisto-osiin tarvitse tehdä muutoksia. Luonnollisesti tällaista ominaisuutta voidaan hyödyntää tehtäessä toiselle asiakkaalle rinnakkaista ratkaisua toteuttaen Alexanderin (1977), Gamman ja muiden (1999) ajatuksia ongelman ratkaisemisesta aina samanlaisen peruskaavan mukaan, kuitenkin niin että saavutettu lopputulos on joka kerta erilainen.

6.2.3. Hibernate

Hibernate on tehokas, korkean suorituskyvyn omaava avoimen lähdekoodin Java-kirjasto objektimallien ja relaatiotietokantojen yhdistämiseen. Sen pääasiallisin tehtävä on Java luokkien ja tietokantataulujen sekä edellisten tietotyyppien yhdistäminen, missä ohjelmointikehys kuvaa objektiorientoituneen domain-mallin perinteiseksi tietokantatauluksi ja SQL (standardized query language) kyselyiksi. (Wikipedia 2009b, Hibernate 2009) Hibernaten keskeisin käyttötarkoitus on yhteen sovittaa näiden kahden välisiä eroavaisuuksia siten, että tietotyypit, tallennettu tieto ja sen rakenteet sekä eheys säilyvät tiedon siirtyessä sovelluksen ja tietokannan välillä.

Tiedon eheyden hallinta on tärkeä osa sovelluksen toiminnallisuutta ja käyttöä. Tiedon eheys tarkoittaa, että tieto on todenmukaista, ajantasaista vain yhden kerralla muokattavissa ja kaikki muokkaukset ovat atomisia – kaikki tai ei mitään – operaatiota. Näiden atomisten operaatioiden hallintaan käytetään tietokannan transaktiohallinnan, *transaction management*, piirteitä. Määritelmältään transaktio on tietokannan pienin suoritettava työ, jota hallinnoidaan tietokannan kannalta yhtenä operaationa. On tavanomaista, että käsiteltäessä sovelluksen yksittäistä loogista komponenttia joudutaan sen sisältämä tieto hajauttamaan useampaan erilliseen tietokantatauluun. Tällöin olemassa se vaara, että tietoa kirjoitetaan eriaikaisesti, useana erillisenä toimintona, jolloin niiden keskinäinen yhteys katoaa tietokantaa käsiteltäessä. Käyttäen transaktioita voidaan tällainen joukko operaatioita niputtaa yhdeksi, jolloin voidaan taata sen eheyden säilyminen. (Wikipedia 2009c) Eli käytännössä tämä tarkoittaa sitä, että transaktion peruuttaminen peruuttaa myös kaikki sen osavaiheet ja tietokantaan ei tallennu väärää tai vajanaista tietoa.

Koska Hibernate ei ole sidottu suoraan tietokantaan vaan se käyttää tietokantoja SQL-rajapintojen kautta on se riippumaton käytettävästä tietokannasta.

Modulaarisuuden näkökulmasta Hibernate toimii adapterina sovittaen kaksi erilaista ajatusmaailmaa yhteen. Sen välitön hyöty on siinä, että sovellusta voidaan kehittää käyttäen ohjelmointikielen standardoituja ratkaisuja, Hibernate toimii välittäjänä, mustana laatikkona, omien rajapintojensa kautta ja tietokanta on abstrahoitu pois välittömästä businesslogiikan kehityksestä.

6.2.4. Struts

Struts on Apache säätiön avoimen lähdekoodin web-sovelluskehys Java EE web sovelluksille. Se perustuu Java Servlet API:n, *application programming interface*, käyttöön luoden kehyksen model-view-controller, MVC, arkkitehtuurille. Perinteisessä weblomakkessa sovelluskoodia – hieman lähestymistavasta riippuen – ei ole voitu erotella riittävästi laajempien projektien tarpeeseen ja siitä johtuen Java koodi on sekoittunut sivun kuvauksen HTML –kieleen. (Struts 2000)

Strutsin tavoitteena on luoda selkeästi erilliset kokonaisuudet mallille, jonne sisällytetään sovellusoperaatiot, käyttöliittymä (HTML-sivujen muodostus) ja kontrolleri, joka välittää tietoa edellisten välillä. Struts antaa kontrollerin ja tukee esityskerroksen sivumallineita, joita määritellään joko XML/XSLT tai JSP pohjaisesti. Sovelluskehittäjä laatii mallin toteutuksen ja määrittelytiedoston, jolla sidotaan malli, näkymä ja kontrolleri. Kontrolleri saa pyynnöt clientilta määrittelytiedostossa asetetun toimijan, *action*, muodossa. Toimija edelleen välittää tiedon mallille ja mallilta takaisin clientille lähetettäväksi. (Struts 2000)

6.2.5. Sovelluskehyksien edut ja haasteet

Sovelluskehyksiä valittaessa joudutaan tärkeitä valintoja. Mitä tehdään itse ja mitä otetaan ulkopuolelta? Käytetäänkö kaupallista vai avoimen lähdekoodin kehystä? Mitä ovat edut ja haasteet?

Oman kehityksen puolesta puhuu aina se, että silloin saadaan täsmäratkaisu. Toisaalta vaakakupissa on se, ettei sellaista ole olemassa, sen kehitys vie aikansa ja kuinka luotettavaksi se muodostuu. Ongelmatilanteissa ratkaisu ehkä voidaan löytää ja toteuttaa nopeastikin. Valmista kehystä käytettäessä, läpi menoaikaa voidaan merkittävästi lyhentää, mutta silläkin on hintansa: se ei välttämättä ole optimaalisin ratkaisu, siihen ei löydy osaamista tai se on vaikeasti hallittavissa. Ja muutosten läpi saaminen ulkoisen tahon hallitsemaan kokonaisuuteen voi olla hidasta.

Tärkeitä kriteerejä valittaessa kaupallisen ja avoimen lähdekoodin kehyksiä on osaaminen ja verkostot. Avoimen lähdekoodin edullisuutta arvioitaessa on kiinnitettävä huomiota omaan osaamiseen, dokumentaation ja tuen tasoon, forumin aktiivisuuteen, jotka osoittavat arvonsa siinä kohdin, kun ongelmia ilmenee. Kaupallisten ratkaisujen osalta yleensä dokumentaatio ja tuki ovat yleensä hyvin järjestettyjä.

Sovelluskehysten käyttöön liittyy yleensä omat haasteensa: osaaminen, käyttöönotto, monimutkainen konfigurointi, vaikea hallittavuus ja koodin pirstoutuminen, vaikeasti debugattava. useampi samanaikainen kehys ja niiden keskinäinen riippuvuus esimerkkeinä mainittakoon.

Tehokkuuden kannalta arvioitaessa täytyy huomioida käytettävissä olevat resurssit ja realiteetit. Tämä johtaa siihen päätelmään, että käytettävien kehysten *oletetaan* olevan tuotantolaatuisia ja toiminnaltaan luetettavia. Oletus tässä yhteydessä tarkoittaa sitä, että teknologioista käytetään vain sellaisia versioita, joilla on virallisen julkaisun status ja eri kehittäjäfoorumeilla tunnustettu toimivuus.

6.3. Laadun ja testauksen kehittämisestä

Sovelluskehityksen perinteinen ongelma on testauksen vähyys – eli loppukäyttäjät testaavat järjestelmän käytössä. Järjestelmien monimutkaisuudesta johtuen asiakkaat eivät saa järjestelmää luovutettaessa valmista toimivaa järjestelmää, vaan yleensä sellaisen kokonaisuuden, joka toimii oleellisilta osiltaan sille tarkoitetussa tehtävässä. Yksi tavoite, joka tälle projektille asetettiin, oli testauksen parantaminen. Aiemmat versiot testattiin lähinnä toiminnallisesti ja yksikkö- ja automaatiotestejä ei juuri käytetty. Merkittävin parannus saatiin aikaan lisäämällä yksikkö- ja automaatiotestejä. Näiden avulla voidaan nopeasti ja yksinkertaisesti varmistua siitä, että sovelluskoodi toimii ja käyttäytyy odotetusti. Näillä menetelmillä pystyttiin varsin menestyksekkäästi varmistamaan sovelluslogiikan oikea toiminnallisuus.

Kehitystyötä tukevien sovelluskehysten käyttö edesauttaa myös testausta: ne on alun perin suunniteltu tukemaan testausta ja niihin kehitettävät ominaisuudet ja ne tukevat toiminnallisuuksien kehittämistä riittävän pieninä kokonaisuuksina, jolloin automaattisen testauksen soveltaminen on mielekästä.

Seuraavat luvut käsittelevät käytettyjä automaattitestauksen tekniikoita ja työvälineitä. Järjestelmien välisen yhteensopivuuden testaus – ns. interoperability testing, IOP – on tässä yhteydessä jätetty tarkastelun ulkopuolelle, koska modulaarisuudella on vain vähäinen vaikutus järjestelmien välillä.

