TAMPEREEN TEKNILLINEN YLIOPISTO

VILLE KERÄNEN

TUOTETIEDONHALLINTAJÄRJESTELMÄN INTEGRAATIO

Diplomityö

Tarkastaja: professori Kai Koskimies
Tarkastaja ja aihe hyväksytty tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 6. kesäkuuta 2012

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
KERÄNEN, VILLE: Tuotetiedonhallintajärjestelmän integraatio
Diplomityö, 44 sivua, 6 liitesivua
Elokuu 2012
Pääaine: sulautetut järjestelmät
Työn tarkastaja: professori Kai Koskimies
Avainsanat: Järjestelmäintegraatio, ESB, EAI, EIP, tuotetiedonhallinta, PDM, XML, SOAP

Yritysten tietojärjestelmät koostuvat useista pienemmistä kokonaisuuksista, joiden toiminta on saatettava yhtenäiseksi siten, että se tukee yrityksen liiketoimintamallia mahdollisimman luonnollisesti. Tämän toteuttamiseksi on eri sovellukset saatettava keskustelemaan keskenään hallitusti ja varmasti. Sovellusten integrointi on yleinen ongelma ja näihin liittyvät projektit pitkiä sekä vaativia.

Ratkaisuksi on tarjottu erilaisia arkkitehtuureja kuten point-to-point, enterprise application integration (EAI) ja enterprise service bus (ESB). Näistä ESB on nykyään yleisimmin käytössä oleva malli. ESB:n toteuttamiseksi on kehitetty erilaisia väliohjelmia, jos jotka kokoavat yhteen integrointia tukevia toimintoja.

Sovellusten välinen tiedonvaihto keskittyy viestitykseen, jossa sovellusten ohjelmointirajapintojen kautta tehdään pyyntöjä toiminnoista. Toiminnot kuvataan palveluina. Yleinen viestimuoto on rakenteiset dokumentit, käytännössä XML. SOAP:n avulla voidaan pyynnöt kuvata myös XML muotoisina, ja vastaus saadaan niin ikään SOAP:n avulla hyötykuorman sisältyessä viestin *body* osioon.

Tuotetiedonhallintajärjestelmän (PDM) tarkoituksena on tarjota yritykselle sijainti eri tuotteisiin liittyvälle tiedolle. Tämän työn puitteissa tuote on sähkömoottori, ja PDM sisältää tiedot kaikista tehdyistä moottoreista, niiden sisältämistä osista sekä niihin liittyvistä mekaanisista suunnittelukuvista. PDM:ään voidaan myös kuvata valintasäännöt tuotekonfiguraattorille, jolloin asiakkaan haluaman tuotteen rakenne voidaan tehdä automaattisesti ilman, että järjestelmä sisältäisi valmiiksi juuri kyseisenlaisen tuotteen.

Tässä työssä on tavoitteena saada yrityksen uusi PDM integroitua alihankkijan järjestelmään. Myös uuden toiminnanohjausjärjestelmän (engl. Enterprise Resource Planning, ERP) integrointia sivutaan integraation suunnittelumallien (engl. Enterprise Integration Patterns) avulla. Esityönä selvitetään miten tuoterakenne saadaan määriteltyä uuteen PDM:n, ja miten vanhasta PDM:stä saadaan tuotua data uuteen järjestelmään. Tämän lisäksi suunnittelumallien avulla hahmotellaan ratkaisuja järjestelmää vaivaaviin ongelmiin.

Työn tuloksena saatiin määriteltyä tuoterakenne, sekä tuotua vanhasta PDM:stä valintasäännöt ulos. Tälle datalle tehtiin muunnos Python-kielellä, jonka jälkeen se saatiin vietyä uuteen järjestelmään. Suunnittelumalleilla suunniteltiin mahdollinen integrointiratkaisu, ja uuden PDM-sovelluksen XML-datalle saatiin tehtyä muunnos Python-kielellä. Se vastaa muunnoksen jälkeen vanhan PDM-sovelluksen XML-muotoa. Näin alihankkijan ei tarvitse tehdä muutoksia järjestelmäänsä uuden PDM:n tullessa käyttöön.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KERÄNEN, VILLE: Integration of Product Data Management system
Master of Science Thesis, 44 pages, 6 appendix pages
August 2012
Major: embedded systems
Examiner: professor Kai Koskimies
Keywords: System integration, ESB, EAI, EIP, product data management, PDM, XML, SOAP

Information systems in an enterprise are compiled from numerous smaller entities. The over all functionality must be merged together in a way that supports the business model of the enterprise as a whole. To implement this, all the applications must speak together in a controlled and determinated fashion. Application integration is a common problem and the projects are often long and demanding.

There are different architectures to solve this including point-to-point, enterprise application integration (EAI) and enterprise service bus (ESB). ESB is the one which is mostly used today, and there are many solutions presented as middleware which gather different supporting functions to integration.

Application-to-application data transfers are handled with messaging, where the application programming interfaces (APIs) are used to request functionalities. A function is realized as a service. The usual format for messages is extended mark-up language (XML). With the help of simple object access protocol (SOAP) one can describe the service request in XML, and the answer is also given as a SOAP message including the payload as a message body.

Product data management system (PDM) is used to offer a place for product data for the company. Within this thesis, a product is an electric motor, and the PDM has all the information on the motors already done, including the components and mechanical designs. PDM systems also usually implement a configurator using configuration rules. This makes it possible for the customer to order a motor which hasn't yet been built. No further design is needed, as the PDM can configure the motor from the pre-defined rules.

The object of this thesis is to integrate the new PDM software with the system of the subcontractor. Also, the integration of the new ERP software (Enterprise Resource Planning) is inspected with the help of EIP (Enterprise Integration Patterns). First, the product structure logic of the new PDM must be defined. After that it must be defined how the data can be migrated from the old PDM. Also, some problematic issues in the system are approached via integration patterns.

As a result the product structure was defined, and the configuration rules were imported and converted for the new PDM. The conversion was done with Python scripting language. Integration patterns were used to design a possible solution for the integration, and Python was used to develop a script to transcript the new PLMXML in to the old XML format. This way, the subcontractor will not need any changes after the new PDM is in use.

ALKUSANAT

Tässä se nyt on, 6-vuotiaana Hakalan ala-asteella aloittamani koulutusputken huipentuma. Opiskelujen saattaminen päätökseen ei aina ollut täysin varma asia, mutta erinäisten vastoinkäymisten myötä vahvistuneen "no vaikka sitten väkisin"-mentaliteetin avulla sekin hoitui. Opinnot takana, elämä edessä!

Tämän työn apuna on ollut ihmisiä, joita tahtoisin kiittää. Työn tarkastajana toimivaa professori Kai Koskimiestä, sekä Kalle Vallia Konecranes Oyj:ltä joka oli apuna tuoterakenteen määrittelyssä. Myös Tieto Oyj:n Markus Lindberg oli apuna tietokannan kanssa.

Edellä mainittujen lisäksi kiittäisin myös heitä, jotka muuten vain ovat hienoja ihmisiä. Heistä esimerkkeinä toimikoon vanhempani, naapurintyttö ja ystävät. Kiitos. Myös muutama laulu- ja soitinyhtyekin tulee mieleen.

Toscana, Italia 08.07.2012

Ville Keränen

SISÄLLYSLUETTELO

LYHENTEET JA KÄSITTEET 7
1 Johdanto 10
1.1 Työn tausta ja rajaus 10
1.2 Työn sisältö 11
1.3 Työn tavoite 12
2 Systeemi-integraatio 13
2.1 Datamigraatio 13
2.2 API ja väliohjelmat 14
2.3 Viestiformaatit ja datan muoto 17
2.3.1 Siirtotiedostot 17
2.3.1.1 XML 17
2.3.1.2 PLMXML 18
2.3.1.3 iDoc 18
2.3.2 Tietokannat tiedonvaihdon välineenä 20
2.4 Integrointiarkkitehtuurit 20
2.4.1 Point-to-point 20
2.4.2 EAI 22
2.4.3 ESB 24
2.4.4 SOA ja palvelut 25
2.5 WSDL ja SOAP 27
2.6 Integraation suunnittelumallit 29
3 Järjestelmän kuvaus ja projektin tavoitteet 34
3.1 Nykytilanne 34
3.1.1 Integraation kohteet 35
3.1.1.1 PDM 35
3.1.1.2 ERP 37
3.1.1.3 CAD 38
3.1.2 Viestiliikenne järjestelmässä 38
3.2 Integraation ja migraation tarve 42
3.3 Tavoitetila 43
4 Uusi järjestelmä 44
4.1 Teamcenter-Navision integraatio 45
4.1.1 Teamcenter-Navision WebSphere-vuo 46
4.2 Teamcenter-SAP integraatio 47
4.2.1 Teamcenter-SAP WebSphere-vuo 48

4.3 Datamigraatio ... 48
4.3.1 Tuotemallien valintasääntöjen siirtäminen Teamcenteriin ... 48
5 Parannuksia järjestelmään ... 50
5.1 Tapaus: pyyntöjen hukkuminen ... 50
5.2 Tapaus: nimike varattuna ... 52
6 Arviointi ... 53
6.1 Tuotemalli ja valintasäännöt ... 53
6.2 Integraatio ja muunnossääntö ... 53
LÄHTEET ... 55
LIITE I: SQL haku valintasäännöille ... 57
LIITE II: Python lähdekoodi valintasääntöjen muuntajalle ... 58
LIITE III: Python lähdekoodi rakennemuunnokselle ... 60

LYHENTEET JA KÄSITTEET

ANSI ASC X12	Rakenteisen tiedonsiirron standardia ASC X12 ylläpitävä komitea (engl. American National Standards Institute Accredited Standards Committee)
Asiakas	Asiakas-palvelin–mallissa palvelimelta toimintoja tarvitseva puoli, yleensä siis laite tai ohjelma
AtonPDM	Modultek Oy:n tuotetiedonhallintajärjestelmä
Body	Dataviestin osa joka sisältää siirrettävän informaation
BAPI	SAP:n API (ks. ohjelmointirajapinta) (engl. Business Application Programming Interface)
Datatyypin määritelmät	Kieli jolla XML-dokumentille määritellään skeema (engl. Data Type Definition, DTD)
EAI	Ohjelmistokehys joka tarjoaa ratkaisuja ohjelmistointegraatioon pohjautuen EAI-arkkitehtuuriin (engl. Enterprise Application Integration)
EDI	Rakenteisen tiedon siirto organisaatioiden välillä elektronisesti (engl. Electronic Data Interchange)
EDIFACT	Yhdistyneiden kansakuntien kehittämä EDI-standardi (engl. Electronic Data Interchange for Administration, Commerce and Transport)
ESB	Ohjelmistokehys joka tarjoaa ratkaisuja ohjelmistointegraatioon pohjautuen ESB-arkkitehtuuriin (engl. Enterprise Service Bus)
ESQL	Tapa jolla tietokantakyselyjä voidaan tehdä perinteisen ohjelmakoodin lomassa (engl. Embedded Structured Query Language)
Etäproseduurikutsupohjainen integraatio	Rajapinta jonka avulla kutsutaan sovelluksen funktioita (engl. Remote Procedure Call, RPC)
FTP	Tiedonsiirtoprotokolla (engl. File Transfer Protocol)
Header	Dataviestin osa joka sisältää siirtoon liittyviä tietoja
HTTP	Tiedonsiirtoprotokolla (engl. HyperText Transfer Protocol)
Hub-and-spoke	Integraatioarkkitehtuuri jossa kaikki tieto kulkee keskiössä sijaitsevan *hub*in kautta
iDoc	SAP-järjestelmän käyttämä dataformaatti (engl. Intermediate Document)
Integraation suunnittelumallit	Kokoelma korkean tason malleja erilaisista ratkaisuista integraatio-ongelmiin (engl. Enterprise Integration Patterns, EIP)
Integraatiotyylit	Suunnittelumallien kategoria, eri tavat joilla sovellukset voivat siirtää dataa keskenään (engl. integration styles)
Järjestelmänhallinnan mallit	Suunnittelumallien kategoria, eri tavat joilla integroitua kokonaisuutta voidaan hallita ja testata (engl. system management patterns)

Kanavamallit	Suunnittelumallien kategoria, sisältää kuvaukset perimmäisistä väliohjelmien ominaisuuksista (engl. channel patterns)
Lohko	Pienempi osa suurempaa kokonaisuutta, yleensä datatietue (engl. block)
Löyhä kytkentä	Sovellusten integrointi siten, että ne ovat mahdollisimman vähän riippuvaisia toistensa toiminnasta (engl. loose coupling)
Muunnosmallit	Suunnittelumallien kategoria, eri tavat joilla viestin muotoa voidaan muokata (engl. transformation patterns)
Ohjelmointirajapinta	Rajapinta jonka kautta sovelluksen toimintoja voidaan kutsua (engl. Application Programming Interface, API)
Oliopohjainen integraatio	Rajapinta jonka avulla kutsutaan sovelluksen yksittäisiä olioita (engl. Object Request Broker, ORB)
Osaluettelo	Tuotetiedonhallintajärjestelmän tietue joka sisältää kaikki kyseisen tuotteen koostavat osat (engl. part list)
Palvelin	Asiakas-palvelin–mallissa asiakkaalle toimintoja tarjoava puoli, yleensä siis laite tai ohjelma
Palvelu	WSDL-elementti, kokoelma porteista jotka muodostavat palvelun (engl. service)
Palveluarkkitehtuuri	Arkkitehtuuri jossa sovelluksen toiminnot kuvataan palveluina (engl. Service Oriented Architecture, SOA)
PLMXML	Siemens AG:n Teamcenter-ohjelmiston käyttämä dataformaatti (engl. Product Lifecycle Management XML)
Point-to-point	Integrointiarkkitehtuuri jossa kaikki sovellukset on yhdistetty suoraan toisiinsa
Portti	WSDL-elementti, siirrettävän datan määränpää, esimerkiksi URL (engl. port)
Porttityyppi	WSDL-elementti, palvelun rajapinnan kuvaus kokoelmana toimintoja, jotka vähintään yksi portti toteuttaa (engl. port type)
Python	Korkean tason ohjelmointikieli
Päätepisteiden mallit	Suunnittelumallien kategoria, eri tavat joilla viesti-järjestelmän asiakkaat tuottavat ja kuluttavat viestejä (engl. endpoint patterns)
Reititysmallit	Suunnittelumallien kategoria, eri tavat joilla viestejä voidaan ohjata vastaanottajalle (engl. routing patterns)
Sitoutuminen	WSDL-elementti, määrittelee porttityypin protokollan ja datamuodon (engl. binding)
SAP	SAP AG:n toiminnanohjausjärjestelmä
SMTP	Sähköpostiprotokolla (engl. Simple Mail Transfer Protocol)