6.3.1. Test-driven development

Test-driven development, (Beck 2003), on ohjelmiston kehitystekniikka, joka perustuu lyhyiden kehityskierrosten toistamiseen: ensin kehittäjä laatii automatisoidun testitapauksen, joka määrittelee halutun parannuksen tai uuden funktion, sitten tuottaa koodin, joka läpäisee testin ja lopuksi jäsentelee koodin niin että se täyttää sille asetetut vaatimukset. Tekniikan kehittäjäksi mainitun Kent Beckin toteamus TDD:n käyttämisen suosivan yksinkertaisia rakenteita ja kasvattaa luottamusta on käytännön työssä osoittautunut oikeaksi päätelmäksi. (Wikipedia 2009d, Beck 2003)

Käytännön tasolle TDD jalkautuu, kun kehitystyössä käytetään yksikkötestaukseen soveltuvaa testauskehystä, kuten JUnit-työkalua.

6.3.2. Junit

JUnit on työkalu automatisoituun yksikkö- ja kehittäjien oman, java-kielisen, koodin testaukseen (Junit 2009). Sen avulla tehdään yksinkertaisia, toistettavia testitapauksia, joilla voidaan varmentaa kehitetyn sovelluskoodin oikeata toiminnallisuutta erilaisissa tilanteissa, kuten syötteiden oikeellisuuden ja virhetilanteiden hallinnassa. Periaatteelliselta tasolta katsottuna yksikkötestauksella kuormitetaan testattavaa luokkaa sen ulkoisten rajapintojen kautta ja luokan riippuvuudet ovat minimoitu toteuttaen ne testausta tukevilla toistimilla, joilla voidaan simuloida erilaisia kuormitus- ja virhetilanteita. Mitä alhaisemmat riippuvuudet luokalla on, sitä yksinkertaisempaa sen testaaminen yleensä on: paras testattavuus saavutetaan, kun luokka tukee rajapintamäärittelyjä sekä metodien että riippuvuuksien suhteen.

Testityökalua ei sisällytetä varsinaiseen sovellukseen, mutta sitä käytetään aktiivisesti kehityksen tukena. Ketterien kehitysmenetelmien myötä testauksen luonne on muuttunut reaktiivisesta ajattelusta ennakoivampaan suuntaan. Siinä missä reaktiivinen testaus on tyytynyt toteamaan toteutuksen oikeellisuuden, ennakoivassa test driven development ajattelussa pyritään toteuttamaan testit ensin ja toteutus vasta sitten. Tämän ajattelun keskeisimmät kulmakivet ovat, että testauksen hyödyt saavutetaan heti ja testauksen tekeminen pakottaa suunnittelemaan toteutuksen mahdollisimman alhaisin riippuvuuksin, joka taas edesauttaa, tosin yksilötasoista, modulaarisuutta, kuten Beck (2003) on todennut.

6.3.3. Yksikkötestauksen edut ja haasteet

Kehitystyölle on edullisinta virheen löytäminen mahdollisimman aikaisessa vaiheessa. Kokemusperäinen havainto on, että se korreloi usein korjaamisen vaikeuden kanssa – mitä myöhemmin löydetty, sitä vaikeampi korjattava. Tätä näkökulmaa vasten ajatus testiä vastaan tehtävästä toteutuksesta on houkutteleva ja, jos sillä on vielä koodia parantava vaikutuskin alkaa se jo kuulostaa uskomattomalta. Koira on tähänkin haudattuna: Ambler (2002) toteaa 100 %:n regressiotestauksen olevan myytti. Käytännön työssä on osoittautunut samansuuntaisesti, että työskenneltäessä sellaisien osuuksien kanssa, jotka voidaan testitilanteessa täysin kontrolloida, menetelmä on kannuksensa ansainnut. Mutta esimerkiksi testattaessa sovelluksen businesslogiikkaa, testitapaukseen on valjastettava mukaan jo muita komponentteja – kuten luvuissa 6.2.2 ja 6.2.3 esitellyt Spring Framework ja Hibernate-kehykset – alkaa testitapausten luominen olla jo varsin haasteellista. Käyttöliittymän tasolle vietynä automatisoitu yksikkötestaus alkaa olla jo kohtuuton urakka. (Ambler 2002) Kysymys ei ole siitä eikö näin voitaisi tehdä vaan siitä, että siihen käytettävä työpanos ja -aika eivät vastaa siitä saatavaa hyötyä.

Käytännössä automaatiotestauksen haasteisiin voidaan vastata toisella lähestymistavalla, jossa järjestelmäkomponenttien yhteensopivuutta mitataan välillisesti käyttöliittymän kautta. Käytettävissä on sekä käyttöliittymän automaattiseen testaukseen olevia työkaluja että perinteistä käyttäjätestausta. Näitä ei kuitenkaan tässä yhteydessä tarkistella laajemmin, koska niillä ei ole kokonaisuuden kannalta suurta merkitystä.

7. Case study

Tässä luvussa tarkastelen tapauskohtaisesti kahta järjestelmään toteutettua varianttia kronologisessa järjestyksessä. Samalla käyn läpi toteutuksessa ilmenneitä haasteita ja tehtyjä valintoja. Ensiksi tarkastelen taustatilannetta ja kehityksen lähtökohtia edeten ensimmäisenä toteutettuun runkoliikenteen ja sitten jakeluliikenteen varianttiin. Lopuksi käyn läpi johtopäätökset.

7.1. Taustatilanne

Projektia aloitettaessa oli tiedossa, että järjestelmä toteutetaan kahdelle eri asiakkaalle räätälöitynä. Tämä tarkoitti järjestelmän mukauttamista asiakkaiden tarpeisiin ja järjestelmiin. Lisäksi järjestelmään lisättiin uusia toiminnallisuuksia, joiden avulla voidaan paremmin hallita mm. käyttäjäoikeuksia ja tallentaa järjestelmän hallitseman tiedon tilan muutoksia. Ei-toiminnallisena vaatimuksena oli järjestelmän testauksen ja ylläpidettävyyden kehittäminen.

Koska tiedettiin, että kyseessä olevat asiakkaat toimivat erilaisella toimintafilosofialla, joten heidän kesken oli sovittu järjestelmän mahdollisesta kehittämisestä myös yleispäteväksi logistiikan tuotannonohjausjärjestelmäksi.

Näitä lähtötietoja vasten oli houkuttelevaa luoda järjestelmään muuntelua tukevaa joustavuutta, koska toimittajan kokemukseen perustuva näkemys oli, että asiakkaat tulevat vaatimaan räätälöintejä. Minimissään nämä tarkoittavat konfigurointityylisiä asiakasparametrisointeja kuten yrityslogoa ja laajimmillaan asiakaskohtaisen sovelluksen tai järjestelmien integrointia. Muuntelua tukeva järjestelmä myös tiedetään testauksen kannalta yksinkertaisemmaksi.

Näkemys tässä vaiheessa oli, että toteutuksien välillä on samankaltaisuutta siinä määrin että muutokset ovat varsin pieniä variointeja eivätkä kysymyksessä ole eri tuotteet.

7.2. Kehityksen lähtökohdat

Tarkoituksenmukaisinta oli ennen uuden toiminnalisuuden lisäämistä tarkastaa mahdollisuus järjestelmän mahdollisuudet sopeutua muutokseen. Varsin pian kuitenkin ymmärrettiin, että ylläpidettävyyden vuoksi toteutukseen pitää tehdä merkittäviä muutoksia, jotka selkeästi rajaavat toteutuksen eri kerrokset ja luovat yhtenäisen tavan liittää ne yhteen. Lisäämällä vaatimuslistaan edellä mainitut uudet ominaisuudet oli realistisinta tut-

kia vain valmiita sovelluskehyksiä, joista kyseiset toiminnallisuudet olisivat valmiina. Tämän pohjalta tutkittiin erilaisia vaihtoehtoja, joista esille seuloutui Hibernaten ympärille kehitetyt kehykset.

Ensimmäisenä pyrkimyksenä oli edetä bottom-up -tyylisesti luoden hybridiratkaisu, jossa vanhaa sovellusta olisi vähitellen uudistettu käyttäen olemassa olevaa tietokantaa. Tämän haasteeksi muodostui vanhan version sidosteinen toteutus, missä yksittäinen muutos saattoi vaikuttaa koko järjestelmän toimintaan. Täten uuden osuuden lisääminen saattoi laukaista virheen, jonka korjaamiseen kului merkittävästi kehitysaikaa. Tämä, onneksi, havaittiin alkuvaiheessa ja arvioitaessa työmääriä kävi ilmi, että uuden toteutuksen tekeminen vie mitä todennäköisimmin vähemmän aikaa.

Lähestymistavaksi muodostui edellä mainituista syistä kunnianhimoinen top-down ratkaisu. Sovelluskehityksen kannalta vaihtoehto oli hyvä, koska historian painolasti ei rajoittaisi työtä ja perusrakenteet ja ratkaisumallit ovat olemassa. Tietokannan tauluihin kohdistuisi muutoksia uusien toiminnallisuuksien ja vaatimusten myötä. Koska koko järjestelmän toiminnallisuus nojaa vahvasti tietokannan automaattisiin oikeellisuuden varmistaviin ominaisuuksiin, esimerkiksi tietotyypit ja tiedon yhtenäisyys, oli suosiollista, ettei vanhaa järjestelmää tarvinnut ylläpitää näiden muutosten osalta.

7.3. Runkoliikennevariantti

Järjestelmään toteutettiin ensimmäisenä runkoliikennöintiä tukeva toiminnallisuus. Ajatuksena on, että tämä vastaisi tuotteen peruslinjaa ollen niin sanottu "vanilla" versio. Toteutuksessa tukeuduttiin vahvasti tuotteen historiaan ja siten keskeisimmät ratkaisut perustuivat aiempaan versioon. Tästä muodostui rajaus tuotteen perusosalle, johon sisällytettiin mm. seuraavat logistiikan toiminnallisuudet:

- yleis- ja perustietojen hallinta,
- sopimukset,
- tilaukset ja niiden seuranta,
- järjestely- ja reititystoiminnot,
- poikkeamien hallinta,
- perusraportit sekä
- näiden käyttöliittymät.

Tuotteen perusosaa, ei nähty voitavan jakaa enää pienempiin moduuleihin, koska näiden kokonaisuuksien välillä on suuret keskinäiset riippuvaisuudet. Taulukko 5 havainnollistaa esimerkinomaisesti ominaisuuksien välisiä riippuvaisuuksia.

Taulukko 5. *Perusosan toimintojen ja käsitteiden keskinäisiä riippuvaisuuksia.*

Käsite	Kuvaus, esim.	Riippuvaisuus, esim.
Perustiedot	Järjestelmässä käytettävät yksiköt.	
Yleistiedot	Käytettävissä olevat resurssit, tuotteet, paikkatiedot, reitityksset ja reitit, jne.	Resurssin kapasiteetin määrittely edellyttää yksikkötietoja.
Sopimukset	Määrittelee reitin, ajopäivät ja kuljetustarpeet.	Sopimusreitillä on kuljetustarve: lähtö- ja päätepisteen aikaikkunoineen sekä kuljetettava(t) tuotteen(t).
Tilaus	Tilaus määrittelee ajankohdan ja kuljetustarpeen.	Tilaus perustuu sopimukseen ja noudattaa sen rakenteita.
Ajojärjestely ja reititys	Reitille määritellään kuljetusresurssi ja kuljetustarpeet.	Reititys perustuu sopimusreitteihin ja kuljetustarpeet tilauksiin.
Poikkeamien hallinta	Poikkeamatietojen keräys jälkikäsittelyä varten.	Riippuvaisuuksia sopimuksiin, tilauksiin, reitteihin...
Raportit	Tallennettua tietoa erilaisista tuotantotapahtumista.	Reitit, kuljetustarpeet, tilaukset...

Lisäksi perustoiminnallisuuksiin sisältyvät käytettyjen sovelluskehittimien tuomat uudet ominaisuudet:

- tapahtumahistoria,
- käyttäjätietojen hallinta sekä
- näihin sisältyvä maksimaalinen joustavuus ja tiedon abstraktio.

Käyttöliittymä on oleellinen osa sovellusta. Luonnollisesti siinä on tietorakenteita vastaavat näkymät ja siten sidoksissa käytettävissä olevaan tietoon. Se on tässä yhteydessä eroteltuna muista, koska se on suunniteltu tukemaan pienimuotoista, konfigurointiperusteista variointia, jossa varsinaista toiminallisuutta ei muuteta mutta käyttöliittymässä voidaan esittää asiakkaan logo ja komponenttien ulkoasua voidaan hallinnoida erikseen. Mutta, se on myös yksi mahdollinen moduuli, sillä järjestelmässä on myös varauduttu siihen, että käyttöliittymä voidaan tehdä myös muuna kuin web-selainpohjaisena clienttina.

Vaikka runkoliikennetoteutus tehtiinkin asiakkaan tarpeen mukaan, lisättiin ajoneuvopääteyhteys järjestelmään erillisenä moduulina. Perusteena tälle oli vaatimus, että vanhan järjestelmän kanssa yhteensopivia ajoneuvopäätteitä ei päivitetä, ja siten tiedonsiirto piti toteuttaa vanhan ja uuden järjestelmän välillä kahdensuuntaisena. Koska tieto-

rakenteissa oli merkittäviäkin eroavaisuuksia, suoran yhteyden tekeminen tietokantojen välillä olisi luonut näiden välille ratkaisun, joka ei ole yleistettävissä muille vastaaville asiakkaille. Tämä ratkaistiin toteuttamalla adapterimoduuli, joka vastaa perusosan ulkopuolisena komponenttina tiedon siirtämisen järjestelmien tietokantojen välillä. Ja yleisellä tasolla ratkaisumalli on sama kuin jakeluliikenteen ratkaisussa, jota tarkastellaan jäljempänä.

7.4. Jakeluliikenteen variantti

Jakeluliikenteen varianttia toteutettaessa lähtökohtana oli edetä pienin muutoksin runkoliikenteen vanilla versiosta. Jakeluliikenne on uusi kokonaisuus. Vaikka asiakkaalta saatiin monipuolinen ja kattava tuki erilaisten ongelmien selvittämiseen, jakeluliikenteen toteutus kärsi alhaisesta maturiteetista. Esimerkiksi mitä pidemmälle selvitystyössä edettiin, sitä selkeämmäksi kävi aiempien ratkaisujen sopimattomuus. Eroavaisuuksia löytyi mm. seuraavista kokonaisuuksista:

- sopimusrakenteiden keskeiset käsitteet,
- reitin ja lenkin eroavaisuudet,
- käyttötapausten erilaisuus ja
- käyttöliittymien toiminnallisuus.

Talletettavan tiedon muoto ja rakenne on merkittävästi erilainen runko- ja sopimusliikenteen ratkaisujen välillä. Sopimukset olivat sekä rakenteeltaan että rajauksiltaan erilaisia: kun runkoliikenteessä sopimus on aina reitille, jolla on selkeä lähtö- ja päätepiste, jakeluliikenteessä määritellään esimerkiksi postinumeroihin perustuva alue, muttei päätepistettä. Edelleen runkoliikenteessä kuljetustarve on sidottu reittiin ja jakeluliikenteessä sopimukseen. Vastaavia tapauksia voidaan osoittaa suuri määrä.

Kun tähän asti modulaarisuuden lisääminen oli perustunut sovelluskehittimien tarjoamiin ratkaisuihin, jotka perustuvat staattisille tietokantarakenteille, seurasi rakenteiden muuttumisesta iso ongelma kuinka hallita varioituvat tietorakenteet. Vaihtoehtoisina ratkaisumalleina nähtiin kenttien tai rajapintojen yleistäminen tai variantti, joilla kaikilla on omat huonot puolensa: Kenttien yleistäminen olisi edellyttänyt tietotyyppitarkistusten vähentämistä, jota lähtökohtaisesti ei toivottu. Rajapintojen yleistäminen olisi tuonut suuren joukon uusia metodeja ohjelmointirajapintoihin sekä tarpeettomia kenttiä tietokantatauluihin, joilla myös olisi saattanut olla merkitystä tietokannan automaattitarkistuksiin. Erillinen variantti vastaavasti pitää metodien ja tietokantataulujen kenttien lukumäärän kohtuullisena, mutta pakottaa laajoihin muutoksiin läpi eri sovelluskerrosten.

Arvioitaessa edellä mainittuja vaihtoehtoja, erillisen variantin muodostaminen nähtiin parhaimpana vaihtoehtona, koska kyseessä olisi eri tuote. Tätä edesauttoi asiakkaan vahva näkemys siitä millaisia ovat erilaiset käyttötapaukset ja niitä tukevat käyttöliittymät, jotka olivat runkoliikenneversiosta selvästi poikkeavat. Lisäksi muut ratkaisut olisivat luoneet lisäarvotonta varioitumista.

Asiakkaan toimitukseen sisältyivät lisäksi seuraavat lisämoduulit:

- laskutus,
- työvuorosuunnittelu,
- työaikatapahtumat,
- tilausten vastaanotto eräajona ja
- ajoneuvopääteohjelmisto.

Laskutusta lukuun ottamatta nämä moduulit voitiin toteuttaa ilman muutoksia perustoiminnallisuuteen, joka on tässä sama kuin runkoliikennevariantin yhteydessä mainittu. Myös laskutuksen toteutus kärsi alhaisesta maturiteetista, joka ilmeni riippuvaisuuksina eri toiminnallisuuksiin. Pääsääntöisesti riippuvuudet pystyttiin pitämään yksisuuntaisena (laskutus→perustuote), mutta paikoin integroitua ratkaisua ei voitu välttää vaan riippuvuuksia syntyi myös lisämoduuleihin. Integroitu ratkaisu hyväksyttiin tässä yhteydessä mahdollisena ratkaisuna sen yksinkertaistavan luonteen vuoksi.

7.5. Johtopäätökset

Tarkasteltaessa tehtyjä toteutuksia käy ilmi, että runko- ja jakeluliikenteen ratkaisut eroavat toisistaan niin paljon, että niitä voidaan pitää erillisinä tuotteina, variantteina.

Projektin saavutuksiin voidaan lukea runkoliikennevariantin rakenteellisesti selkeä, ylläpidettävä rakenne ja core-osuuden toiminnallisuuksien kattava yksikkötestaus. Modulaarisesti tarkastellen tämä ratkaisu muistuttaa eniten moduulijärjestelmää. Erityisesti rakenteellisen hajauttamisen avulla coren oikean toiminnan todentaminen yksinkertaistui sillä tavoin merkittävästi, että kehityksen pystyi toteuttamaan yksi kehittäjä melkein kokonaisuudessaan. Tämän mahdollisti virheenetsintään ja korjauksiin käytetyn ajan vähentyminen. Edut olivat suurimmillaan, kun testattava toiminnallisuus on yksinkertainen perustoiminto. Korkeamman tason sovelluslogiikan kanssa tapausten monimutkaisuus, esimerkiksi tarvittavien esitietojen asettaminen, alkoi muodostua rajoittavaksi tekijäksi.