SOAP	Integraatioprotokolla (engl. Simple Object Access Protocol)
SOAP-kirje	Datarakenne johon SOAP-muotoinen viesti kääritään (engl. envelope)
Solmu	Toiminnon tai toimintoja sisältävä etappi viestin reitillä (engl. node)
SQL	Kieli jolla voidaan tehdä tietokantakyselyjä (engl. Structured Query Language)
Tapahtumaohjattu kuluttaja	Viestin kuluttajan tapa reagoida viesteihin, toteuttaa toiminnon tapahtuman laukaisemana (engl. event-driven consumer)
Tietokoneavusteinen suunnittelu	Konkreettisen tuotteen suunnittelu tietokoneen avulla (engl. Computer Aided Design, CAD)
Tiivis kytkentä	Sovellusten integrointi siten, että ne ovat riippuvaisia toistensa toiminnasta (engl. tight coupling)
Toiminnanohjausjärjestelmä	Järjestelmä joka optimoi yrityksen tuotantoprosessin käytössä olevien resurssien perusteella (engl. Enterprise Resource Planning, ERP)
Toiminto	WSDL-elementti, kuvaus palvelun toiminnosta (engl. operation)
Tuotemalli	Kaikki yhden tuoteperheen mahdolliset variaatiot (engl. Generic Product Structure, GPS)
Tuotetiedonhallintajärjestelmä	Järjestelmä joka sisältää yrityksen kaikkiin tuotteisiin liittyvän informaation (engl. Product Data Management, PDM)
Tyyppi	WSDL-elementti, ilmoittaa käytössä olevat datatyypin määritelmät (engl. types)
URL	Merkkijono joka viittaa Internetissä sijaitsevaan resurssiin (engl. Uniform Resource Locator)
Viesti	WSDL-elementti, abstrakti määritelmä siirrettävästä datasta (engl. message)
Viestin muodostamisen mallit	Suunnittelumallien kategoria, kuvaa viestin tarkoituksen, muodon ja sisällön (engl. message construction patterns)
Viestipohjainen integraatio	Integraatiotapa jossa sovellukset vaihtavat dataa viesteinä (engl. message oriented)
Väliohjelma	Integraation keskipiste, sovellus joka tarjoaa valikoiman integrointiominaisuuksia (engl. middleware)
Web-palvelu	Järjestelmä jossa sovellukset integroidaan verkon yli (engl. web service)
WSDL	Web-palveluiden kuvauskieli jolla määritellään miten palvelua käytetään (engl. Web Services Description Language)
XML	Rakenteisten dokumenttien standardi (engl. eXtensible Markup Language)
XSLT	XML-muotoinen dokumenttien muunnoskieli (engl. eXtensible Stylesheet Language Transformations)

1 Johdanto

Monimutkaisia tuotteita valmistavien yritysten avuksi on kehitetty lukuisia ohjelmistoja, joilla hallinnoidaan tuotantoon, tuotteisiin ja asiakkaisiin liittyviä prosesseja niiden tehostamiseksi. Yrityksen tietojärjestelmään liittyy kokonaisuudessaan yleensä useita eri alan ja valmistajan ohjelmia. Näiden tuottava käyttö vaatii niiden saumattoman toiminnan yhteistyössä, ja tähän tähdätään erilaisissa integraatioprojekteissa. Tavoitteena on saada tieto liikkumaan eri ohjelmien välillä siten, että oikea tieto olisi saatavilla mahdollisimman nopeasti sillä henkilöllä, jonka vuoro on esimerkiksi seuraavaksi tehdä osansa tuotteen suunnittelusta. Usein tarpeellinen tieto siirtyy ohjelmien välillä automaattisesti erilaisten tapahtumien laukaisemina, jolloin näihin ei tarvita ihmistä väliin.

Yrityksen siirtyessä uuden valmistajan ohjelmistoon on huomioitava se, miten siinä voidaan käyttää vanhaan ohjelmaan tallennettua tietoa. Tämä tieto on todennäköisesti täysin erilaisessa muodossa tietokannassa, sillä eri ohjelmat sisältävät harvoin yhteensopivat tietomuodot. Datamigraatiossa pyritään muokkaamaan olemassa oleva data sellaiseksi, että se tukisi mahdollisimman suoraan uuden ohjelman toimintoja.

Jotta edellä mainitut ongelmat olisi ratkaistavissa, on ensimmäiseksi tutkittava kyseessä olevan tuotteen, eli nostimen moottorin, rakenne niin vanhassa kuin uudessa ohjelmassa. Tästä saadaan selkäranka toteutettavalle työlle.

1.1 Työn tausta ja rajaus

Konecranes toimii maailman nostinmarkkinoilla. Sen asiakaskuntaan kuuluu niin laivanrakennusteollisuus, voimalaitokset kuin ydinvoimalat, joten tuotteiden toimivuus ja luotettavuus ovat kriittisen tärkeitä ominaisuuksia.

Yhtiön pääkonttori sijaitsee Hyvinkäällä Suomessa, ja tuotantolaitoksia sillä on yhteensä 12 eri maassa. Vuoden 2010 liikevaihto ylitti 1500 miljoonaa euroa, ja työntekijöitä on listoilla 10042 henkeä. [1]

Nostinlaitteiden kunnossapito on suuri osa Konecranesin toimintaa. Kunnossapito kattaa kaikenmerkkiset teollisuusnosturit, satamalaitteet ja työstökoneet, ja palveluihin kuuluu muun muassa ennakoiva kunnossapito, päivystyshuoltokäynnit sekä varaosat ja erilaiset

modernisointiprojektit. Vain noin neljäsosa näistä palveluista kohdistuu Konecranesin valmistamiin tuotteisiin.

Toinen pääalue on laitteet, jossa tarkoituksena on tarjota nostureita ja sen komponentteja asiakkaille. Tuotteita myydään niin Konecranes brändin alaisena, kuin itsenäisillä tuotemerkeillä kuten STAHL CraneSystems, Verlinde ja Sanma. Nämä erottaa toisistaan asiakas, Konecranes brändätyt tuotteet myydään suoraan loppuasiakkaalle, kun muita tuotemerkkejä myydään esimerkiksi muille nosturivalmistajille. Konecranesin tuotteisiin kuuluu myös nostureiden räjähdyssuojateknologia, sekä konttien ja muun raskaan tavaran käsittelyyn tarkoitetut nostimet [1].

Tämä työ on osa Konecranes Oyj:n tietojärjestelmän päivitysprojektia. Järjestelmä koostuu useasta eri ohjelmasta, sisältäen muun muassa myynnin, asiakassuhteiden, suunnittelun ja tuotteiden hallintaa helpottavia ratkaisuja. Tarkoituksena on kattaa kaikki tarjousten tekemisestä valmiin tuotteen toimitukseen, ja uuden tuotteen suunnittelusta vanhojen ylläpitoon. Tuote tarkoittaa koko laajuudessaan nostinta.

Tämä työ on kuitenkin edellisestä kuvauksesta rajattu tuotteen osalta pelkästään nostimen moottoriin, sekä järjestelmän osalta valmiiden moottoreiden tietojen siirtämiseen ja muokkaamiseen. Käytännössä tämä tarkoittaa tuotetiedonhallintaan (engl. Product Data Management, PDM) ja toiminnanohjaukseen (engl. Enterprise Resource Planning, ERP) keskittymistä.

1.2 Työn sisältö

Työssä käydään ensin läpi integraatioon ja migraatioon liittyviä teoreettisia ratkaisuja, sekä esitellään erilaisia integrointiteknologioita. Tämän jälkeen esitellään yrityksen tietojärjestelmän toimintaa sekä sen rakenne, ja kuvaillaan tilaa, johon on tarkoitus päästä. Tämä luo pohjan itse selvitystyölle.

Selvitysosuudessa tutkitaan, miten moottoreiden tuotemallit voidaan rakentaa uuteen tuotetiedonhallintajärjestelmään. Kun se on selvitetty, voidaan etsiä ratkaisut näiden rakenteiden siirtämiseksi järjestelmän sisällä, ja miten vanhat tuotemallirakenteet voidaan muokata ja siirtää uuden järjestelmän alle.

Kolmas osio esittelee parannusehdotuksia yrityksen käytössä olevaan ratkaisuun. Siinä etsitään olemassa olevia heikkouksia ja huonoja toteutustapoja, sekä suunnitellaan integraation suunnittelumalleilla ratkaisuja. Viimeiseksi käydään läpi työn onnistumista annettujen vaatimusten valossa.

1.3 Työn tavoite

Päivitysprojektin lopullisena tavoitteena on saada järjestelmä toimimaan vähintään yhtä hyvin kuin nykyään. Kokonaistavoite ylitetään mitä todennäköisimmin, sillä korvaavat ohjelmat ovat ominaisuuksiltaan kehittyneempiä. Tämä työ toimii eräänlaisena esiselvityksenä sille, miltä pohjalta projektia voitaisiin lähteä toteuttamaan, ja minkälaisia parannusmahdollisuuksia nykyiseen ratkaisuun on olemassa.

Ratkaisuissa pyritään ottamaan huomioon sovellusten uudet mahdollisuudet, ja hyödyntämään niitä mikäli ne tuovat lopputulokseen havaittavaa lisäarvoa ilman, että niiden käyttö lisää merkittävästi työmäärää.

2 Systeemi-integraatio

Teknologia-alan yrityksissä on menneisyyttä tarkastellessa ollut perinteisesti käytössä monia eri ohjelmia, joista jokainen on toteuttanut oman, kapea-alaisen tehtävänsä yrityksen tuotantoprosessissa. Nämä ohjelmat on räätälöity kyseisen yrityksen tarpeita vastaaviksi, ja niitä ei ole suunniteltu toimimaan itsenäisesti keskenään. Niiden käyttämät tiedostomuodot ovat suljettuja, ne eivät sisällä rajapintoja joiden kautta toimintoja voisi käyttää ohjelman ulkopuolelta, eikä niitä ole muutenkaan toteutettu toimimaan osana suurempaa kokonaisuutta [2, sivu 4].

Tällaisten ohjelmien integrointi on hyvin riskialtista ja kallista työtä. Jo pelkän suunnitteluvaiheen toteuttaminen on vaativaa. Jokainen ohjelma tarvitsee osaltaan oman asiantuntijansa, ja koska ohjelma ei välttämättä ole käytössä kuin yhdessä yrityksessä, ei näitäkään henkilöitä yleensä ole montaa. Projekti voi mahdollisesti myös kaatua heti alkutekijöihinsä mikäli todetaan, ettei ohjelmien integrointi ole mahdollista joko teknillisistä tai taloudellisista syistä. Vaikka työhön olisikin jo ryhdytty, on vaarana ettei sitä voida saattaa loppuun. Vuosien uurastuksen jälkeen tulee eteen tilanne, jossa jokin aiemmin tunnistamatta jäänyt ongelma estää toteutuksen, tai budjetti ei enää riitä eteenpäin viemiseksi. Mikäli projekti saatettaisiinkin loppuun, ei lopputuloksen laadusta ole takeita ongelman vaikeuden takia. Kun toistensa kanssa alun perin täysin yhteensopimattomat ohjelmat liitetään yhteen, seuraa tästä täysin uusia ongelmia [3].

Myöhemmin markkinoille tuli niin sanottuja pakattuja ohjelmistoja, jotka sisälsivät useita liiketoimintaprosessiin liittyviä ohjelmia erillisinä komponentteina. Nämä osasivat vaihtaa dataa keskenään, ja niitä oli mahdollista räätälöidä yrityksen tarpeiden mukaisiksi valitsemalla halutut toiminnot. Näiden myötä ymmärrettiin, kuinka tärkeää ohjelmien integrointi on, ja läheisessä menneisyydessä on myös tämän takia siirrytty avoimempaan suuntaan ohjelmistojen rajapinnoissa.

2.1 Datamigraatio

Kun yritys päivittää jonkun osan järjestelmästään, on integroinnin lisäksi otettava huomioon se, miten käytöstä poistuvaan ohjelmaan tallennettu tieto on käytettävissä uudessa mahdollisimman tehokkaasti. Tämän datan tuottamiseen on todennäköisesti käytetty paljon aikaa, ja sitä on kerääntynyt vuosien saatossa suuret määrät, joten on selvää että sen muokkaaminen uuteen järjestelmään ohjelmallisesti on tehokkaampaa

kuin sen syöttäminen manuaalisesti. Vaikka datamigraatiossa on pohjimmiltaan kyse vain tiedon siirrosta uuteen paikkaan, täytyy siinä ottaa huomioon useita asioita kuten ymmärtää uuden järjestelmän tuomat hyödyt, löytää oleellisin tieto, saada se mahdollisimman puhtaaksi ja olla siirtämättä turhaa dataa. Myös säännönmukaisuuden noudattaminen lailliselta ja hallinnolliselta kannalta on otettava huomioon [4, sivu 9]. Näiden laiminlyönti ymmärrettävästi tuottaa käyttökelvotonta dataa ja aiheuttaa huomattavia ongelmia.

2.2 API ja väliohjelmat

Ohjelmointirajapinta (engl. Application Programming Interface, API) käsittää sovelluksen ulkopuolelle näkyvät funktiot ja rutiinit. Alkujaan näillä mahdollistettiin käyttöjärjestelmän ja eri oheislaitteiden ajureiden keskinäinen kommunikointi. Nykyään rajapintoja tehdään monenlaisiin eri sovelluksiin, koska näiden avulla on mahdollista käynnistää jokin toiminto ja saada sen tulos ilman, että jossain välissä tarvittaisiin ihmisen antamaa vastetta. Ilman näitä rajapintoja olisi mahdotonta integroida sovelluksia toimimaan keskenään yhtenä kokonaisuutena. Tapa on ollut käytössä olio-ohjelmoinnissa, jossa rajapinnoilla voidaan ohjata olioiden toimintaa ja tilaa käynnistämällä niiden funktioita. Tekniikka on vain siirretty sovellusten sisäisestä käytöstä koko sovelluksen ulkopuolelle näkyväksi liittymäksi, ja oliot ovat kasvaneet pienistä tietyn toiminnon luokista täysiverisiksi ohjelmistoiksi.

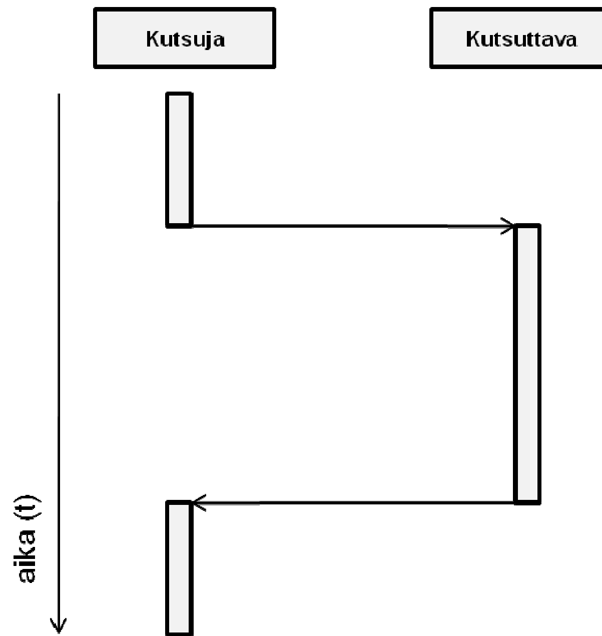
Ohjelmointirajapinnoilla väliohjelmat voivat keskustella muiden sovellusten vastaavien rajapintojen kanssa. Koko prosessin kannalta ei ole olemassa yksiselitteisesti asiakasta ja palvelinta, vaan rooli vaihtuu sovellusten välillä sitä mukaa, kutsuuko ohjelma jonkin funktion käynnistämistä, vai kutsutaanko sitä antamaan vastaus jonkin toisen sovelluksen toimesta.