Samansuuntaisiin ongelmiin törmättiin myös käyttöliittymän kehityksessä, koska käyttöliittymässä testitapausten monimuotoisuus ja -mutkaisuus sekä heterogeeninen kehitysympäristö vaikeuttivat merkittävästi automaattitestausta. Core-osuudessa testiympäristö oli homogeeninen ja jokainen testitapaus selkeästi rajattu.

Spring-frameworkin XML-konfiguraatiotiedostoihin perustuva määrittely sallii tehokkaan muutoksen tekemisen rinnakkaisiin versioihin, joko periyttämällä olemassa olevaa ratkaisua tai tekemällä täysin uuden. Hibernate-frameworkin avulla voidaan tietokantamoottoria vaihtaa, jos esimerkiksi järjestelmän suorituskykyä halutaan parantaa.

Jakeluliikennevariantilla voidaan katsoa saavutettavan samat edut kuin runkoliikennevariantillakin puhtaasti sovelluskehityksen kannalta katsoen. Moduloinnin kannalta ongelmaksi muodostui keskeisten tietokantarakenteiden eroavuus eri varianttien välillä.

Sovelluskehykset, joita tässä yhteydessä käytettiin, eivät ole optimoituja tietorakenteiden muutoksille. Tätä haastetta ei osattu huomioida riittävän aikaisessa vaiheessa. Kerätyn kokemuksen mukaan on syytä arvioida uudelleen kuinka ja minkä suhteen modulointia halutaan jatkossa tehdä. Nyt käytetty toimintoperustainen lähestymistapa lienee toimiva, jos variointi tehdään jatkossa vain runkoliikenne- tai jakeluliikennevarianttien välillä. Se ei kuitenkaan ole tehokkaasti sovellettavissa runko- ja jakeluliikenteen väliseen variointiin.

Jatkon kannalta on mielekkäämpää jatkaa tarkastelua siitä näkemyksestä, että kyseessä on samaan platformiin perustuva tuoteperhe, jossa platformi perustuu geneeristen sovelluskehyksien käytölle. Ratkaisut ovat samaan viitearkkitehtuuriin perustuvia sovelluksia modulaarisesta arkkitehtuurista.

Tuoteratkaisut perustuvat valittuihin strategioihin (A ja B) ja asiakasvariantit (Asiakas Y-Z ja N) realisoituvat muutosdeltoina ($\Delta_{Y,\,Z\,ja\,N}$) vanilla (X ja M) eli perusversioon. Arkkitehtuuria on havainnollistettu taulukossa 4.

Taulukko 6. *Platformiin perustuva tuoteperhe.*

Asiakasvariantti	Asiakas X (vanilla)	*Asiakas Y* (Δ_Y)	*Asiakas Z* (Δ_Z)	Asiakas M (vanilla)	*Asiakas N* (Δ_N)
Tuoteperhe	Runkoliikenneratkaisu (Strategia A)			Jakeluliikenneratkaisu (Strategia B)	
Platform	työkierto + sovelluskehykset				

Tällä yleistyksellä platformia voidaan käyttää myös muiden tuotteiden kehittämiseen.

Miksi lopputulos eroa alkuperäisestä näkemyksestä? Modulaarisuuden kannalta keskeisin haaste oli tuotteen epäkypsyys moduloinnille. Vain toisesta tuotteesta oli kokemusta ja järjestelmää ei ollut valmisteltu ennakolta modulointiin sopivaksi.

Tuotekokemuksen tuoma ero oli selkeä, sillä historian tuoma kokemus välittyi runkoliikenteen osalta valmiina ratkaisumalleina ja rakenteina, kun jakeluliikenteelle nämä oli luotava tyhjästä. Koska tieto puuttui ensimmäistä toteutusta tehtäessä, heijastui tämä toisessa työläinä ongelmina ja ratkaisuna, jotka ei täysin tyydytä ensimmäistä tavoitetta eli varioituvuuden parantamista ensimmäisten toteutusten välillä.

Puuttuva tuotekokemus ei ole periaatteellisella tasolla este moduloinnille. Kun platform on kehitetty (valmisteltu) modulointiin sopivaksi, rajapinnat ohjaavat moduulin kehitystä järjestelmään sopivaksi. Tässä yhteydessä rajapintojen kehitykseen olisi kuitenkin tarvittu toisen tuotteen tuomaa kokemusta.

Lopputulos vahvistaa Nielsenin (2009, s. 38) vaatimusta siitä, että kehitykseen liittyy kaksi erillistä vaihetta: valmistava ja suorittava. Valmistavat toimenpiteet, preparatory activities, ovat platformin kehitystä ja suorittavat toimenpiteet, execution activities, tuotekehitystä. Loogisesti valmistavat toimenpiteet pitää tehdä ennen suorittavia (Nielsen 2009). Kuten edellä on välillisesti esitetty, tämä ei tuotekehityksessä ole aina mahdollista.

Tilanteissa, jossa jouduttiin turvautumaan integroituun ratkaisuun, on tarpeen arvioida uudelleen moduulirakennetta ja/tai sijoittaa vaikeasti moduloitavia toimintoja integroituun perustoiminnallisuuteen.

Jatkossa on syytä kiinnittää huomioita niihin ratkaisuihin, joilla voidaan lisätä tuotteiden yhteisiä ominaisuuksia. Tällaisena voidaan mainita esimerkiksi työkierron hallinnan tukeminen ja yleisesti sellaiset ominaisuudet, jotka perustuvat enemmän tilan kuin tiedon muutokseen. Toisena merkittävänä kehitysalueena näen käyttöliittymän ja soveltuvuuden kehittämisen muunteluun sopivaksi myös vaihtoehtoisten ratkaisujen osalta. Kuvaavaa käyttöliittymän haastavuudelle on se, että erittäin suurin osa kehitystyöstä kului juuri sen tekemiseen. Kehitystyön resursoinnista noin 20 % käytettiin tietokannan ja toimintojen server-puolen kehittämiseen, loppuaika kului käyttöliittymän ja sen toiminnallisuuden kehittämiseen. Syitä tälle on useita: käyttöliittymään sisältyi useita haastavia toiminnallisuuksia, web-tekniikka on vaatimuksiin nähden osin epäkypsä teknologia ja tehokkaiden työkalujen vähyys.

8. Tulokset ja havainnot

Tässä luvussa käsittelen työssä tehdyt tärkeimmät tulokset ja havainnot.

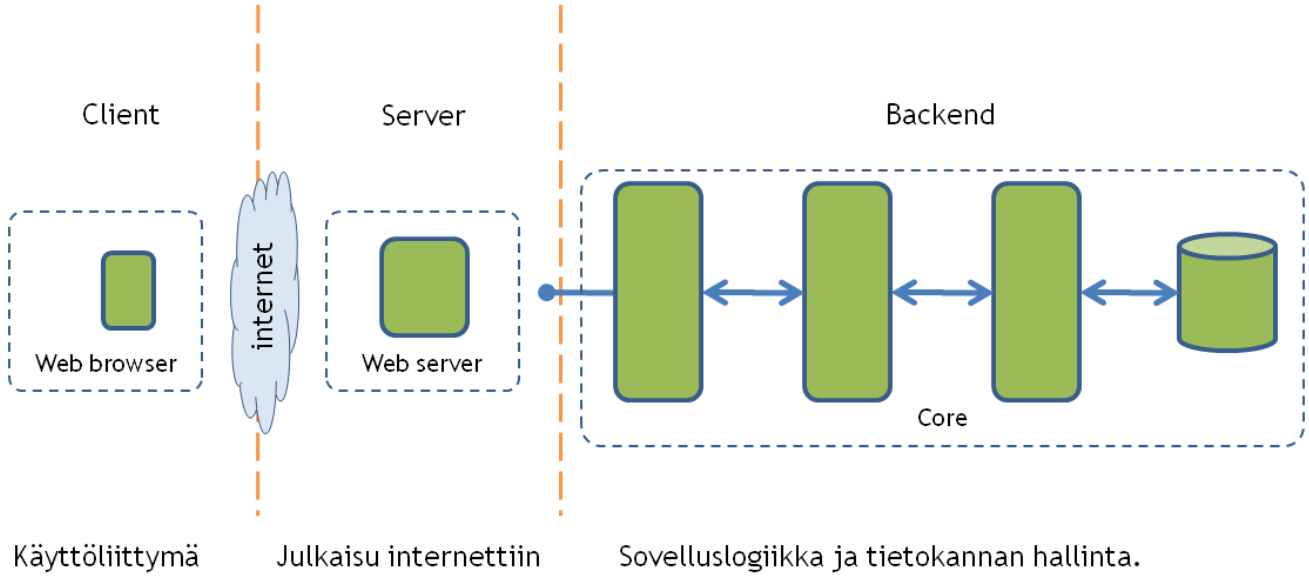
8.1. Visiosta ja toteutuksen haasteista

Kirkasotsainen visio tavoiteltavista ominaisuuksistaan on lyhykäisyydessään ytimekäs – sovellusta voidaan nopeasti muokata kulloisenkin tarpeen mukaan: muutoksille joustava, helposti laajennettava modulaarinen tuote, joka on ylläpidettävä ja selkeä rakenteeltaan (kuva 12).

Kuva 12. *Visio.*

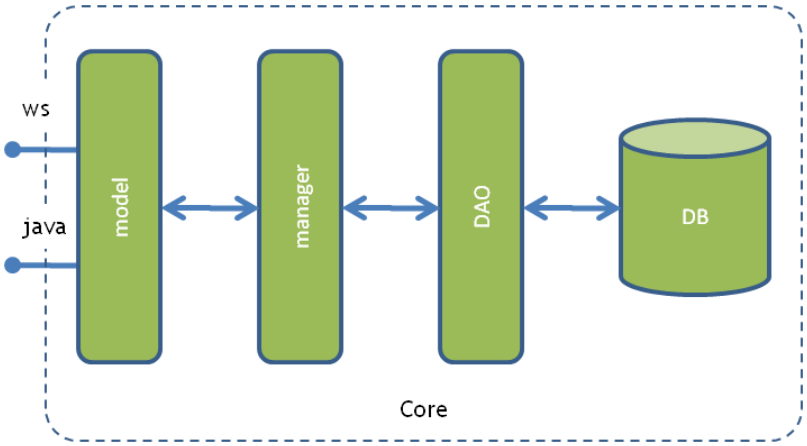
Keskeinen ohjuri kehityksessä oli arkkitehtuuri, joka olisi ylläpidettävä mutta sallisi uusien ominaisuuksien joustavan lisäämisen. Tästä käytännöllisestä lähestymistavasta johtuen modulaarisuuden lisääminen ei siten ollut päätavoite vaan ikään kuin sivutuote muusta kehitystyöstä.

Arkkitehtuuriset muutokset pyrkivät selkeisiin kokonaisuuksiin: käyttöliittymän client, internet-julkaisun web server ja sovellustason toteutuksen backend (kuva 13). Näissä korostuvat eri toiminnallisten osuuksien vaatimukset ja joilla olisi alhaisin mahdollinen määrä keskinäisiä sidoksia: tietokantayhteys, sovelluslogiikka ja käyttöliittymä. Käytettyjen sovelluskehysten avulla tässä kohdin onnistuttiin verrattain hyvin. Kokonaisuudet voidaan liittää toisistaan riippumatta määrittelytiedostojen avulla.



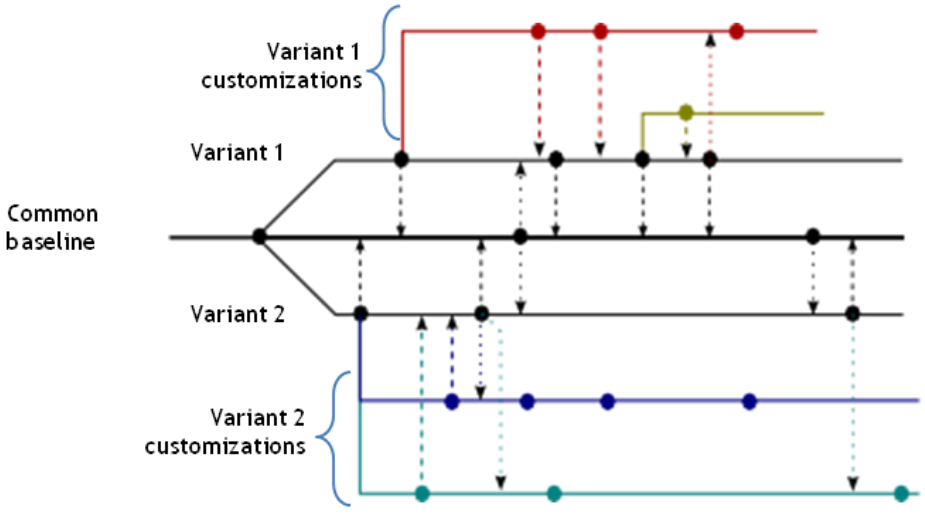
Kuva 13. *Arkkitehtuurilliset kokonaisuudet.*

Rakenteellinen kokonaisuus on selkeä. Kuitenkin sisäiseen toteutukseen liittyy piilevä haitta tiedon pirstoutumisesta. Käytettyjen sovelluskehittimien ja –kehysten avulla rakenteeseen muodostui vakioituja, yhdenmukaisia ratkaisuja ja erilaisia toiminnallisia kerroksia (kuva 14). Kerroksittainen lähestyminen ohjaa toteutuksen siten, että kukin kerros keskittyy pienimpään mahdolliseen tehtäväänsä. Koodin selkeyden kannalta tämä on erinomainen asia, josta maksettava hinta on se, että pirstoutumisen hahmottaminen on kehittäjälle vaativaa. Tehokas tekeminen edellyttää kohtalaisen hyvää ymmärrystä käytettävistä tekniikoista ja niihin liittyvistä oletuksista, toiminnallisuuksista sekä asetuksista ja niiden konfiguroinneista. Edelleen tämä nostaa vaatimustasoa myös sen suhteen, että ymmärretään valitun teknologian kokonaisuudelle aiheuttama impakti ja soveltuvuus modulaarisuuteen.



Kuva 14. *Ydinosan arkkitehtuurikerrokset: model on sovelluslogiikan, manager kontekstin hallinnan ja DAO dataobjektien tietokantakuvauksen kerros.*

Juuri kehitystyökalujen soveltamisessa modulaarisuuteen koettiin yksi suurimmista haasteista. Tavoitteena oli luoda sovellukselle alusta – *common baseline*, eräänlainen platform – johon kaikki tärkeimmät muutokset olisi toteutettu ja asiakasvariantin toteutuksessa näitä ominaisuuksia sitten tarpeen mukaan deltoina aktivoitu ja deaktivoitu hyödyntämällä ohjelmointikielen periyttämiskäytäntöjä. Tästä jouduttiin kuitenkin luopumaan, koska sovellusrakenteen manager tasolla ja käyttöliittymän integroitumisesta johtuen löytyi kohtia, joiden yhteensovittaminen ei käytettävissä olleen ajanpuitteissa ollut yksinkertaisesti mahdollista ja ratkaisuna jouduttiin käyttämään lähdekoodien haaroitusta. Ratkaisun suurin ongelma piilee ylläpidettävyydessä ja ylläpitoprosessin hoidossa eli kuinka välitetään muutokset oikein ja oikea aikaisesti kaikkiin koodihaaroihin (kuva 15). Tältä osin alkuperäistä ongelmaa ei voitu ratkaista.



Kuva 15. *Muutoksen palautus muihin kehityslinjoihin.*

Työssä edettiin toteuttamalla ensin runkoliikenne- ja sitten jakeluliikennevarianttia. Jälkimmäistä toteuttaessa realisoitui ongelma, jossa tallennettavan tiedon erilaisuus haastoi modulaarisuuden. Käytetty Hibernate ohjelmointikehys on optimoitu abstrahoimaan tietokanta sitä käyttäviltä objekteilta, mutta sitä ei ole optimoitu tietorakenteen muutoksille. Käytännössä tämä tarkoitti interface-muutoksia.

Uuden varianttikohtaisen kentän lisääminen tietokantatauluun aiheuttaa pitkän ketjun vyöryviä muutoksia: tietokantataulu, DAO objekti, DAO objektin interfacen muutos, business-logiikka, käyttöliittymä... Erityisesti muutokset, jotka kohdistuvat – myös – vain välittäjinä toimiviin kerroksiin ovat tässä yhteydessä haitallisia. Lisäksi työn edetessä kävi ilmi, että näitä kenttäkohtaisia muutoksia olisi useita.