Väliohjelma (engl. middleware) yhdistää eri sovellukset toisiinsa ja hoitaa viestien reitityksen, sekä datamuunnokset eri tiedostomuotojen välillä. Näin saadaan tietty läpinäkyvyys erilaisilla alustoilla ja protokollilla toimiville ohjelmille. Ohjelmat, jotka on kirjoitettu eri kielillä eri käyttöjärjestelmille ja tietokonearkkitehtuureille voidaan saattaa kommunikoimaan keskenään. Väliohjelman ei tarvitse paljastaa kuin ohjelmointirajapintansa ulkomaailmalle, ja tämän kautta voidaan eri ohjelmat konfiguroida vaihtamaan dataa. Jotta väliohjelma saataisiin toimimaan näin

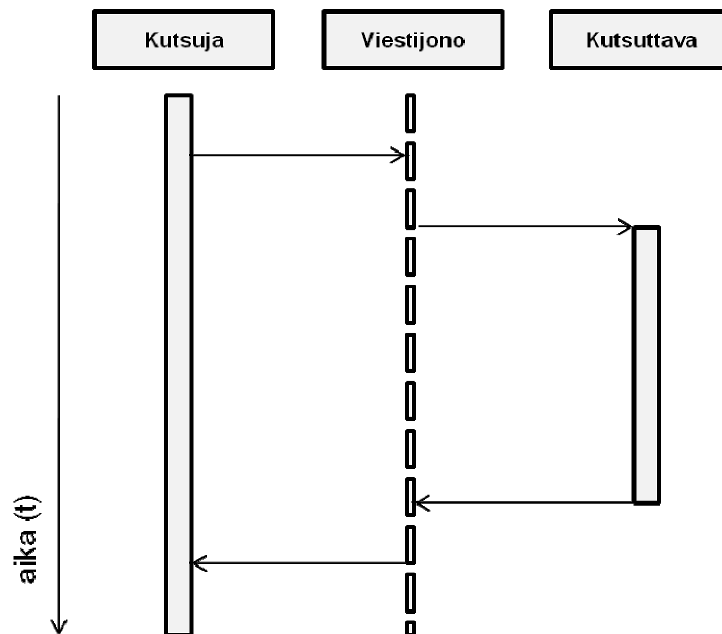
mutkattomasti, vaatii sen käyttöönotto tarkkuutta koska hyvin monta asiaa täytyy huomioida. Jo eri tietokonearkkitehtuurien erot tulkita tavun eniten merkitsevä bitti voi aiheuttaa vakavia ongelmia, mikäli sen merkitystä ei huomioida.

Eri ohjelmointirajapintoja tarjotaan eri integrointiongelmien ratkaisuksi, mutta useat väliohjelmat sisältävät kokonaisvaltaiset toiminnot. Viesti- (engl. message oriented), etäproseduurikutsu- (engl. Remote Procedure Call, RPC) ja oliopohjaisuus (engl. Object Request Broker, ORB) ovat yleisimpiä väliohjelmien tarjoamia tapoja välittää dataa eri sovellusten välillä. Eri tapoja yhdistämällä saadaan yrityksen kaikki sovellukset integroitua keskenään ilman, että jonkun kohdalla tarvitsisi tinkiä suorituskyvystä tai tietoturvallisuudesta.

Väliohjelma voi välittää yhdistämilleen sovelluksille dataviestejä joko synkronisesti tai asynkronisesti. Synkronisessa kommunikoinnissa kutsuva ohjelma joutuu odottamaan kutsumaansa ohjelmaa niin kauan, että saa siltä vastauksen. Kutsujan on siis pysäytettävä koodinsa suorittaminen aina kutsun tehdessään, ja sen jatko on täysin riippuvainen siitä, miten kutsuttava ohjelma suoriutuu tehtävästään (kuva 1). Etäproseduurikutsu käyttää synkronista kommunikointia hyväkseen. Asynkronisessa muodossa kutsujalle jää kontrollimahdollisuus itselleen. Se lähettää toimintapyynnön ja jatkaa toimintaansa normaalisti (kuva 2). Pyyntö jää niin sanottuun viestijonoon väliohjelmassa, ja kutsuttava koodi suoritetaan sitten kun sen suorittava ohjelma on tilassa jossa se on mahdollista [5, sivu 2]. Asynkronisessa muodossa on kyse viestipohjaisesta kommunikoinnista, ja tämä on yleinen valinta koska se toteuttaa löyhää kytkentää (engl. loose coupling) sovellusten välillä. Sovellusten, järjestelmien ja tietokantojen välinen kommunikointi on oltava asynkronista, jotta jonkin komponentin toimimattomuus, verkossa tapahtuvat viiveet ja muut tilanteet joissa järjestelmän osat eivät voi olla yhtä aikaa toiminnassa, voitaisiin eliminoida.



Kuva 1: Synkroninen viestinkulku [5, sivu 2]



Kuva 2: Asynkroninen viestinkulku [5, sivu 2]

2.3 Viestiformaatit ja datan muoto

2.3.1 Siirtotiedostot

Tiedon siirtämiseen on olemassa useita eri muotoja. Yksinkertaisimmillaan kyseessä voi olla tekstitiedosto, jonka sisältämä tieto tunnistetaan rivin perusteella. Jokaisella rivillä on tietoa, ja sille on määritelty ennalta tietty merkitys. Tämä ei kuitenkaan ole kovinkaan joustava tapa, ja usein käytetäänkin rakenteellisia dokumentteja.

2.3.1.1 XML

XML (engl. eXtensible Markup Language) on merkkauskieli, jonka avulla tiedolle annetaan tarkoitus. Kieli ei määrittele tiedon esitykselle ulkoasua, vaan se jätetään tietoa tulkitsevan osapuolen harteille. Näin saadaan erittäin hyvin siirrettävissä ja muokattavissa olevaa dataa. Perinteiset tiedostomuodot vaativat syvähköä tietämystä kyseisestä formaatista, koska itse tieto ja sen esittämiseen tarkoitetut ominaisuudet on tallennettu binäärimuodossa tiedostoon. Jokaisella tiedostomuodolla on tähän oma tapansa, joten yhden henkilön on raskasta osata tulkita useampaa eri formaattia. XML yksinkertaistaa tilannetta huomattavasti. Se antaa tiedolle merkityksen, esimerkiksi

```
<teksti>
 <otsikko>Diplomityö</otsikko>
 <leipäteksti>Jaarittelua</leipäteksti>
</teksti>
```

Näin jokainen tulkkaaja ymmärtää että kyseessä on tekstipätkä, joka alkaa otsikolla ja jatkuu sen jälkeen leipätekstillä. Se, minkä kokoisella, värisellä ja millä kirjaisimella otsikko ja itse teksti esitetään, on tulkkaajan päätettävissä. Voi myös olla, että XML kuvaa vain esimerkiksi tietokantaan siirrettävää tietoa. Tällöin ulkoasua ei ole olemassa.

Rakenteellisuus tulee esimerkissä esiin <teksti>-tagilla. Sen avulla määritellään yksi tekstielementti, johon otsikko ja leipäteksti kuuluvat. Näin elementtiä voidaan käsitellä yhtenä kokonaisuutena. Nämä ominaisuudet helpottavat tiedon tulkintaa, jolloin sen ohjelmallinen käsittelykin yksinkertaistuu [6, sivu 2].

XSLT (engl. eXtensible Stylesheet Language Transformations) on muunnoskieli jonka avulla voidaan muuntaa XML-tiedostoja erilaisten skeemojen välillä. Yksinkertaisesti tämä tarkoittaa tagien muuttamista jonkin toisen skeeman mukaiseksi, mutta XSLT:llä on mahdollista muodostaa myös täysin erilaisia tiedostomuotoja, kuten PDF tai JPG,

riippuen minkälaista tietoa lähde XML-tiedosto merkkaa. XSLT käyttää W3C:n XPath-kieltä dokumentin sisältämien elementtien navigoimiseen. Myös monille muille kielille on vastaava tuki, jolloin muunnoksia voi tehdä helposti eri tekniikoilla. Tässä työssä muuntamiseen käytetään Python-kieltä, ja sen *xml.dom.minidom*-laajennusta.

2.3.1.2 PLMXML

PLMXML (engl. Product Lifecycle Management XML) on Siemens PLM Softwaren kehittämä avoin ja dokumentoitu XML-skeema, joka on tarkoitettu tarkan tuotetiedon siirtämiseen eri ohjelmien välillä. Tiedostossa voidaan kuvata erilaisia tietoja, kuten ominaisuuksia ja geometria- sekä rakennetietoja. Näin saadaan vaikkapa kokonainen, kolmiulotteinen suunnittelupiirros XML-muotoiseksi ja täten helposti siirrettäväksi. Konkreettisten tuotetietojen lisäksi voidaan tiedostoon tallentaa myös tuotteen elinkaareen liittyvää dataa.

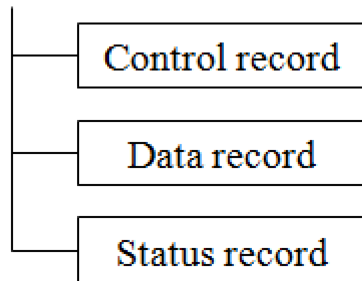
Tiedoston hyödyntämiseksi omissa ohjelmistoissa on olemassa valmiit C++-kirjastot. PLMXML plug-in mahdollistaa skeeman muokkaamisen omiin tarpeisiin, jolloin tiedoston merkkaaman tiedon tarkkuutta voidaan säätää tilanteen mukaan, eikä turhaa dataa tarvitse siirtää. Data adapteri plug-inin avulla voidaan lukea erilaisia tiedostomuotoja ja poimia näistä tarpeellinen tieto (geometria, rakenne, materiaalit) PLMXML-muotoiseksi [7].

2.3.1.3 iDoc

IDoc (engl. Intermediate Document) on toiminnanohjausjärjestelmä SAP:in käyttämä tietorakenne, jonka avulla se voi siirtää dataa joko SAP:in sisällä eri prosessien välillä, tai kolmannen osapuolen ohjelmistojen kanssa. IDoc on määritelty noudattamaan tiettyjä standardeja, kuten EDI (engl. Electronic Data Interchange), ANSI ASC X12 (engl. American National Standards Institute Accredited Standards Committee) ja EDIFACT (engl. Electronic Data Interchange for Administration, Commerce and Transport). Tietorakenteen sisältämien elementtien koko ja formaatti määritellään jonkin edellä mainitun standardin mukaan [8, sivu 908].

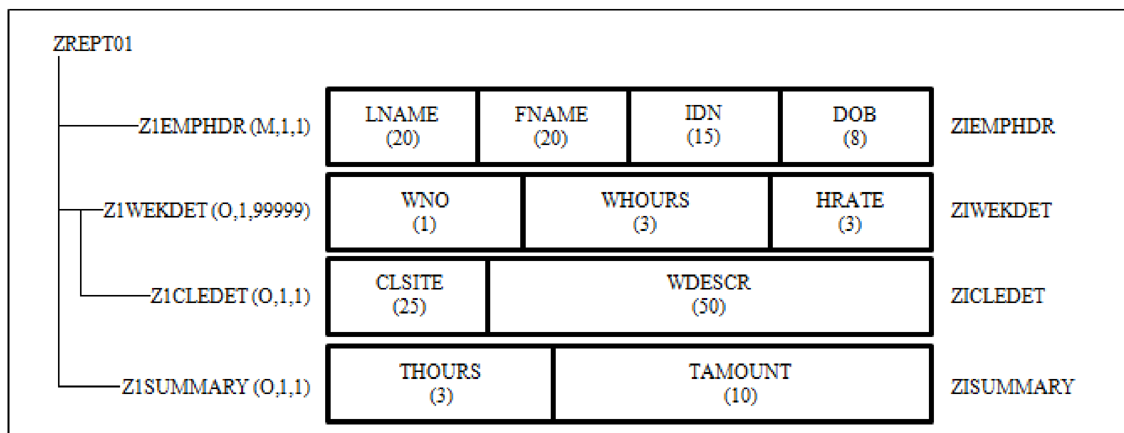
Kuva 3 näyttää, mitä iDoc tietorakenne pitää sisällään. Se koostuu kolmesta osiosta, joita ovat *control*-, *data*- ja *status*–tietueet. Control sisältää tietoa iDoc:in tyypistä, lähettäjästä ja kohteesta. Status sisältää tilannetietoa, kuten onko vastaanottajaa löydetty

ja vaatiiko tiedosto konversioita lähteen ja kohteen välillä. Data-osio sisältää itse siirrettävän hyötykuorman [8, sivu 913].



Kuva 3: SAP:in iDoc formaatin rakenne [8, sivu 913]

Kuvassa 4 esitellään yhden iDoc:in datatietueen rakenne. Tietueen nimi on ZREPT01, ja se sisältää neljä segmenttiä (Z1EMPHDR [=työntekijän tiedot], Z1WEKDET [=viikottaiset tiedot], Z1CLEDET [=mahdollisesti lisätietoja kyseiseltä viikolta] ja Z1SUMMARY [=yhteenveto]). Segmenttien puurakenne kuvaa segmenttien keskinäistä suhdetta toisiinsa, ja segmenttien esittelyn jälkeen olevat kirjaimet tarkoittavat segmentin pakollisuutta (M=pakollinen, O=optionaalinen), ja numerot ilmaisevat kuinka monta kappaletta kyseistä segmenttiä voi sisältyä tähän iDoc rakenteeseen (alaraja, yläraja). Jokainen segmentti sisältää vielä vaihtelevan määrän kenttiä (Z1EMPHDR: LNAME [=sukunimi], FNAME [=etunimi], IDN [=ID], DOB [=syntymäaika]) ja näiden kenttien pituudet on myös määritelty [8, sivu 914].



Kuva 4: SAP:in iDoc-formaatin datatietueen rakenne [8, sivu 914]

2.3.2 Tietokannat tiedonvaihdon välineenä

Yleinen tiedon varastoimismuoto on tietokannat. Integrointia voi tapahtua myös suoraan tällä tasolla, jolloin usea ohjelma lukee samaa tietokantaa ja näin pystyy vaihtamaan dataa keskenään. Tätä kutsutaan myös informaatio-orientoituneeksi integraatioksi.

Tietokannat voidaan integroida myös virtuaaliseksi tietokannaksi. Tällöin eri ohjelmistojen erityyppiset tietokannat yhdistetään siten, että niitä luettaessa saadaan esille näkymä, joka yhdistää kaikki tietokannat. Näin tietokantaa voidaan käyttää, kuin se olisi yksi iso kanta, johon kaikki ohjelmat suorittavat kyselynsä [9, sivu 6]. Tällainen toteutus toimii hyvin, mikäli tietoa ei päivitetä kannassa kovin tiiviisti. Korkea päivitystaajuus aiheuttaa sen, että eri kantojen synkronointi täytyy olla erittäin hyvin suunniteltu, ja tämä voi osaltaan aiheuttaa ratkaisun ongelmallisuuden.

Integroinnissa yritetään myös saavuttaa tietty abstraktoinnin taso, jotta sovelluskerros saataisiin erilleen, mutta suora tietokantaintegraatio ei tätä pysty toteuttamaan. Myös asiakassovellus vaatisi koodissaan tietokannan käsittelyyn liittyviä käskyjä, ja tämä on tietoturvan kannalta riski.

2.4 Integrointiarkkitehtuurit

2.4.1 Point-to-point

Point-to-point oli ensimmäisiä topologioita, joilla lähdettiin ratkaisemaan ohjelmaintegraatiota. Siinä yksinkertaisesti yhdistetään kaikki ohjelmat toisiinsa suoraan, jolloin ne kommunikoivat toistensa API:en kautta. Jokainen suunta vaatii oman yhteytensä, eli kahden ohjelman integrointi vaatii kaksi yhdistämispistettä. Näitä yhteyksiä kutsutaan myös riippuvuussuhteeksi, sillä toisen ohjelman toiminta vaatii myös toisen ohjelman toiminnan [10, sivu 2]. Tämä suhde voi olla joko tiivis tai löyhä. Point-to-point toimii tarpeeksi hyvin kun systeemi on pieni, ja käsittää korkeimmillaan kolmesta viiteen ohjelmaa. Normaalisti yrityksissä on kuitenkin huomattavasti laajemmat järjestelmät, jolloin point-to-point ratkaisusta tulee hyvin nopeasti erittäin monimutkainen ja jopa sotkuinen.