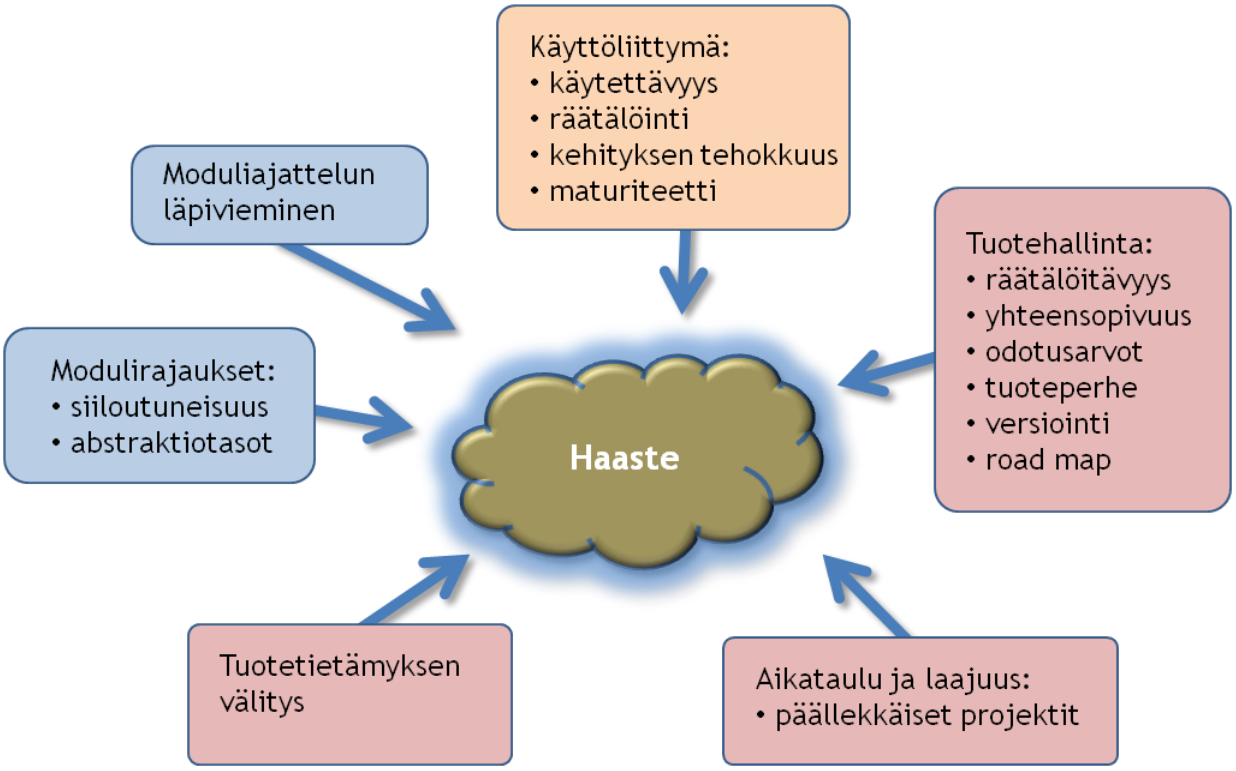
Mahdollisina vaihtoehtoina tässä nähtiin objektien periyttäminen tai geneerisen tietoblokin käyttäminen. Objektien periyttäminen olisi tuonut rinnakkaisia parametreja, jotka ovat osin päällekkäisiä, pahimmillaan tietotyyppiä muuttavia tai poissulkevia. Geneerisen tietoblokin käyttäminen vastaavasti heikentäisi tietokannan automaattisten tyyppitarkastusten hyödyntämistä.

Tehokkaan ratkaisun löytäminen edellyttää objektimallin generalisaatiota. Tähän ei työssä kuitenkaan ryhdytty, koska käytettävissä oleva aika oli rajallinen ja siihen olisi tarvittu merkittävä resursointi ja valmiiden rakenteiden muutostyö. Työmäärä arvioitiin sellaiseksi, että varianttikohtainen integroitu ratkaisu tulee nopeammaksi.

Lopputulos oli, että muodostui kaksi erillistä tuotetta, jotka perustuvat samaan platformiin: erilliset versiot runko- ja jakeluliikenteeseen.

Työn toteuttamisen kannalta keskeisimmät haasteet voidaan tiivistää kolmeen pääryhmään (kuva 16):

- tuotehallinta ja projektointi,
- modulointi ja
- toteutus.



Kuva 16. *Työn keskeisimpiä haasteita.*

8.1.1. Tuotehallinta ja projektointi

Tuotehallinnan keskeisin haaste oli saada kaikille työhön osallistuneille riittävä ymmärrys sovellusalueesta ja sen erikoispiirteistä. Lisäksi kokonaan uuden konseptin, jakeluliikenteen, lisääminen sovellukseen räätälöintinä oli haastava.

Tuotteeseen liittyy suuret odotusarvot sen mahdollisuuksista. Koska tuotteen historia on ollut pitkälti asiakastarpeen sanelemaa, suunnitelmallinen ja systemaattinen toimintojen mukaan tuominen road mapin avulla on ollut lähes olematonta. Näiden seikkojen vallitessa moduulirajojen löytäminen tarkoituksenmukaisella tavalla on ollut vaikeata. Moduulirajaukset ovat jääneet luonteeltaan yleisluontoisiksi, kuten laskutus, koska tulevaisuuden kehityssuunnat ovat hämärän peitossa. Yleisluontoisuus on myös saattanut estänyt havaitsemasta niitä kaikkia ohjureita ja rajauksia, joita tässä yhteydessä olisi ollut tarpeen havaita.

Projektointi on tässä liitetty tuotehallinnan yhteyteen koska pääsääntöisesti tuotehallinta määrittelee mitä toteutetaan. Muutokset ja kehitys toteutetaan projekteina, niillä on tavoite, aikataulu ja budjetti (Repo 2006). Ja tästä siten määräytyy se aika mikä on käytettävissä suunnittelulle ja toteutukselle. Kun kyse on projektista, on aikataulupaine yleensä kova ja aikaa suunnittelulle on rajatusti; ratkaisu on löydyttävä annetussa ajassa.

Aika tai paremminkin sen puute ohjasi ratkaisuihin, joissa paras ratkaisu oli usein hyvän ratkaisun vihollinen: valinta piti tehdä toteutuksen ja teorian väliltä. Kaikkia ratkaisuja ei siis pystytty täysin toteuttamaan luomatta riippuvaisuuksia osioiden välille. Kaikkein keskeisimmin tämä tuli vastaan juuri laskutuksen ja tilityksen osalla, koska niiden suorittamiseen tarvitaan tietoa järjestelmän kaikista osioista.

Johtuen tuotteen historiasta kehitystyö on pääsääntöisesti ollut runkoliikennekeskeistä. Jakeluliikenteen mukaan tuominen on uusi ilmiö ja osittain juuri tästä vähäisestä kokemuksesta tuotteen maturiteettitaso ominaisuuksien ymmärtämiseen on alhainen. Verrattaessa tilannetta runkoliikennettä ja jakeluliikennettä tehtyjen ratkaisujen välillä, runkoliikenteen korkeampi maturiteetti näkyi käytännössä siinä, että runkoliikennettä varten ratkaisut olivat olemassa, kun jakeluliikennettä varten ne piti erikseen kehittää.

Tuotenäkemystä rajoitti myös se tosiasia, että suurin osa kehittäjistä ei ollut työskennellyt logistiikan parissa: siten omakohtaisen kokemuksen tuoma objektiivisyys ja kriittisyys ei välttämättä haastanut kaikkia osia parhaalla mahdollisella tavalla. Tämä heijastui myös projektityöskentelyyn, sillä kirjoittajan näkökulmasta projektityössä ei ollut riittävän laajaa analyyttistä keskustelua ideoiden jalostamiseksi ja haastamiseksi.

8.1.2. Modulointi

Systemaattinen modulointi edellyttää koko tuotetietämyksen hyödyntämistä. Yhden näkökulman, esimerkiksi sovelluskehittäjän, läpi katsominen ei tuota riittävää käsitystä moduloinnin tasoista vaan mitä laaja-alaisemman tiedon valossa sitä voidaan tehdä sitä paremmaksi voidaan olettaa lopputulosta. Lisäksi on syytä kartoittaa eri osioiden välisiä riippuvaisuuksia systemaattisesti lähtien liikkeelle perusteista esimerkiksi ominaisuusmatriisin avulla.

Erityisesti moduloinnin lähtöpistettä tulee tarkastella kriittisesti. Nykyinen toteutus perustuu toimintopohjaiseen näkemykseen ja varioinnin kehitystä on eniten ajanut sovelluskehityksen tarpeet. Tuotehallinnan puolelta moduloinnin ohjaus oli vähäisempää ja ehkä siitä syystä kehitys keskittyi toiminnallisiin moduuleihin, kuten laskutus ja tilitys. Toiminnalliset moduulit ovat tuoneet mukanaan toteutuksen siiloutumista. Siiloutumisella tarkoitan sovellusarkkitehtuurin kerroksien läpi porautuvaa integroitua toteutusta muodostaen moduloinnin kannalta epäedullisia riippuvuuksia.

Edelleen moduloinnissa tulee kaikkien osapuolien sisäistää tarvittava ajatusmalli ja omaksua sitä tukevat työskentelytavat. Se käytännössä tarkoittaa sitä, että joissakin yhteyksissä abstraktiota on nostettava yleisemmälle tasolle, jotta tavoite saavutetaan. Ja tämä sillä tasolla, että sitä voidaan rutiinilla soveltaa myös epäkiitollisissa olosuhteissa.

Välttämättä tällaisessa projektissa ei saavuteta merkittävää mittakaavaetua, ainakaan ensimmäisten asiakkaiden kanssa vaan tämä investointi alkaa maksaa itseään takaisin noin 5-10 räätälöinnin jälkeen.

8.1.3. Toteutus

Myös toteutukseen käytettävien välineiden on sekä omalta käytettävyydeltään, toimivuudeltaan ja soveltuvuudeltaan sovittava käyttötarkoitukseen. Työssä käytetty webselainpohjainen tekniikka sopii mainiosti hajautettuihin järjestelmiin, joissa huippuluokan käytettävyys ei ole ensisijaista. Kun kuitenkin tavoitteena oli tuottaa järjestelmä kiireen kanssa jatkuvasti painiville käyttäjille, korostuu käytettävyys kaikilta osiltaan: käytön helppoutena, visuaalisesti selkeänä ja toiminnaltaan varmana ja nopeana, on selainpohjainen teknologia äärirajoillaan. Selainpohjaisiin ratkaisuihin yleensä vielä liittyy riippuvaisuuksia käytettyyn selaimeen, on jatkoa ajatellen syytä kiinnittää huomiota siihen millä tekniikalla käyttöliittymät toteutetaan.