Tiivis kytkentä on usein point-to-point integraation tuloksena syntyvä tila ohjelmien välillä. Systeemin päivittämisen kannalta tämä on epätoivottava asia. Mikäli jonkin moduulin rajapintaa muutetaan, täytyy kaikkien tätä liittymää käyttävien

asiakasohjelmien koodia muuttaa myöskin. Tämä johtuu siis käyttöliittymää hyödyntävän tekniikan ominaisuuksista, ei niinkään itse liittymän toteutuksesta [10, sivu 4].

Kuva 8 esittää, miten toteutus lähtee käsistä kun yhteyksien lukumäärä kasvaa. Esimerkissä on kahdeksan ohjelmaa, ja niiden välille on tehty 11 polkua. Kaikkia ohjelmia ei ole vielä edes yhdistetty, mutta silti monimutkaisuus näkyy selvästi kuvaa katsomalla. Tätä näkymää kutsutaankin spagetiksi, joka hyvin kuvaa lopputuloksen kompleksisuutta.

Kuva 8: Point-to-point integroitu järjestelmä [10, sivu 3]

Kuvassa tiedon siirtosuuntaa kuvataan nuolen avulla (esimerkiksi CC -> BILL on yhteys joka siirtää dataa CustomerCaresta BillingSystemiin), ja pallot ilmaisevat integraatiopisteen.

2.4.2 EAI

EAI (engl. Enterprise Application Integration) on integrointiin suunnattu kehitysalusta. Se kokoaa yhteen erilaisia tekniikoita ja palveluja joiden avulla voidaan toteuttaa laajoja yhdistämisprojekteja, joissa integroidaan yrityksen koko liiketoimintaprosessiin liittyvät ohjelmat. Koska kyseessä on pienempiin osakomponentteihin hajautettu järjestelmä, on hyväksikäytettävä tekniikoita jotka ratkaisevat tämän kaltaisen arkkitehtuurin esiintuomia ongelmia. Näitä ovat:

1. Yhteentoimivuus: järjestelmän eri komponentit käyttävät mahdollisesti eri käyttöjärjestelmiä, tiedostoformaatteja ja ne on toteutettu eri ohjelmointikielillä.
2. Dataintegraatio: Modulaarisen, hajautetun järjestelmän toimivuudeksi on tärkeää kehittää yhtenäinen tapa tiedon kuljettamiseksi eri ohjelmien välillä, jotta tiedon yhtenäisyys toteutuisi.
3. Kestävyys, vakaus, laajennettavuus: kolme tärkeintä ominaisuutta jotka ylläpitävät modulaarista infrastruktuuria. [11]

Markkinoilla on useiden valmistajien EAI-ratkaisuja, ja näistä jokainen tarjoaa omanlaisensa vastauksen. Näillä kaikilla on kuitenkin tarkoitus parantaa tuottavuutta pääasiassa samoin keinoin. EAI-paketteihin on yleensä sisällytetty sisäänrakennetut API:t, joiden avulla ne voivat kommunikoida yhdistämiensä ohjelmien kanssa, ja näin saadaan eliminoitua tarve eri ohjelmien välisille yksittäisille point-to-point yhteyksille. Näille eri tietolähteille (moduuleille, ohjelmistoille) muodostetaan yhtenäinen datamuoto, tai datalle yhtenäinen sijaintipaikka, jolloin kyseinen informaatio saadaan käyttöön yhden työkalun avulla. Tämä auttaa yksinkertaistamaan monia liiketoimintamalleja, sekä vähentää loppukäyttäjien vaatimaa koulutusta, koska koko järjestelmä voidaan toteuttaa käytettäväksi yhden käyttöliittymän kautta [12].

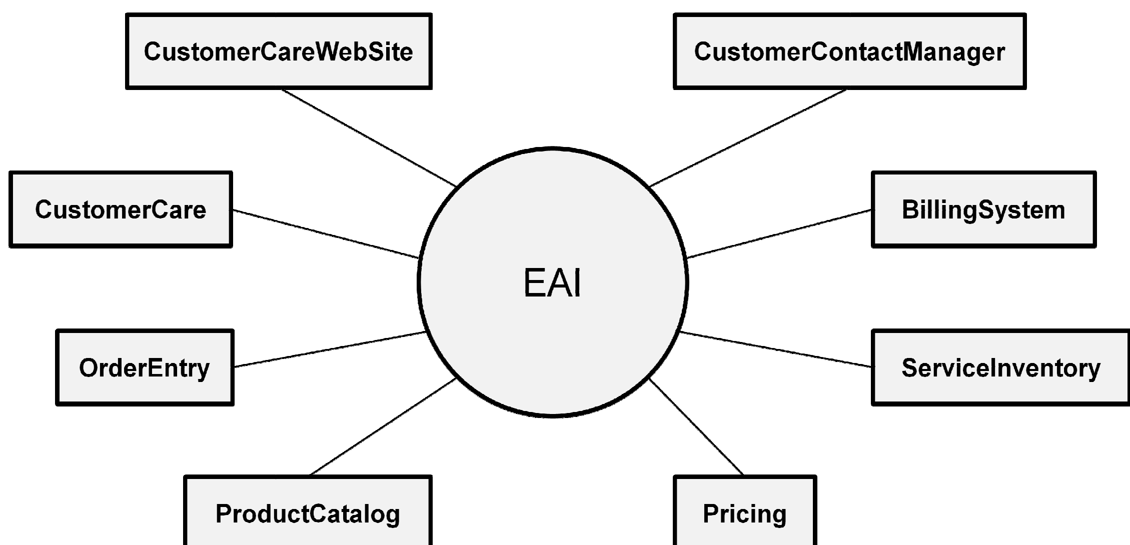
Yrityksen jokaiselle henkilölle ja prosessille on taattava resurssien saatavuus, vain näin saadaan tietojärjestelmästä täysi hyöty irti. Henkilöt koostavat osastoja, joilla jokaisella on oma tehtävänsä. Osastoilla on siis omat tietojärjestelmänsä, ja yksi EAI:n tarkoituksista on näiden prosessien yhdistäminen [12].

Käytössä olevien ohjelmien ja datan perusteella voidaan muodostaa sääntöjä sekä liiketoimintatapoja. Nämä voidaan toteuttaa myös EAI-järjestelmään ja sitä myötä

uusiin järjestelmään integroitaviin sovelluksiin, jolloin ohjelmien lisääminen helpottuu [12].

Kuvassa 9 esitetään miten esimerkkijärjestelmämme moduulit on yhdistetty toisiinsa EAI-periaatteen mukaisesti. Kyseessä on niin sanottu *hub-and-spoke*–malli (voi olla myös *spoke-and-hub*), jossa keskellä toimiva väliohjelma hoitaa eri ohjelmien välisen liikenteen [10, sivu 5]. Näin saadaan tiedonsiirto, tiedon muokkaus ja muut funktiot keskitettyä yhteen pisteeseen, eikä jokaisella ohjelmalla tarvitse olla erikseen omaa yhteyttä jokaiseen muuhun ohjelmaan. Tämä on mahdollista, koska viestin lähteellä ja kohteella ei tarvitse olla toisistaan minkäänlaista tietoa, vaan väliohjelma hoitaa reitityksen oikeaan osoitteeseen.

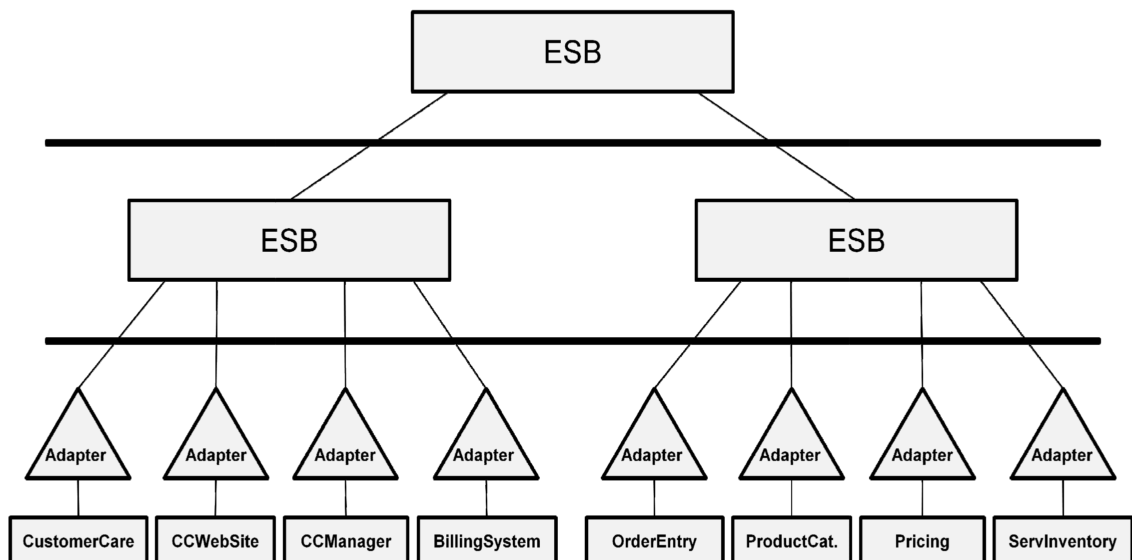
Kuvan 9 ratkaisu on periaatteessa vielä point-to-point integraatio, sillä jokainen ohjelma on yhdistetty toisiinsa väliohjelman kautta. Tästä huolimatta arkkitehtuuri selventää täysiverisen point-to-pointin spagettimaisuutta mahdollistamalla reittien vedon yhden pisteen kautta. Näin keskitetty ratkaisu yksinkertaistaa niin toteutusta kuin lopputulostakin, mutta tuo muassaan myös uusia ongelmia. Ainoastaan keskipisteen täytyy kaatua ja koko järjestelmä on alhaalla. Väliohjelma muodostaa myös pullonkaulan viestien reitille, ja tiedon siirtyminen hidastuu kuorman ollessa liian suuri.



Kuva 9: EAI arkkitehtuuri järjestelmässä [10, sivu 5]

2.4.3 ESB

ESB (engl. Enterprise Service Bus) kehitettiin antamaan vastaus EAI:sta löytyneisiin huonoihin ratkaisuihin. Suurin ongelma lienee EAI:n käyttämä yhden pisteen integraatio, jossa väliohjelma toimii heikkona lenkkinä. ESB ratkaisee tämän ongelman siirtämällä integrointipisteen keskeltä arkkitehtuurin sivuille. Kuva 10 kuvaa hierarkista ESB-arkkitehtuuria, jossa integroinnin vaatimat toimenpiteet suoritetaan jokaisen sovelluksen kohdalla erillisissä adaptereissa. Nyt väylän tehtäväksi ei jää kuin viestin siirtäminen järjestelmässä, adapterit ovat tehneet vaadittavat muutokset viestille, jolloin sille on saatu järjestelmän yhteistä kieltä mukaileva sisältö. Kuvan 10 ratkaisun hierarkisuus esiintyy käytännössä niin, että jokainen yrityksen sisäinen osasto voi abstraktoida toimintansa omaan mustaan laatikkoonsa [10, sivu 6]. Näin saadaan osastojen toiminta piilotettua kokonaisuuden kannalta, eikä korkealla tasolla toimiessa tarvitse tietää yksittäisten sovellusten toteutuksesta, vaan koko osasto voidaan mieltää yhtenä ”toimintona”.



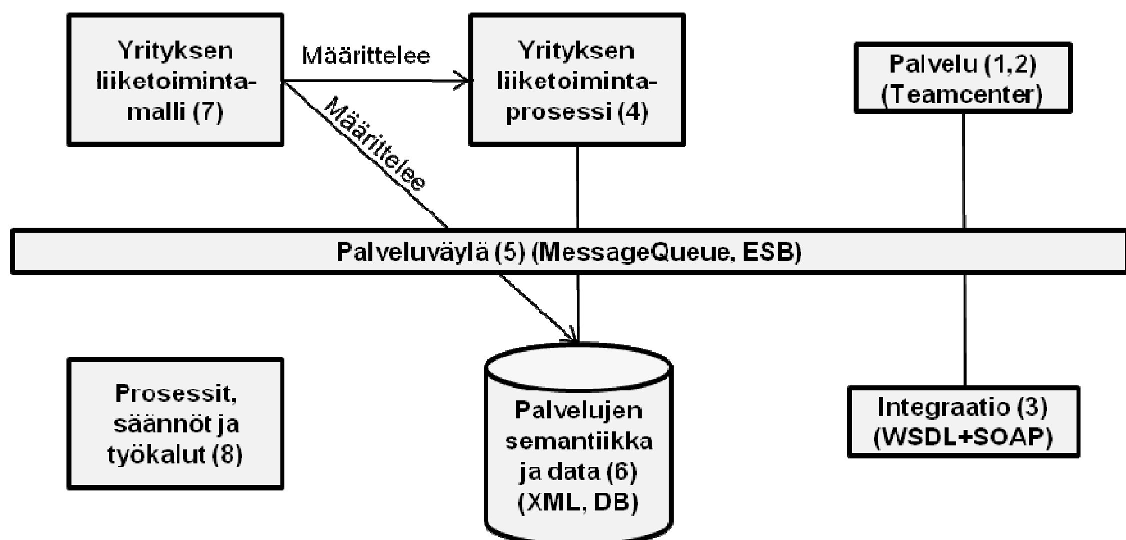
Kuva 10: Hierarkinen ESB [10, sivu 7]

Tässä työssä käytettävä väliohjelma, IBM WebSphere MessageQueue, käsitetään täysiveriseksi ESB-kokonaisuudeksi. Sen avulla voidaan rakentaa erilaisia adaptereja sovelluksille, reitittää viesti toiselle sovellukselle ja ennen sovellukselle vientiä rakentaa jälleen uusi adapteri viestille.

2.4.4 SOA ja palvelut

Palveluarkkitehtuuri (engl. Service Oriented Architecture, SOA) on 1990-luvulla alkaneen ohjelmistointegroinnin kulminoituma. Sen sijaan, että järjestelmään kuuluvat sovellukset ja niiden tarjoamat toiminnot käsitettäisiin teknisinä komponentteina, matalalla tasolla riveinä koodia ja tietokantatauluja, kuvataankin ne palveluina jotka ovat osa yrityksen toimintaprosessia. Näin saadaan erotettua toteutus sen mahdollistamasta toiminnosta, ja integrointia voidaan suunnitella korkeammalla tasolla, kokonaisena toimintamallina. Tämä vähentää ohjelmien keskinäistä riippuvuutta, ja sen seurauksena myös järjestelmän eri osien päivittäminen ja vaihtaminen yksinkertaistuu.

Kuva 5 esittää palveluarkkitehtuurin määrittelyyn ja toteutukseen liittyvät seikat niin palveluiden kuin tekniikan kannalta. Siihen liittyvät kahdeksan kohtaa on käyty tarkemmin läpi kuvan jälkeen.

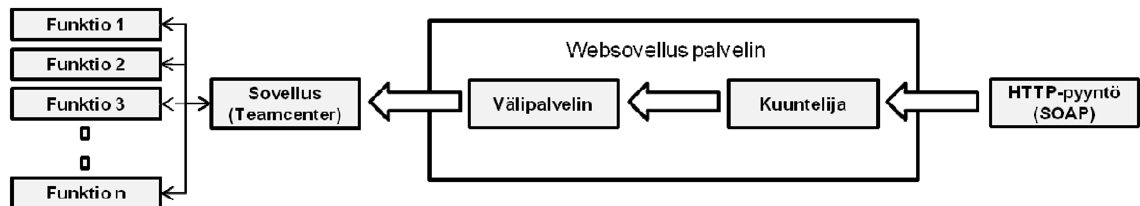


Kuva 5: Palveluarkkitehtuurin määrittely [13, sivu 37]

1. Palveluiden määrittely, minkälaisia toimintoja järjestelmä tarjoaa ja mitkä niistä voidaan kuvata palveluksi.
2. Palveluiden toteuttamisen ja käytön määrittely, miten ja mihin palveluita käytetään, miten laaja toiminto voidaan määritellä yhdeksi palveluksi ja mitä toimintoja palvelun käynnistäminen vaatii. Tässä työssä kohdan 1 ja 2 palvelut ovat Teamcenterin tarjoamia funktioita.
3. Sovellusten integrointi palveluympäristöön, yleiskäyttöinen tapa miten eri toiminnot kuvataan palveluiksi ja miten näitä tarjoavat sovellukset yhdistetään palveluväylään ja toisiinsa. Teamcenter käyttää tämän toteuttamiseen WSDL- (engl. Web Services Description Language) ja SOAP-tekniikoita (engl. Simple Object Access Protocol).
4. Palveluiden yhdistäminen kokonaisuudeksi, miten palveluista saadaan yksi kokonaisuus, josta saadaan yksi suurempi, yrityksen koko toimintamallia tukeva laaja osakokonaisuus.
5. Teknisen infrastruktuurin määrittely, miten sovellukset integroidaan toisiinsa ja miten ne kommunikoivat keskenään. Konecranesin ympäristössä palveluväylänä toimii WebSphere väliohjelma, jonka avulla infrastruktuuri noudattaa ESB:tä.
6. Palveluiden yhteisen semantiikan ja datan määrittely, mitä tietoa on siirrettävä eri palveluiden välillä ja mikä sen merkitys on koko liiketoimintamallin kannalta. Tässä työssä data määritellään XML-muotoiseksi sovellusten välisessä liikenteessä, ja tiedon lopullinen sijainti on sovellusten tietokannoissa.
7. Linjataan palvelut yrityksen liikestrategian ja tavoitteiden kanssa jotta se tukisi koko liiketoimintamallia.
8. Lopuksi määritellään koko arkkitehtuurin toiminta, sen käyttöön liittyvät säännöt ja työkalut. [13, sivu 37]

Web-palvelut (engl. web services) on keskeinen käsite integraatiossa. Se käsittää tapoja, joilla sovelluksen toimintoja voidaan käyttää verkon yli avointen protokollien avulla. Näin saadaan periaatteessa normaalista ohjelmistosta verkkosovellus, jolloin sen funktionaaliset toiminnot saadaan erotettua omaan käyttöliittymäänsä itse sovelluksesta. Yleensä web-palvelun kautta käytetään jotain tiettyä toimintoa, eikä sitä ole tarkoitettu niinkään täydelliseksi käyttörajapinnaksi jollekin tietylle ohjelmalle, vaan että sen avulla koottaisiin eri sovellusten toimintoja yhteen. Näin saadaan kierrätettyä koodia, eikä samoja toimintoja tarvitse toteuttaa jokaiseen niitä tarvitsevaan sovellukseen.

Ajatellaan jotain tiettyä sovellusta X, joka koostuu pääohjelmasta ja erilaisista alifunktioista, joita pääohjelma kutsuu. Tilanne kuvaa hyvin asiakas-palvelin–mallin, jossa pääohjelma käyttäytyy asiakkaana, joka kutsuu palvelimelta palveluja, käyttäen palvelimen julkisia rajapintoja toimintojen käynnistämiseksi. Tämän periaatteen pohjalta voidaan web-palveluita soveltaa tässä työssä käsiteltävään aiheeseen. Kuva 6 esittää, miten palvelua kutsutaan Teamcenteriltä. Kuuntelija on komponentti, joka vastaanottaa palvelupyynnön esimerkiksi HTTP-protokollan (engl. HyperText Transfer Protocol) avulla SOAP-muodossa. Välipalvelin muuttaa pyynnön sovelluksen ymmärtämäksi koodiksi, joka taas kutsuu sen perusteella jotain funktioistaan ja palauttaa vastauksen samaa polkua pitkin. Web-palvelut osaavat toteuttaa siis viestipohjaisen kommunikoinnin muodon esimerkiksi SOAP:in avulla. Web-palvelut käsittävät myös etäproseduurikutsut SOAP:in avulla.



Kuva 6: Web-palveluiden toimintalogiikka

2.5 WSDL ja SOAP

WSDL on kieli, jonka avulla voidaan kuvata web-palvelu XML–dokumenttina. Se kuvaa palvelun keskeisimmät ominaisuudet, kuten sen mitä palvelu tekee, missä se sijaitsee ja kuinka palvelun voi käynnistää. WSDL-dokumentti määrittelee palvelut portteina (engl. port, endpoint). Viesti (engl. message) sisältää abstraktin kuvauksen siirrettävästä datasta, ja porttityyppi (engl. port type) kerää yhdeksi kokonaisuudeksi eri toimintoja (engl. operations). WSDL–dokumentti määrittelee täten palvelun seitsemällä eri elementillä:

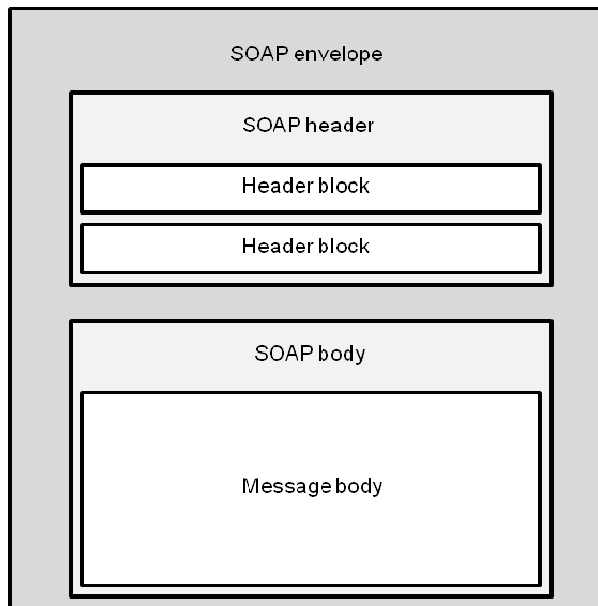
1. Tyypit (engl. types): määrittelee mitä datatyypin määritelmiä (engl. Data Type Definition, DTD) on käytössä, kuvaa palvelun rajapintaa
2. Viesti (engl. message): abstrakti määritelmä siirrettävästä datasta, kuvaa palvelun rajapintaa

3. Toiminto (engl. operation): kuvaus palvelun tukemasta toiminnosta, kuvaa palvelun rajapintaa
4. Porttityyppi (engl. port type): kokoelma toimintoja, jotka vähintään yksi portti toteuttaa, kuvaa palvelun rajapintaa
5. Sitoutuminen (engl. binding): määrittelee tietyn porttityypin protokollan ja datamuodon, esimerkiksi HTTP tai SMTP (engl. Simple Mail Transfer Protocol) ja XML
6. Portti (engl. port): verkko-osoitteen ja sitoutumisen avulla määritetty fyysinen määränpää datalle, esimerkiksi URL (engl. Uniform Resource Locator)
7. Palvelu (engl. service): kokoelma toisiinsa liittyvistä porteista, jotka muodostavat yhdessä jonkin palvelun. [14, sivu 5] [15, sivu 50]

Kun palvelu on kuvattu WSDL:än avulla, tiedetään, miten palvelua voidaan käyttää. Tämän jälkeen palvelut saadaan keskustelemaan keskenään SOAP:ia käyttämällä. SOAP on protokollastandardi, jonka avulla voidaan lähettää ja vastaanottaa viestejä Internet ympäristössä. Se tukee löyhää kytkentää palvelujen välillä, jolloin prosessien itsenäisyys toteutuu, eikä palvelun käyttäjällä tarvitse olla tietoa käytetystä ohjelmointikielestä, käyttöjärjestelmästä tai fyysisestä alustasta. Yleisin käytetty protokolla on HTTP ja dataformaatti XML [14, sivu 5].

Kuvassa 7 näemme SOAP-viestin rakenteen. Viesti on koottu yhteen niin sanottuun kirjeeseen (engl. envelope), jonka pääosat ovat *header* ja *body*. Vapaaehtoisen headerin sisältö koostuu lohkoista (engl. block) jotka sisältävät tietoa siitä, miten viestiä tulisi käsitellä (tietoturvamääritykset, kohdeosoitteet). Lohkoja voidaan lisätä vapaavalintaisesti tarpeen mukaan, joten viestivuon jokaiselle solmulle saadaan määriteltyä omat vaatimukset. Näin saadaan lisättyä viestille vastaanottajia ilman, että viestin koko rakennetta täytyisi muokata uusiksi [16, sivu 28].

Itse siirrettävä tieto sisältyy SOAP body–osion *message body*–lohkoon. Viesti on yleisimmin XML-muotoista dataa prosessoinnin helpottamiseksi, mutta myös binääridataa voidaan enkoodata tekstimuodossa mukaan [16, sivu 28].



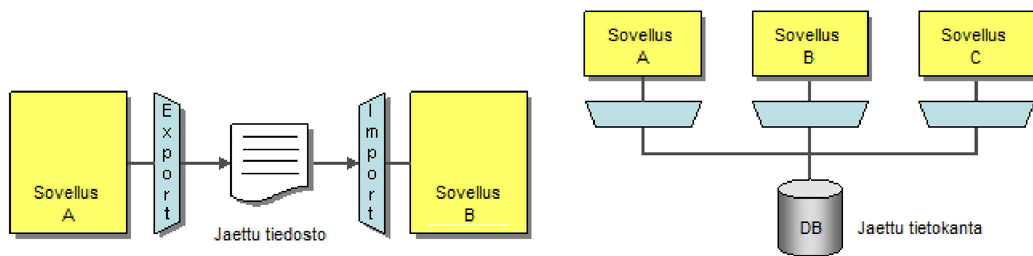
Kuva 7: SOAP viestin rakenne [17, sivu 13]

SOAP:ia voidaan käyttää kahdessa eri toimintamuodossa, sen avulla voidaan lähettää etäproseduurikutsuja ja se voi toimia viestipohjaisena kommunikaattorina, joten SOAP tukee niin tiiviisti (engl. tight coupling) kuin löyhästi kytkettyjä sovelluksia [16, sivu 28].

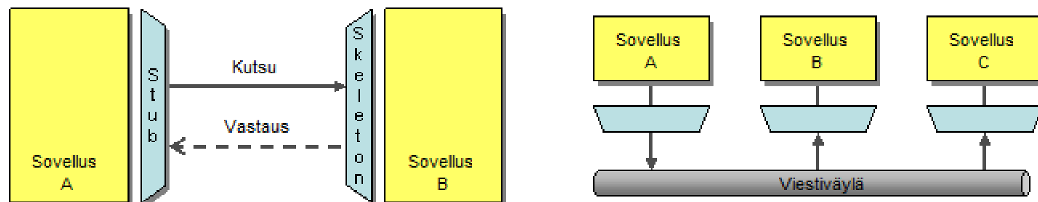
2.6 Integraation suunnittelumallit

Integraation suunnittelumallit (engl. Enterprise Integration Patterns, EIP) on kokoelma yleisiä, integraatioprojekteissa esiintyviä tapahtumaketjuja tai usein toistuvia ratkaisuja. Niiden avulla pyritään helpottamaan integraatioarkkitehtuurin suunnittelua tarjoamalla tapa kuvata malli korkealla tasolla. Graafeilla kuvataan sovellusten välistä tiedonsiirtoa viestinvälityksen avulla. Nuoli ilmaisee siirron suunnan, ja lähde- sekä kohdesovellusten välillä voidaan suorittaa erilaisia reitityksiä ja muunnoksia viestille. Viesti voi sisältää kohteeseen tallennettavaa dataa, tai käskyn käynnistää jonkin kohteen funktioista. Vastauksena saadaan siten joko tieto tallennuksen onnistumisesta, tai suoritetun funktion ulostulo. Ratkaisuja on olemassa kymmeniä ja niitä tunnistetaan kokoajan lisää, mutta olemassa olevat voidaan kategorioida seuraaviin ryhmiin:

1. Integraatiotyylit (engl. integration styles): eri tavat, joilla sovellukset voidaan integroida. Näitä on olemassa neljä tapaa. Ensimmäinen on tiedoston siirto, jossa sovellukset tuottavat tiedostoja ja lukevat toisten sovellusten tuottamia tiedostoja (kuva 11). Jaettu tietokanta käyttää tiedon välityksessä samaa tietokantaa, johon sovellukset tallentavat tietoa ja suorittavat lukunsa (kuva 11). Etäproseduurikutsu käyttää sovellusten julkisia rajapintoja niiden toimintojen kutsumiseksi (kuva 12). Viimeisenä on vielä viestitys, jossa sovellukset liitetään yhteiseen viestiväylään, jonka kautta kommunikointi tapahtuu (kuva 12) [18].

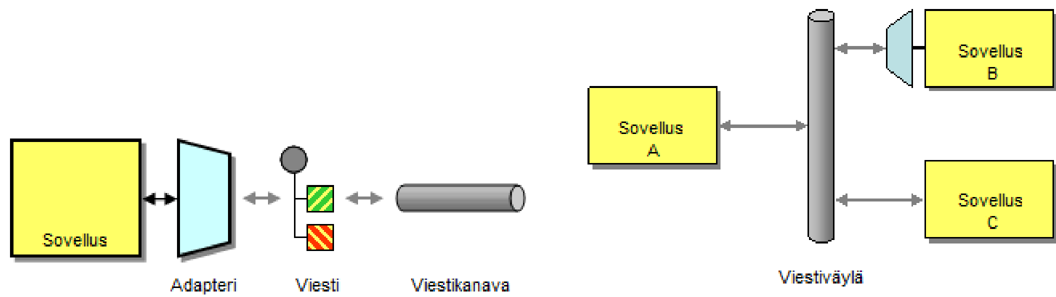


Kuva 11: Tiedon välitys yhteisen tiedoston ja jaetun tietokannan kautta [18]



Kuva 12: Tiedon välitys etäproseduurikutsun ja viestiväylän avulla [18]

2. Kanavamallit (engl. channel patterns): viestisysteemin perimmäiset ominaisuudet, jotka suurin osa kaupallisista ratkaisuista toteuttaa. Näistä esimerkkeinä kuvassa 13 kanava-adapteri, joka osaa käyttää sovelluksen API:a tuodakseen ja viedäkseen tietoa viestien muodossa kanavaan, sekä viestiväylä, jonka avulla sovellukset voidaan yhdistää toisiinsa löyhästi kytkien [18].