Toteutuksen suurimmaksi haasteeksi muodostui käyttöliittymä. Projektiryhmässä käyttöliittymästä työstettiin projektin aikana useita erilaisia, innovatiivisiakin, vaihtoehtoja. Ongelmaksi muodostui kuinka toteuttaa graafisesti haastava käyttöliittymä webselaimissa. Struts tekniikka perustuu HTML:n ja javascriptien käyttöön. Kehittäjien kannalta suurimmat haitat olivat riippuvuus kirjastoista ja niiden versioista sekä kehittyneiden debug-työkalujen puute.

9. Jatkokehitys

Työn aikana selkiytyi uusia vaatimuksia ja tarpeita, joita on syytä arvioida jatkokehitystä määriteltäessä.

9.1. Modulaarisuuden kehittäminen

Tehty ratkaisu pyrki luomaan tuotteelle modulaarisen rakenteen. Teknisessä mielessä tämä onnistui varsin hyvin, sillä tuotteen rakenne selkeytyi ja toimintojen välille pystyttiin laatimaan rajapinnat, joiden puitteissa ne toimivat. Näkisin kuitenkin, että tässä on vasta tehty ensimmäinen vaihe eli luotu perusta, *enabler*, todellisen modulaarisuuden kehittämiselle.

Seuraava askel eteenpäin on kokonaisnäkemyksen kehittäminen läpi toimittajan organisaation, jotta moduulien toiminnoista ja rajauksista voidaan mielekkäästi keskustella ja löytää ne elementit, joilla moduulirakennetta voidaan edelleen kehittää.

Ensimmäisen vaiheen ratkaisuissa moduulit olivat hyvin siiloutuineita funktionsa mukaan ja loivat modulaarisuutta rajoittavia integroituneita vertikaaleja läpi sovelluskerrosten, joista esimerkkinä voidaan mainita laskutus. Sidoksia luovia tekijöitä oli erilaisten työvaiheiden ja toimintojen keskeinen merkitys osana laskutusaineistoa. Arvioinnin arvoisena vaihtoehtoisena näen, että laskutus- ja tilitysmoduulin toiminnallisuus kerrostettaisiin omiksi moduuleikseen eli luonteeltaan vertikaalisesta funktiosta tulisi horisontaalinen toimintokerros. Ja yleistäen, jokaisen vertikaalin mielekkyys versus horisontaali tulisi arvioida erikseen erilaisine laajennusvaihtoehtoineen. Myös huomioiden sen, että integroitukin ratkaisu voi olla käypä vaihtoehto.

Uudet moduulit ovat järjestelmälle sekä mahdollisuus että haaste. Uusia moduuleja arvioitaessa pitää kuitenkin olla selkeä säännöstö sille, mitkä ovat perustamiskriteerit ja kuinka hallitaan liiallisen varioitumisen aiheuttamat ongelmat. Säännöstö voi muodostua toiminnallisesta, teknisestä ja kaupallisista vaatimuksista sekä sisältää päätöksenteon kriteerit ja vastuun.

Järjestelmän joustavuutta muutoksiin voidaan parantaa myös sillä, että työkierron vaiheistusta ja sen räätälöityvyyttä kehitetään. Näkisin, että tällä voidaan merkittävästi vähentää sitä painetta, joka luo sisäistä variointia asiakaskohtaisten moduulien kehittämiseen. Muutossuunta tässä yhteydessä on kohti vielä pienempiä toiminnallisia moduleja, joita voidaan konfiguroinnin avulla säädellä joustavina ketjuina.

9.2. Työn ja työvälineiden kehittäminen

Koska on odotettavissa, ettei samanlaista resursointia ole jatkossa käytettävissä tuotteen kehitykseen kuten tässä projektissa on ollut, tulee muutosten hallintaan ja toteutukseen kiinnittää huomiota. Toimintamalli pitäisi enemmän olla jatkossa enemmän bottom-up kuin top-down -tyylinen. Eli inkrementit ovat pienempiä kokonaisuuksia ja parannuksia, jotka jatkossa mahdollistavat nopeamman siirtämisen tuotantoympäristöön.

Myös kokonaissuoritusta arvioitaessa on syytä kiinnittää huomiota työvälineisiin ja käytettäviin teknologioihin. Jälkiviisaasti arvioiden käyttöliittymään tekemiseen valittiin liian kankea ja epäkypsä teknologia. Koska vaatimukset käyttöliittymälle ovat suuret ja siltä edellytetään monipuolisuutta, on syytä arvioida onko jokin muu vaihtoehto tehokkaampi kuin selain ja/tai voidaanko tehdä joitakin oletuksia selaimen suhteen tai ohjelmointiympäristöön. Pitkän linjan ohjelmointikokemuksen omaavana oma näkemykseni on, ettei JavaScript-pohjainen ohjelmointi ole se paras vaihtoehto vaan itse kaipaisin enemmän kunnollisen ohjelmointialustan, kehityskieleltään homogeenisen alustan ja hyvät debug-työkalut. Selainpohjaisille ratkaisuille on käytettävissä uusia tekniikoita kuten Ruby-on-Rails (Ruby 2003) tai Silverlight (2008) tai client ratkaisuille esimerkiksi Qt (Nokia 2008), tai .NET (Microsoft 2002) -pohjaisia ohjelmointiympäristöjä.

10. Yhteenveto

Tässä työssä on tarkasteltu varioituvan ohjausjärjestelmän modulaarisuuden kehittämistä Sesca Logistics IT Oy:n Fleetlogis logistiikan tuotannonohjausjärjestelmän kolmannen sukupolven versiolla. Päähuomio tässä työssä kiinnitettiin tuotteen muunneltavuuden edellytysten kehittämiseen ja niihin asioihin, joihin tulee tai tulisi jatkossa kiinnittää enemmän huomiota. Tuotannonohjauksen yksittäiset prosessit ja niihin liittyvät yksityiskohdat jäivät tämän selonteon ulkopuolelle.

Muunnos järjestelmän muuttamiseksi kohti suunnitelmallista muunneltavuutta on osa järjestelmän orgaanista uudistumista, jossa merkittävä osa projektia on ollut uusien toiminnallisuuksien lisääminen järjestelmään. Tavoitteena on ollut enemminkin parantaa järjestelmän joustavuutta muutoksiin kuin tehdä siitä täydellinen muunneltava tuote.

Logistiikan toimiala, jolle tuote on suunnattu, on hyvin monimuotoinen ja rakenteeltaan pirstalloitunut. Pääsääntöisesti yritykset keskittyvät joko pitkien etäisyyksien runkoliikenteeseen tai paikalliseen jakelutoimintaan. Yhtenäisen kokonaisuuden muodostaminen on erittäin hankalaa johtuen yrityksiin vakiintuneista useista erilaisista työtavoista ja käytännöistä. Käytännön eroavuuksista huolimatta tarkastelutasoa yleistämällä yrityksistä voidaan kuitenkin löytää yhteneviä prosesseja tai työvaiheita. Merkittävää on kuitenkin että tuotannon läpivientiin sisältyy useita käsityövaltaisia ja tarkkuutta vaativia työvaiheita.

Oleellinen osa järjestelmän toiminnallisuutta on lisätä tuotannon suunnitelmallisuutta ennakko- ja tilannesuunnittelun avulla sekä mahdollistaa eriasteiset toteutumien seurannat. Seurannan tehokkuus antaa yrityksille välineen toimia nopeasti ja kustannustehokkaasti.

Tarkastelun alaisena oli sovelluksen muokkaaminen palvelemaan runko- ja jakeluliikenteen tarpeita. Tästä lähtökohdasta muotoutui samaan platformiin perustuvat tuotteet. Niillä on yhteinen ydinrakenne, prosessi- ja/tai työkierto, mutta tallennettavan tapahtumatiedon sisältö ja laajuus vaihtelee merkittävästi. Juuri kerättävän tiedon erilaisuuden hallinta oli merkittävässä roolissa varianttipäätöstä tehtäessä.

Järjestelmää uudistettiin monella eri tasolla: arkkitehtuuri, käyttöliittymä ja toiminnallisuus kokivat merkittäviä muutoksia. Arkkitehtuuri päivitettiin vastaamaan modulaarisuuden vaatimuksia, käyttöliittymän uudistettiin vastaamaan mm. erilaisten käyttäjäryhmien tarpeita ja järjestelmään lisättiin uusia toimintoja. Tehdyt muutokset ovat kuitenkin vasta alkuaskeleita, ja luonteeltaan enemmänkin mahdollistajia kuin varsinaista varioituvuutta.

Tärkeimpänä havaintona on tässä työssä todettu varioituvuuden kokonaisvaltaisuuden ymmärtäminen läpi koko organisaation. Se edellyttää monenlaisia muutoksia organisaatiossa ja projektityöskentelyssä tavoitteen saavuttamiseksi. Asiakasräätälöinnin yhteyteen liitettynä projektin läpivieminen vaativine moduulimäärittelyineen on erittäin haasteellinen tehtävä. Varioituvuuden kehittäminen siten on ollut yhtälailla tuotteen kuin organisaationkin haaste.