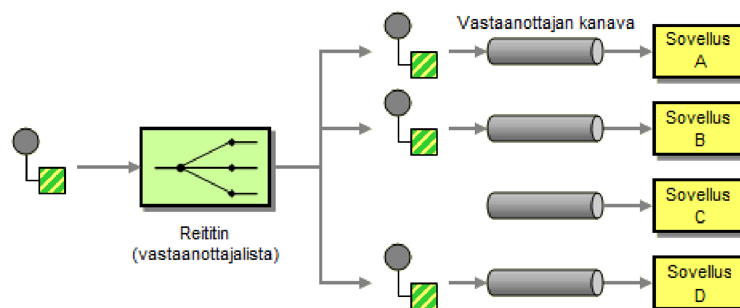


Kuva 13: Adapteri sovelluksen ja kanavan välillä, sekä viestiväylä [18]

3. Viestin muodostamisen mallit (engl. message construction patterns): kuvaa järjestelmän läpi kulkevan viestin tarkoituksen, muodon ja sisällön. Näitä voivat olla esimerkiksi käskyviesti jonka avulla voidaan käynnistää sovelluksessa proseduuri, tai dokumenttiviesti joka sisältää siirrettävää dataa (kuva 14) [18].

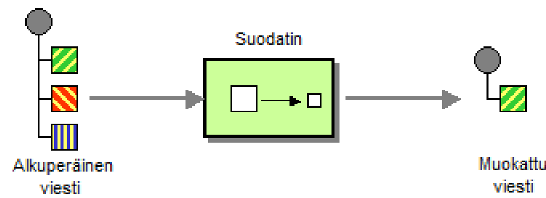
Kuva 14: Käsky- ja dokumenttiviestit [18]

4. Reititysmallit (engl. routing patterns): millä tavoin viestejä voidaan ohjata lähettäjältä oikealle vastaanottajalle käyttäen erilaisia ehtoja. Kuvassa 15 on esitetty yksi yleinen malli (vastaanottajalista, engl. recipient list), jossa viesti reititetään oikealle sovellukselle siirtämällä se ensin kyseiselle sovellukselle varattuun kanavaan [18].



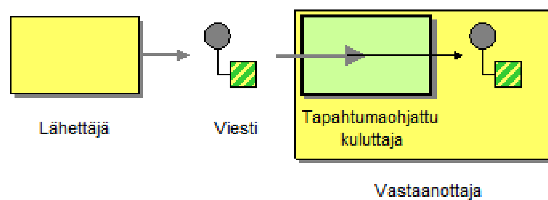
Kuva 15: Viestin reititys oikealle sovellukselle [18]

5. Muunnosmallit (engl. transformation patterns): mallit muuttavat viestin sisällön muodosta toiseen, mikäli päätepisteiden sovellukset käyttävät erilaisia formaatteja. Myös viestin sisältöä ja rakennetta muokataan näillä malleilla, kuten kuvassa 16 nähtävä suodatin, jossa ylimääräinen tietosisältö poistetaan [18].



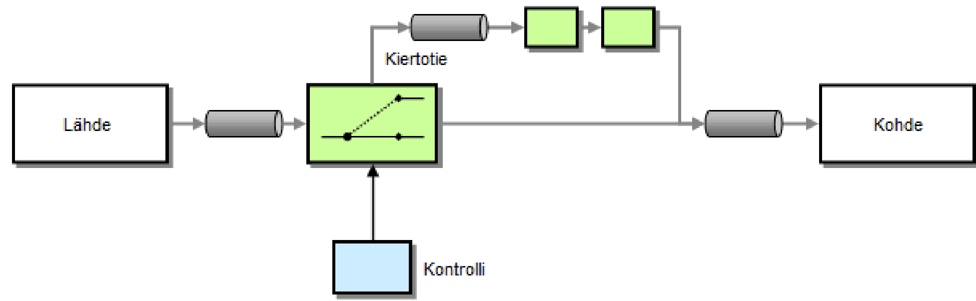
Kuva 16: Viestin sisällön muokkaus [18]

6. Päätepisteiden mallit (engl. endpoint patterns): kuvaavat viestijärjestelmään liitettyjen asiakkaiden (sovellusten) toimintaa sen kannalta, miten ne tuottavat ja käyttävät viestejä. Kuvassa 17 on kuvattu jonkun tietyn tapahtuman laukaisemana tapahtuva viestin vastaanotto (engl. event-driven consumer). Kyseessä voi olla esimerkiksi palvelimelle ilmestyvä tiedosto, jonka vastaanottava sovellus huomaa ja ottaa talteen [18].



Kuva 17: Tapahtumaohjattu kuluttaja [18]

7. Järjestelmänhallinnan mallit (engl. system management patterns): tavat joilla monimutkainen viestijärjestelmä pysyy toiminnassa. Järjestelmä tuottaa, käyttää ja siirtää lukemattomia viestejä päivässä, ja tämä on tietysti hyvin altis erilaisille virheille, pullonkauloille ja muutoksille. Kuva 18 esittää yhden tavan miten viestien oikeellisuutta voidaan tarkkailla järjestelmässä. Toteutetaan kiertotie viestille, jonka kautta mennessään käydään ylimääräisiä testejä läpi, ja näiden perusteella voidaan sisältöä tutkia [18].



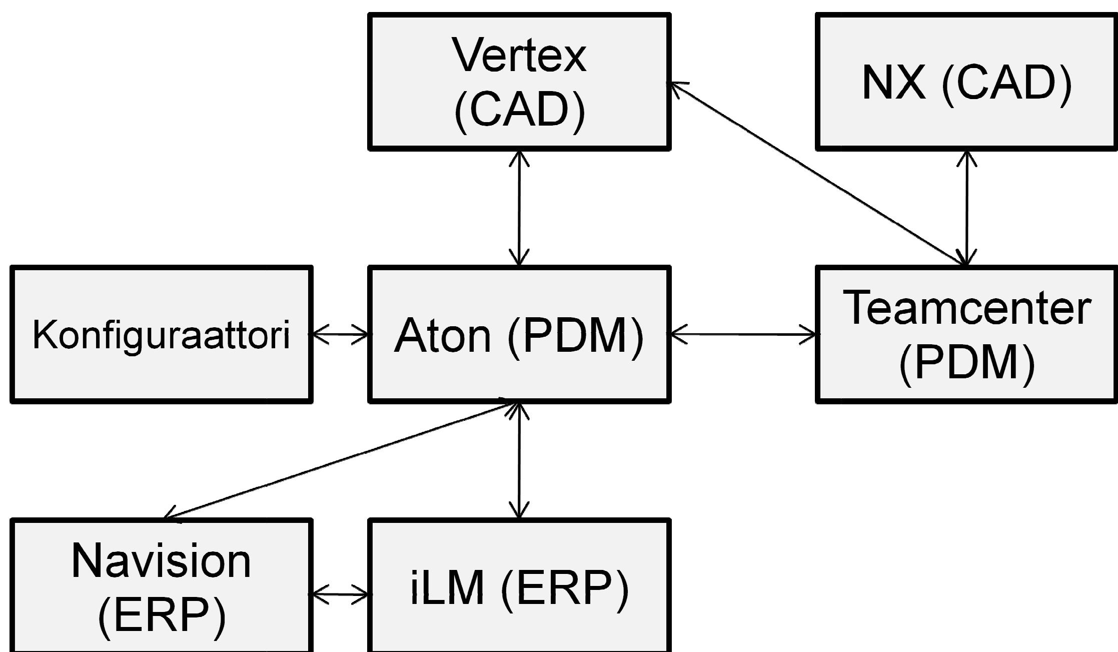
Kuva 18: Viestin tarkitusreitti [18]

3 Järjestelmän kuvaus ja projektin tavoitteet

Tässä luvussa käydään läpi tarkemmin Konecranesin käyttämän tietojärjestelmän rakennetta, siihen liitettyjä ohjelmistoja, niiden välistä viestiliikennettä sekä selvitetään miten tämä työ liittyy niihin.

3.1 Nykytilanne

Kuvassa 21 on esitetty korkealla tasolla miten eri ohjelmat on liitetty toisiinsa, ja mitkä ohjelmat vaihtavat tietoa keskenään.



Kuva 21: Eri ohjelmien väliset yhteydet järjestelmässä tällä hetkellä

Vanhat moottoripiirrokset ovat 2D-muodossa ja tehty Vertexillä, kun taas uudemmat projektit on toteutettu NX:llä 3D-muodossa. CAD-ohjelmista (engl. Computer Aided Design) viedään tuote siihen liitettyyn PDM:ään johon saadaan kuvasta rakenne nimiketasolla. Vanhat kuvat siirtyvät Vertexin ja Teamcenterin välillä, mutta 3D piirrosten rakenne viedään vain Teamcenteriin.

Vanhan ja uuden PDM:än toimiessa rinnakkain on AtonPDM valittu niin sanotuksi *item masteriksi*. Tämä tarkoittaa sitä että nimiketiedot on haettava Atonista, ja myös Teamcenterissä tehdyt muutokset täytyy päivittää Atoniin.

Kun moottorista tehdään tilauspyyntö, se saapuu ensimmäiseksi toiminnan-ohjausjärjestelmä iLM:ään. Tästä tieto kulkee alihankkijan ERP:iin Navisioniin, joka kuittaa sen lähettämällä tilausvahvistuksen iLM:ään. Sille mitä seuraavaksi tapahtuu, on olemassa kolme vaihtoehtoa:

1. Pyydetty rakenne on moottori: konfiguroidaan ja lähetetään rakenne
2. Ei moottori, Navisionissa jo rakenne: tietoa ei siirry Atonista
3. Ei moottori, Navisionissa vanhentunut rakenne: lähetetään uusi rakenne Atonista

Aina kun alihankkija pyytää moottorille rakenteen, käynnistyy konfiguraattori, jotta lähetetty rakenne olisi ajan tasalla. Mikäli valintasääntöjä on päivitetty, moottorin attribuutteja korjattu tai kyseessä on täysin uusi moottori, jolla ei ole vielä rakennetta, niin näin ei siirry väärää tietoa. Navision voi myös pyytää muita kuin moottorinimikkeitä, kuten mekaanisia osaluetteloja tai varaosia. Näitä ei kuitenkaan konfiguroida, vaan niiden rakenteet on siirretty automaattisesti vanhasta järjestelmästä, tai ne on tehty manuaalisesti. Näissä tapauksissa tarkistetaan Atonista nimikkeen muokkauspäivä, verrataan sitä Navisionissa olevaan ja päätetään sen perusteella tapahtuuko siirtoa vai ei.

Tämän prosessin jälkeen lähetetään vielä pakkauslista iLM:ään. Kun moottori on valmistettu, tekee alihankkija sille vielä tietyt testit, ja näiden tulokset siirretään Atoniin ja sen kautta iLM:ään.

3.1.1 Integraation kohteet

3.1.1.1 PDM

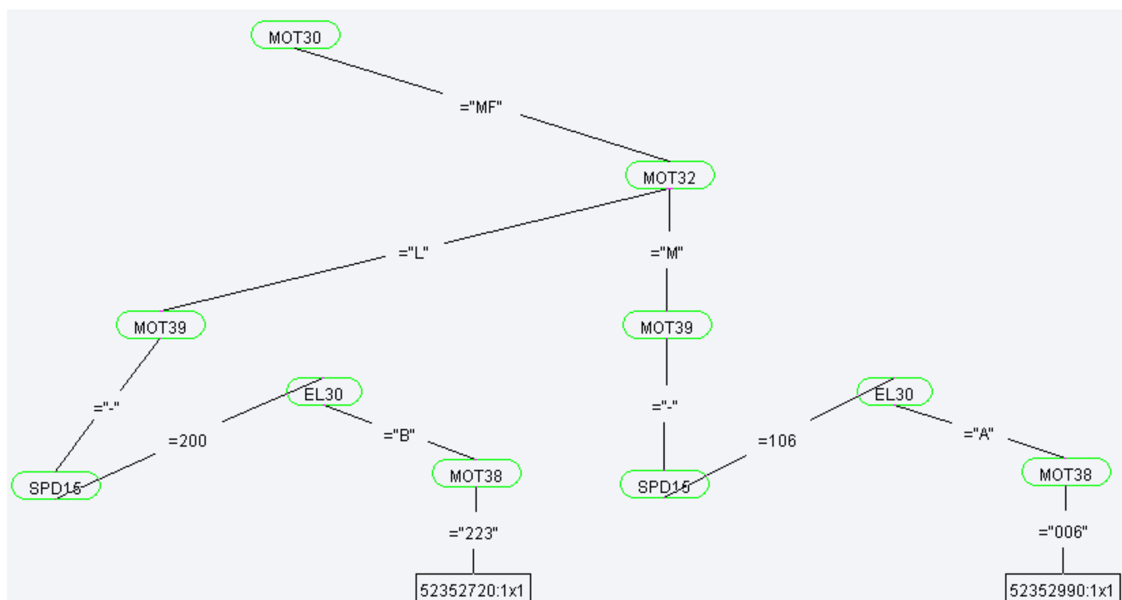
Tuotetiedonhallinnan avulla pystytään hallitsemaan yrityksen tuotteisiin liittyvää informaatiota, kuten tuoterakenteita sekä niihin liittyvien komponenttien ominaisuuksia. Myös tuotteiden valmistukseen vaadittavat dokumentit, piirustukset ja asiakkaille toimitettavat käyttöohjeet löytyvät järjestelmästä.

Platform käsitteen mukainen tuoteperheiden määrittely on tärkeä osa PDM:ää. Näin voidaan erilaisten asiakkaan antamien vaatimusten pohjalta konfiguroida tuote automaattisesti. Eri tuotemalleille on rakennettu niin sanotut valintasäännöt, jonka pohjalta konfiguraattori pystyy annettujen attribuuttien perusteella rakentamaan halutun mukaisen tuotteen.

Kuva 19 esittää osan yhden moottorimallin jarrusäännöstä. Otetaan moottorin esimerkkinimikkeeksi MF16L-200N147B12223T-IP55. Tämän nimikkeen takana PDM säilyttää attribuutteja, jotka ovat suoraan luettavissa myös itse nimikkeestä. Konfiguraattori lukee näitä tietoja ja hakee sen perusteella jarrun moottorille. Esimerkissä moottorimerkki hajoitetaan taulukon 1 mukaisesti, ja näiden arvojen perusteella kuljetaan puussa vasemmanpuoleista haaraa.

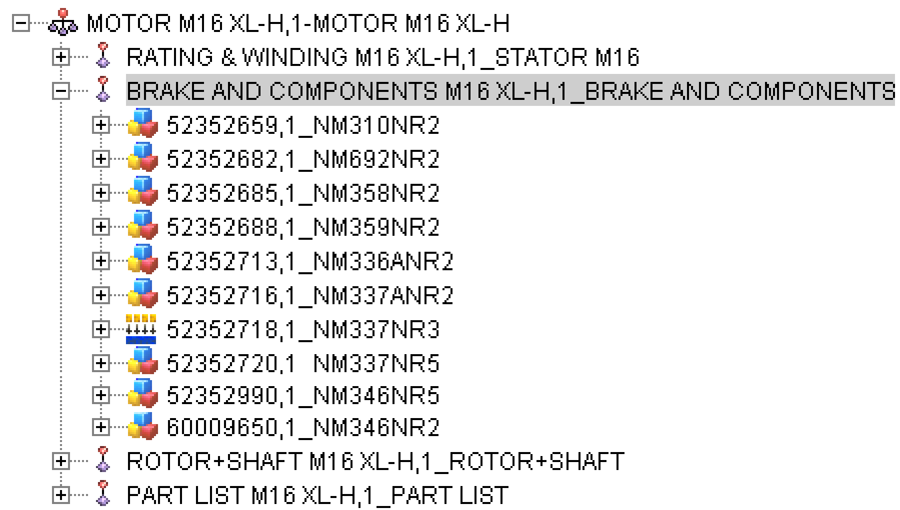
Taulukko 1: Moottorin attribuutit

MF	16	L	-	200	N	147	B	1	2	223	T	-	IP55
MOT30	MOT31	MOT32	MOT39	SPD15			EL30	MOT40		MOT38			



Kuva 19: Moottorin jarrun valintasääntö

Näin konfiguroituisi kyseisen moottorin jarruksi nimike, jonka ID numero on 52352720, ja tämä löytyy jarrusääntöön liitetystä nimikelistasta (kuva 20). Samasta kuvasta käy ilmi myös moottorin rakenne muilta osin. Sisältö näille haetaan jarrujen kaltaisesti.