Lisäksi työssä on tarkastelu mahdollisia jatkokehitysmahdollisuuksia niin tuotteen kuin työvälineiden ja -tapojen osalta.

11. Lähteet

AKE. 2009. ADR [WWW]. Ajoneuvohallintokeskus [viitattu 20.08.2009]. Saatavissa: http://www.ake.fi/AKE/Ammattiliikenne/ADR/.

Alexander, C. 1977. A Pattern Language: Towns/Buildings/Construction. New York: Oxford University Press, Inc.

Ambler, S. 2002. Introduction to Test Driven Design (TDD) [WWW]. [Viitattu 21.8.2009]. Saatavissa: http://www.agiledata.org/essays/tdd.html#TestDrivenDatabaseDevelopment.

Baldwin C.Y. & Clark K.B. 2000, Design Rules – The Power of Modularity. The MIT Press. Cambridge, Massachusetts. ISBN 0-262-02466-7.

Beck, K. 2003. Test-Driven Development by Example. Reading: Addison Wesley Publishing Company. ISBN 0321146530.

Gamma, E. & Helm, R. & Johnson, R., Vlissides J. 1995. Design Patterns - Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley Publishing Company. ISBN 0201633612.

Hibernate. 2009. [WWW] JBoss Enterprise. [Viitattu 21.8.2009]. Saatavissa: https://www.hibernate.org/ (etusivu).

Hölttä, K. 2003. Comparative Analysis of Product Modularization Methods. Teoksessa: Riitahuhta, A. (toim.) ym. Proceedings of NordDesign 2004 Conference: Product Development in Changing Environment, pidetty Tampereella, 18. - 20. elokuuta, 2004. Tampere. TTY Paino. s. 381 - 389.

Junit. 2009. [WWW] Sourceforge. [Viitattu 21.8.2009] Saatavissa: http://junit.sourceforge.net/ (etusivu).

Juuti, T. & Lehtonen, T. 2004. Investing Platforms - Charity Donation or Return on Invest. Teoksessa: Riitahuhta, A. (toim.) ym. Proceedings of NordDesign 2004 Conference: Product Development in Changing Environment, pidetty Tampereella, 18. - 20. elokuuta, 2004. Tampere. TTY Paino. s. 104 -113.

Lehtonen, T. 2003. Modularisuuden muodot, luentomateriaali. Tampere. Tampereen teknillinen yliopisto. Koneensuunnittelu.

Lehtonen, T. 2007. Designing Modular Product Architecture in the New Product Development. Tampere. Tampereen teknillinen yliopisto. ISBN 978-952-15-1924-6.

Leinonen. M. 2007-2008. IT konsultti, Kovanen Logistics Oy. Määrittelypalaverit; Tampere-Helsinki-Vantaa.

Macons. 2006. Ajomestari - toiminnanohjausjärjestelmä kuljetusyrityksille [WWW]. Julkaistu 2006 [viitattu 23.3.2010]. Saatavissa: http://www.ajomestari.com/index.php (etusivu).

Mäki-Mikkilä, M. 2007-2008. Kehityspäällikkö, Swanline Oy. Määrittelypalaverit; Tampere-Helsinki-Vantaa-Kempele.

.NET. 2002. [WWW]. Microsoft Corp. [Viitattu 18.2.2010]. Saatavissa: http://www.microsoft.com/NET/ (etusivu)

Nielsen, O. F. 2009. Continuous Platform Development – Synchronizing Platform and Product Development. Technical University of Denmark, Department of Management Engineering. 151 s.

Pahl, G. & Beitz, W. 1990. Koneensuunnitteluoppi. Helsinki: Metalliteollisuuden kustannus Oy. 608 s. (Käännös teoksesta Konstruktionslehre, Handbuch für Studium und Praxis. 2. painos. Berlin/Heidelberg: Springer Verlag.) ISBN 951-817-468.

Peltonen, H. & Martio, A. & Sulonen, R. 2002. PDM - Tuotetiedon hallinta. Helsinki: IT Press. 166 s. ISBN 951-826-664-6.

Peltonen, P. 2008. Vuoropäällikkö, Kovanen Logistics Oy. Määrittelypalaverit; Helsinki-Vantaa.

Pine, B. J. 1993. Mass Customization. The New Frontier in Business. Boston: Harvard Business School Press cop.

Pulkkinen, A. & Bongulielmi, L. 2004. Design of Product Families for Configuration. Teoksessa: Riitahuhta, A. (toim.) ym. Proceedings of NordDesign 2004 Conference: Product Development in Changing Environment, pidetty Tampereella, 18. - 20. elokuuta, 2004. Tampere. TTY Paino. s. 361 - 370.

Procomp. 2007. Logistiikan ratkaisu [WWW]. Procomp Solutions Oy [viitattu 23.3.2010]. Saatavissa: http://www.procomp.fi/fi/website.nsf/va_W+Pages+By+Id/34ECEE77D5C180BEC22572100029540E (etusivu).

Qt. 2008. [WWW]. Nokia Oyj. [Viitattu 18.2.2009]. Saatavissa: http://qt.nokia.com/ (etusivu)

Rask, P. 2004a. The modular system – half a century later…. Scania World Magazine [verkkolehti]. Nro 1 2004 [viitattu 21.3.2005]. Saatavissa: http://www.scania.com/news/Scania_World_magazine/1_2004/scania_world/modul_1.asp.

Rask, P. 2004b. Modular thinking stronger than ever. Scania World Maganize [verkkolehti]. Nro 3 2004 [viitattu 18.4.2005]. Saatavissa: http://www.scania.com/news/Scania_World_magazine/2004_3/modular_system.asp.

Repo, J. 2006. Projektipäällikkökoulutus. Luentomateriaali. Edutech.

Ruby. 2010. [WWW]. Ruby on Rails. [Viitattu 21.8.2009]. http://rubyonrails.org/ (etusivu).

Sarinko, K. 1999. Asiakaskohtaisesti muunneltavien tuotteiden massaräätälöinti, konfigurointi ja modulointi. Diplomityö. Teknillinen korkeakoulu, Konetekniikan osasto. Espoo. 87 s.

Oy Scan-Auto Ab. PRT-mallisto [WWW] Julkaistu 2002 [viitattu 21.3.2005]. Saatavissa: http://www.scania.fi/Our_trucks/NTR/.

Scania AB. 2005. Annual Report 2004 [WWW]. Julkaistu 2005 [viitattu 21.3.2005]. Saatavissa: http://www.scania.com/Images/6_150_tcm10-75234.pdf.

Sesca. 2007. [WWW]. Sesca IT Logistics Oy. [Viitattu 21.8.2009]. Saatavissa: http://www.fleetlogis.com/avecon/fl_site/what_is.html (etusivu).

Silverlight. 2008. [WWW]. Microsoft Corp. [Viitattu 21.8.2009] Saatavissa: http://www.silverlight.net/ (etusivu).

Sun. 2009. [WWW] Sun Microsystems, Inc. [Viitattu 21.8.2009]. Saatavissa: http://java.com/en/about/.

Struts. 2009. [WWW] The Apache Software Foundation. [Viitattu 21.8.2009]. Saatavissa: http://struts.apache.org (etusivu).

Tiihonen, J. 1999. Kansallinen konfiguroimiskartoitus - asiakaskohtainen muuntelu suomalaisessa teollisuudessa. Lisenssiaatintyö. Teknillinen korkeakoulu, Tietotekniikan osasto. Espoo. 188 s.

Tervola, J. 2004. Linja-auto on räätälintyötä. Metallitekniikka, 56: 1. S. 10 - 15. ISSN 1237-6663.

Ulrich, K. & Eppinger, D. 1995. Product Design and Development. McGraw-Hill, Inc. 289 s. ISBN 0-07-065811-0.

Ulrich, K. & Tung, K., 1991. Fundamentals of product modularity. Design/Manufacture Integration, DE: 39. New York: ASME.

Volvo Bus Finland Oy. 2002. Key Figures [verkkodokumentti]. Julkaistu 2002 [viitattu 21.3.2005]. Saatavissa: http://www.carrus.fi/key_figures.html.

Välimaa, J. 2010. Business Intelligence Solution for Logistic ERP system. Master of Science Theses. Tampereen University of technology, Department of Information and Knowledgemanagement. Tampere. 62 s.

Wikipedia-projektin osanottajat. 2009a. Spring [verkkodokumentti] Wikipedia Foundation, Inc. [Viitattu 20.8.2009]. Saatavissa: http://en.wikipedia.org/w/index.php?title=Spring_Framework&oldid=307786422.

Wikipedia-projektin osanottajat. 2009b. Hibernate [verkkodokumentti] Wikipedia Foundation, Inc. [Viitattu 20.8.2009]. Saatavissa: http://en.wikipedia.org/w/index.php?title=Spring_Framework&oldid=307786422.

Wikipedia-projektin osanottajat. 2009c. Transaction [verkkodokumentti] Wikipedia. [Viitattu 21.8.2009]. Saatavissa: http://en.wikipedia.org/w/index.php?title=Object-relational_impedance_mismatch&oldid=306167071.

Wikipedia-projektin osanottajat. 2009d. Test-driven development [verkkodokumentti]. Wikipedia. [Viitattu 21.8.2009]. Saatavissa: http://en.wikipedia.org/w/index.php?title=Test-driven_development&oldid=307653329.