Kuva 20: Moottorin jarrun nimikkeet

Nykyinen käytössä oleva tuotetiedonhallintajärjestelmä on suomalaisen Modultekin AtonPDM, ja edellä olevat konfigurointiesimerkit on otettu tästä ohjelmasta. Uutena korvaavana järjestelmänä tulee olemaan Siemensin Teamcenter.

3.1.1.2 ERP

ERP eli toiminnanohjausjärjestelmä on laaja ohjelmistokokonaisuus (luvussa kaksi mainittu pakattu ohjelmisto), jonka tarkoituksena on mukailla yrityksen liiketoimintamalleja ja näin yksinkertaistaa ja automatisoida erilaisia sisäisiä prosesseja. Konecranesilla poistuvana järjestelmänä toimii iLM, ja korvaavana ERP:nä tulee SAP.

Toiminnanohjausjärjestelmän perimmäinen tavoite on ennustaa ja tasapainottaa kysyntää ja tarjontaa yrityksen eri resurssien osalta. Se tarjoaa laajan kokoelman työkaluja suunnitteluun ja aikataulutukseen. Nämä työkalut yhdistävät toimittajan ja asiakkaan koko toimitusprosessiin, tarjoaa hyviksi todettuja, ennalta määriteltyjä tapoja päätöksentekoon sekä ohjaa myyntiä, markkinointia, logistiikkaa, ostoa ja tuotekehitystä. Tämän kaiken lopputuloksena on tarkoitus saavuttaa korkean tason asiakaspalvelu ja tuottavuus. Käytännössä tähän päästään järjestelmän tuottamien suunnitelmien ja aikataulujen ollessa optimoituja niin, että oikeat resurssit kuten työvoima, materiaalit ja koneet ovat saatavilla tarvittavissa määrin oikeaan aikaan. [19, sivu 19]

3.1.1.3 CAD

Tietokoneavusteinen suunnittelu tarkoittaa tässä tapauksessa moottorien mekaanisten piirustusten tekemistä. Uusille tuotteille annetaan määritykset ja suunnittelukonsepti jonka pohjalta sitä lähdetään toteuttamaan, ja tämän tuloksena syntyy tuotteelle rakenne, joka integraation kautta siirtyy tuotetietojärjestelmään nimiketasolla. Sen pohjalta päivitetään myös tuotetietoon liittyvät valintasäännöt.

Koska kyseessä on sähkömoottorit, tehdään suunnittelua myös elektroniikkatasolla. Nimikkeissä nämä tiedot löytyvät attribuutteina niin sanottujen sähköarvokorttien sekä käämitysnimikkeiden takaa. Tällä hetkellä on käytössä suomalainen Vertex, mutta tarkoituksena on siirtyä NX:än sen ollessa saman valmistajan tekemä kuin Teamcenter.

3.1.2 Viestiliikenne järjestelmässä

Taulukko 2 kokoaa kaikki väliohjelman yhdistämät järjestelmät, esittää näiden ohjelmien käyttämät viestimuodot sekä mitä dataa lähetetään niin lähteestä kohteeseen kuin kohteesta lähteeseen.

Taulukko 2: Ohjelmien yhteydet, viestien muodot ja sisältö

Lähde	**Kohde**	**Lähdemuoto**	**Kohdemuoto**	**Data** (lähde/kohde)
Aton	Konfiguraattori	Tietokanta	Tietokanta	Attribuutit/rakenne
Aton	Teamcenter	XML	PLMXML	Rakenne, nimikkeet
Aton	iLM	XML	Teksti	Testidata
Aton	Navision	XML	XML	Rakenne/pyyntö, koeajot
Navision	iLM	XML	Teksti	Tilausvahvistus, pakkauslista/tilaus
NX	Teamcenter			Rakenne
Vertex	Aton	VXG	XML	Rakenne
Vertex	Teamcenter	VXG	PLMXML	Rakenne

Viestiliikenteen välityksessä käytetään IBM:än WebSphere MessageQueue väliohjelmaa. Se vastaanottaa lähdeohjelman viestin, muuntaa sen kohdeohjelman ymmärtämään muotoon ja toimittaa sen eteenpäin kohteelle. Vertexin ja Atonin välissä on oma väliohjelma nimeltä Aton Broker. Se kuuntelee HTTP-pyyntöjä, ja palauttaa halutun tiedon HTTP-vastauksena, joka sisältää datan XML-muodossa.

Kun Navision lähettää pyynnön moottorirakenteelle, siirtyy alla oleva sisältö XML-tiedostona järjestelmässä:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<objectarray>
  <item>
    <item_code>52497146</item_code>
    <reference>D0099572/2</reference>
    <system_code>KONNA</system_code>
  </item>
</objectarray>
```

Tämän seurauksena käynnistyy konfiguraattori joka luo kyseiselle moottorille rakenteen. Vastaus lähetetään XML tiedostossa, josta ote kuvassa 22.

```xml
<item>
 <item_code>52622024</item_code>
 <item_ver>1</item_ver>
 <item_type>KCIPROD</item_type>
 <item_group>SA-ASY</item_group>
 <item_status>CHECKED_ADMIN</item_status>
 <item_deleted>N</item_deleted>
 <item_current>Y</item_current>
 <item_language>ENG</item_language>
 <item_magnitude>PCS</item_magnitude>
 <item_desc1>MF13Z-022</item_desc1>
 <item_desc2>PART LIST</item_desc2>
 <item_desc3>V16 Outsourced service</item_desc3>
 <item_desc4 />
 <item_desc5 />
 <item_info>korvaa luettelon MF13X-033</item_info>
 <item_mancode />
 <item_delivery_time />
 <item_oemcode />
 <item_handled>KERANVI</item_handled>
 <item_create>KERANVI</item_create>
 <item_create_date>2008-03-08 12:10:42</item_create_date>
 <item_modify>MST</item_modify>
 <item_modify_date>2010-02-18 14:44:11</item_modify_date>
</item>
```

Kuva 22: Ote moottorin rakenteesta XML muodossa

Kuva näyttää moottoriin liittyvän osaluettelon (engl. part list) MF13Z-022 tiedot. Samassa XML-tiedostossa lähetetään myös kuvan 23 mukainen merkkaus.

```xml
<item_structure>
 <parent_code>52622024</parent_code>
 <parent_version>1</parent_version>
 <item_code>53947122</item_code>
 <item_version>1</item_version>
 <str_partnro>10</str_partnro>
 <str_type />
 <str_deleted>N</str_deleted>
 <str_start_date>2008-03-25 00:00:00</str_start_date>
 <str_stop_date />
 <str_pcs>1</str_pcs>
 <str_qty />
 <str_length />
 <str_width />
</item_structure>
```

Kuva 23: Nimikkeen alirakenne XML muodossa

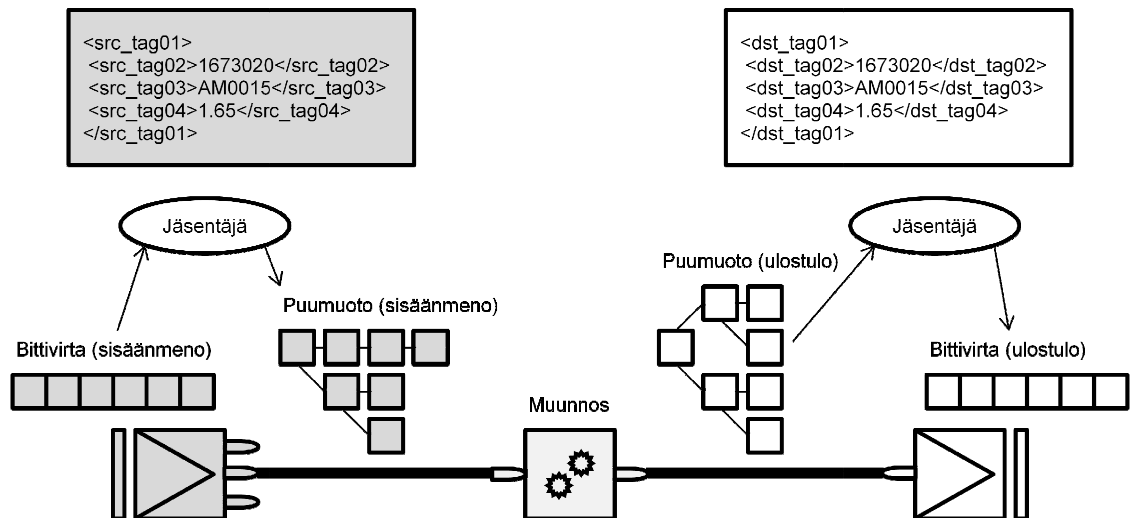
Tästä nähdään osaluettelon MF13Z-022 rivillä 10 olevan nimikkeen tiedot. XML-tiedosto sisältää kaikki moottorin päätason nimikkeiden alirakenteet vastaavalla tavalla. Moottoreihin liittyvä testidata siirretään seuraavankaltaisessa rakenteessa:

```
<serial_attribute>
 <indiv_serial>1673020</indiv_serial>
 <attribute_code>AM0015</attribute_code>
 <tech_value>1.65</tech_value>
</serial_attribute>
```

Näiden esimerkkien perusteella voidaan lähteä etsimään muunnossääntöjä Navisionin ja Teamcenterin keskinäiselle tiedonvälitykselle.

Siirrettävä tiedosto lähetetään ensiksi FTP-palvelimelle (engl. File Transfer Protocol), josta väliohjelma poimii ne säännöllisin väliajoin. Tiedoston muodosta riippuen väliohjelma vie tiedoston sille tarkoitetulle jäsentäjälle. Jäsentäjän tehtävä on muodostaa bittivirrasta oma, väliohjelman sisäinen hierarkinen puumuoto. Kun kyseessä on XML-tiedosto, on ohjelmaan määritelty sille sopiva muunnos esimerkiksi XSLT:llä. Kun muunnos on suoritettu, saadaan tuloksena ulostulon puumuoto jonka jäsentäjä kääntää vielä ulostulon bittivirraksi. Ulostulosta tämä tiedosto viedään kohdeohjelmalle ja viesti on siirretty. Tämä prosessi on käyty läpi kuvassa 24.

Kuvassa 24 on kaksi jäsentäjää, sisäänmenossa ja ulostulossa. Tässä tapauksessa molemmat ovat XML-jäsentäjiä, mutta sisäänmenossa voisi olla vaikka C-kielen tietorakenteita tulkkaava jäsentäjä, jolloin muunnos tapahtuisi kahden täysin erilaisen tiedostomuodon välillä. Toinen esimerkki voisi olla SAP:in iDoc-rakenteen ja XML-dokumentin välinen muunnos. Keskellä oleva muunnoskomponentti sisältää säännöt, jonka mukaan muunnos tehdään. Lähdetiedoston muoto esitetään kuvassa <src_tag0x>-tageilla, ja tuloksena syntyvä kohdemuoto on <dst_tag0x>-tageilla.



Kuva 24: MessageQueuen muunnosvuo [20]

Muunnosvuon graafi vastaa osaltaan integraation suunnittelumalleja, jossa malli esittää kysymyksen, ja vuo antaa vastauksen. Suunnittelumallit kuvaavat koko järjestelmää korkeammalta tasolta, kun taas muunnosvuomalli kuvaa tarkemmin yhtä kokonaista prosessia ja sitä, miten suunnittelumallin tietty tapahtuma voidaan toteuttaa tekniikan kannalta. MessageQueue on näin käytännön ratkaisu suunnittelumallin esittämälle ESB kokonaisuudelle.

3.2 Integraation ja migraation tarve

Integraation tarve kumpuaa yrityksen asettamista tavoitteista, joiden mukaan vanha PDM poistuu käytöstä. Päätös on täysin poliittinen eikä muita vaatimuksia kuin se, että moottorimme ovat uudessa järjestelmässä ajoissa ole annettu. Tämän työn tarkoituksena on saada projekti hyvään alkuun.

AtonPDM:än vaihtuessa Teamcenteriin on vanhasta järjestelmästä saatava siirrettyä moottoreiden sisältöihin liittyvät nimikkeet (kuten aiemmin mainitut sähköarvokortit ja käämitysnimikkeet) ja niiden attribuutit uuteen. Tämä vaatii esitutkimuksena vanhan ja uuden tietokannan rakenteiden selvittämisen, jotta tarvittavat muutokset voidaan toteuttaa ohjelmallisesti. Myös nimikkeisiin liitetyt piirustukset, kokoonpanokuvat ja erilaiset tekstimuotoiset dokumentit on otettava huomioon.

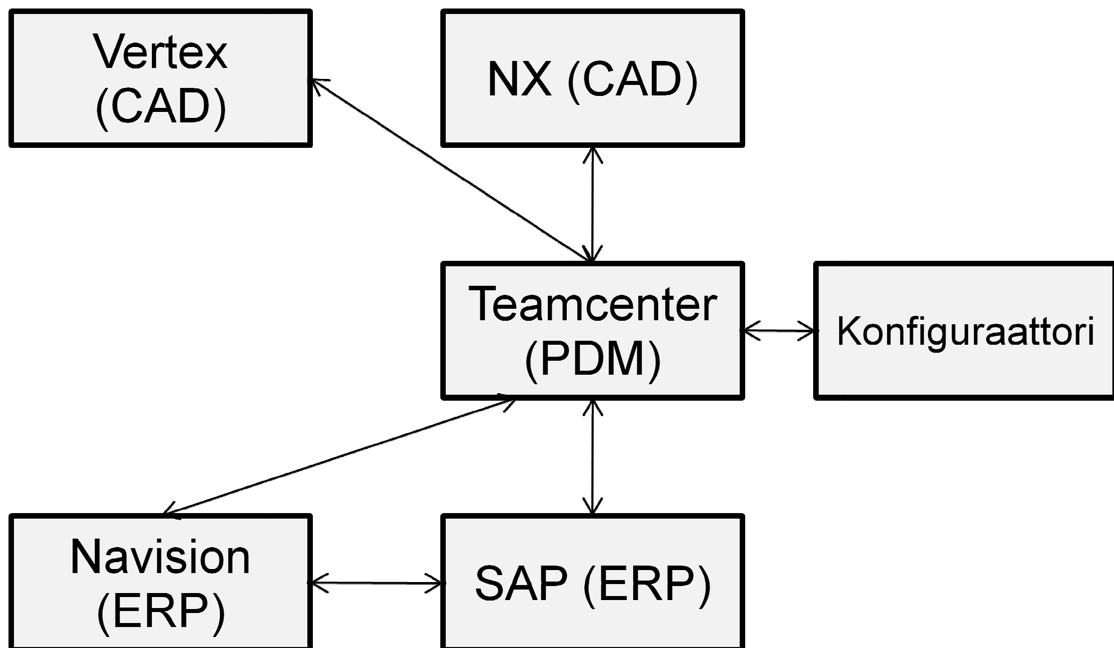
Valintasäännöt on rakennettu AtonPDM:ssä kuvan 19 mukaisesti, eli graafisen liittymän kautta piirtämällä. Näiden tekeminen on noin vuoden projekti kahdelta hengeltä, joten ajansäästön kannalta olisi mielekästä saada siirrettyä nämä tiedot suoraan tietokannasta tietokantaan. Tämä vaatii Teamcenterin tuotekonfigurointilogiikan selvittämisen, sekä sen miten tuotemallien rakenne saadaan toteutettua moottoreille ohjelmassa. Tämän jälkeen on mahdollista toteuttaa näiden tietojen muokkaus ja siirto raakadatan muodossa.

Alihankkijan kokoamista ja testaamista moottoreista tehdään raportit, jotka tallennetaan kyseisen moottorin sarjanumeron alle PDM:ään. Myös näiden tietojen siirtäminen on osa migraatioprojektia, ja tässä työssä on tarkoitus tutkia, miten testidata saadaan siirrettyä uuteen järjestelmään.

Datamigraatio toimii osittain esitietona viestien muuntamiselle ohjelmien välillä, sillä nimikkeiden rakenteet ja niihin linkitettyjen tiedostojen logiikka on oltava selvillä, jotta tiedostojen muoto Teamcenterin osalta saadaan esille.

3.3 Tavoitetila

Kuva 25 esittää projektin tavoitteena syntyvää järjestelmää. AtonPDM jää kokonaan pois ja vanha ERP on korvattu SAP:illa, jolloin syntyy taulukon 3 mukaiset uudet yhteydet. Nyt kaikki viestit toimitetaan MessageQueuen kautta, ja näiden uusien viestien välille on kehitettävä sopiva muunnos väliohjelmassa. Poikkeuksena Teamcenterin ja konfiguraattorin (sekä Teamcenterin ja NX:n) välinen yhteys, joka on toteutettu sovelluksessa sisäisesti. Konfiguroituvan nimikkeen saamat arvot attribuuteilleen otetaan suoraan esimerkiksi Excel taulukkoon kootusta datasta.



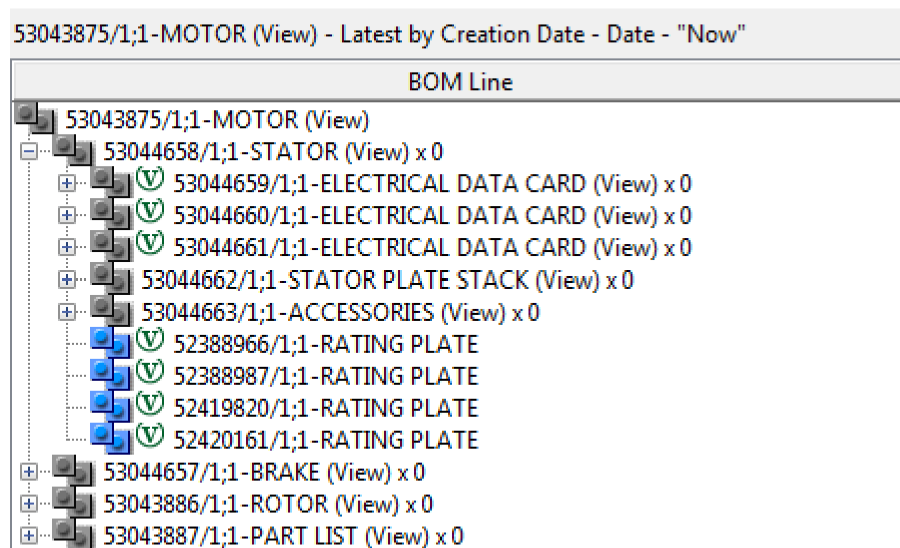
Kuva 25: Eri ohjelmien väliset yhteydet tulevaisuudessa

Taulukko 3: Uudet yhteydet

Lähde	**Kohde**	**Lähdemuoto**	**Kohdemuoto**	**Data** (lähde/kohde)
Teamcenter	Konfiguraattori	Tietokanta	Tietokanta	Attribuutit/ rakenne
Teamcenter	SAP	PLMXML	iDoc	Testidata
Teamcenter	Navision	PLMXML	XML	Rakenne/ pyyntö, testidata
SAP	Navision	iDoc	XML	Tilaus

4 Uusi järjestelmä

Työ aloitettiin määrittelemällä Teamcenteriin yhden moottorimallin rakenne, ja tälle tehtiin geneerinen tuoterakenne (engl. Generic Product Structure, GPS). GPS:än avulla saatiin mallille konfiguroituva nimike (kuva 26). Kuvassa 26 harmaa legopalikka tarkoittaa nimikettä jolla on attribuutteja, ja jotka voivat saada arvoja konfiguraatiossa. Ympyröity V merkkaa nimikkeen, jolle on annettu attribuuttiarvot ja jotka tulevat mukaan rakenteeseen vain, kun tämä attribuutti saa kyseisen arvon. Rakenteen ja konfigurointilogiikan perusteella voidaan selvittää missä muodossa konfiguroituvien nimikkeiden valintasäännöt siirretään Teamcenteriin, ja missä muodossa XML-data viedään ja tuodaan ohjelmaan.



Kuva 26: Moottorin rakenne Teamcenterissä

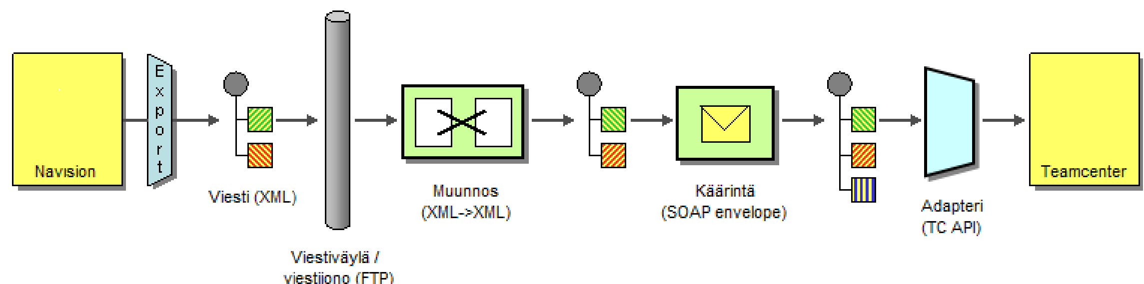
Tavoitteisiin on tarkoitus päästä selvittämällä Atonin tietokannassa olevien valintasääntöjen rakenne, jotta ne voidaan ottaa kannasta ulos ja muokata Teamcenterin säännöiksi sekä tämän jälkeen viedä ne suoraan Teamcenterin tietokantaan. Tuotemallien rakenne viedään myös suoraan uuteen tietokantaan pienin muokkauksin, koska rakenne tulee vastaamaan suurelta osin vanhaa rakennetta. Tiedonsiirto XML-muodossa eri ohjelmistojen välillä hoidetaan Python muunnoksilla siten, että väliohjelman kautta siirtyvä data on muodoltaan samanlaista kuin nykyään. Näin saadaan integrointi alihankkijan järjestelmään mahdollisimman saumattomaksi, eikä sen tarvitse tehdä muutoksia päivityksen takia. SAP integraatio vaatii ESQL (engl.

Embedded Structured Query Language) kielisen säännön suunnittelun, koska iDoc rakenteet ovat tietokantarakenteita. Tietorakenteiden muutokset toteutetaan siis luomalla oikea muunnossääntö sekä suunnittelemalla sopiva viestivuo MessageQueueen.

Tämän jälkeen työssä käydään vielä läpi tapoja, joilla järjestelmässä esiintyviä ongelmakohtia voitaisiin parantaa. Tämä on teoreettinen selvitys, jossa käytetään integroinnin suunnittelumalleja hyväksi.

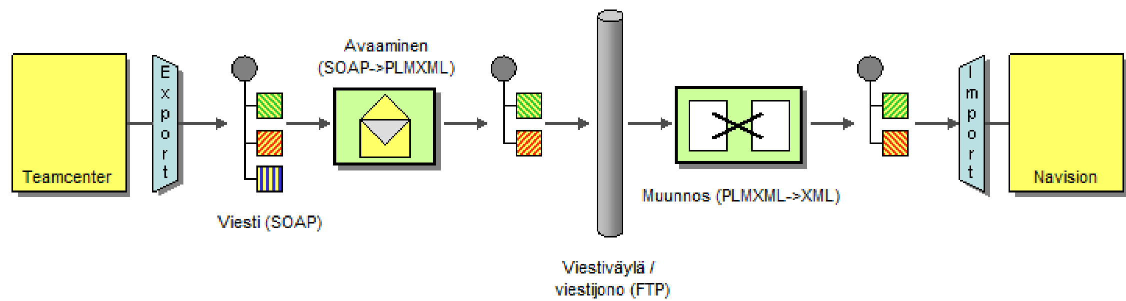
4.1 Teamcenter-Navision integraatio

Teamcenterin viestivuon perusteet ovat viestissä (PLMXML- ja XML-tiedostot) ja viestijonossa (FTP-palvelimella sijaitseva hakemisto). Kuvassa 27 on käyty läpi viestin kulku järjestelmässä kun Navisionista lähtee rakennepyyntö Teamcenterille. Navisionista tuodaan pyyntö XML-muodossa viestijonoon FTP-palvelimelle. Väliohjelma nappaa viestin jonosta ja suorittaa sille muunnoksen SOAP-muotoiseksi viestiksi (SOAP body), jonka jälkeen se kääritään vielä SOAP-kirjeeseen (envelope). SOAP-viestin avulla voidaan kutsua Teamcenterin API:a vastauksen saamiseksi.



Kuva 27: EIP kuvaus väliltä Navision-Teamcenter (pyyntö)

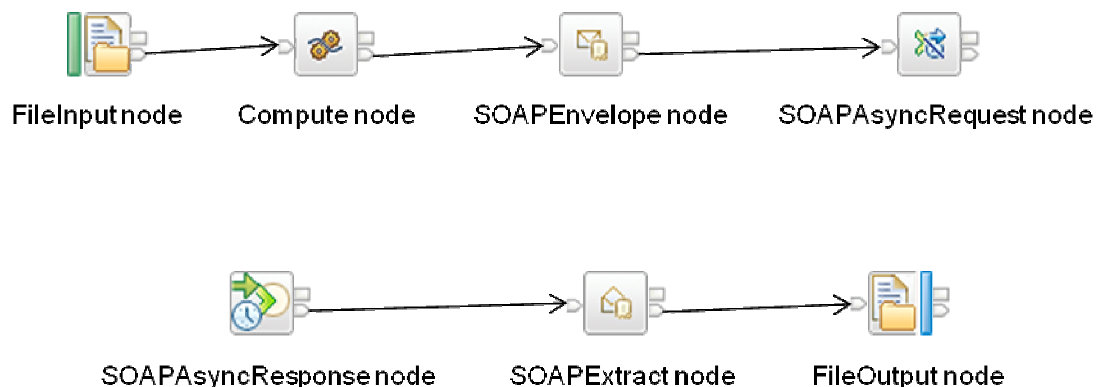
Kuva 28 käy läpi vastauksen toimittamisen. Teamcenter vastaa SOAP-viestillä pyyntöön, joka avataan (SOAP body otetaan talteen jolloin saadaan PLMXML-tiedosto). Viesti viedään jälleen jonoon palvelimelle, jossa tiedostolle suoritetaan muunnos Pythonilla (liite III) XML-muotoon. Tämän jälkeen se viedään alihankkijalle.



Kuva 28: EIP kuvaus väliltä Teamcenter-Navision (vastaus)

4.1.1 Teamcenter-Navision WebSphere-vuo

Edellä käydyt viestivirrat voidaan hajoittaa edelleen pienemmiksi kokonaisuuksiksi WebSphere-väliohjelman viestintälogiikkaa varten (kuva 29). Viestin reitti kuvataan solmuilla (engl. node), jotka jokainen ovat oma toiminnallinen kokonaisuutensa. Yksi solmu voi sisältää joko yksittäisen funktion tai enemmän logiikkaa.



Kuva 29: Viestin kulku WebSphere väliohjelmassa (Teamcenter-Navision)

FileInput: solmu konfiguroidaan lukemaan FTP-palvelimen määriteltyä hakemistoa tietyin väliajoin.

Compute: ekstraktoidaan XML-tiedostosta tarvittavat alkiot: <item>, <reference> ja <system_code>.

SOAPEnvelope: rakentaa SOAP-viestin annetuista alkioista

SOAPAsyncRequest: asynkroninen SOAP-solmu jolloin viesti voidaan lähettää ilman, että suoritusta keskeytetään vastauksen saamisen ajaksi. Koska Teamcenter tarjoaa

valmiit WSDL-kuvaukset palveluistaan, voidaan tämä solmu konfiguroida automaattisesti niitä käyttämään. Solmu lähettää SOAP-muotoisen pyynnön Teamcenterille.

SOAPAsyncResponse: solmu joka ottaa Teamcenterin vastauksen vastaan.

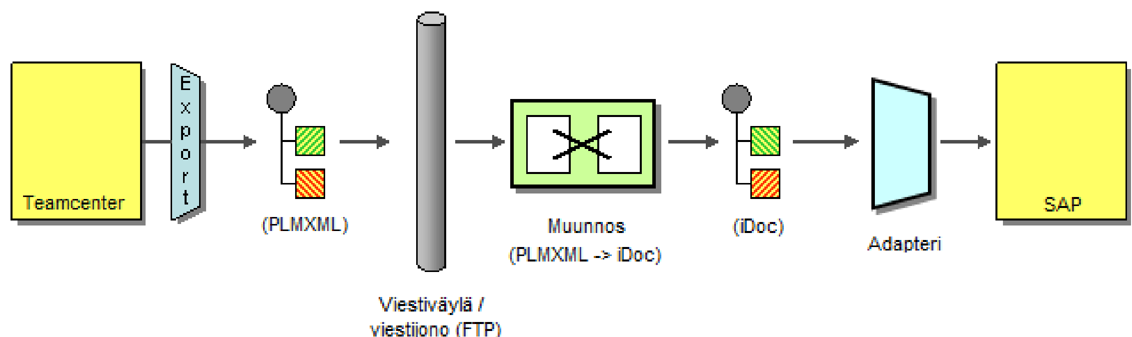
SOAPExtract: riisuu SOAP-muotoisen vastauksen ja antaa ulostulona PLMXML-viestin.

FileOutput: vie PLMXML-viestin FTP-palvelimen hakemistoon. Tämän jälkeen palvelimella suoritetaan Pythonin avulla muunnos PLMXML->XML, mutta se ei kuulu enää WebSpheren toimintaan.

4.2 Teamcenter-SAP integraatio

Uutena ERP–järjestelmänä kokonaisuuteen tuodaan SAP. Moottorien testidatan ensin siirryttyä Teamcenteriin, pitää ne viedä vielä SAP:iin jotta myyntiosasto saa tulostettua testit toimitettavien moottorien mukaan asiakkaalle.

Kuva 30 esittelee prosessin. Uutena komponenttina tulee iDoc-tyyppinen tiedostomuoto, joka täytyy rakentaa PLMXML-tiedoston pohjalta. Muutoin kuvaus ei sisällä mitään Teamcenter-Navision siirrosta poikkeavaa, joten toimintoja ei EIP:n osin käydä tarkemmin läpi.



Kuva 30: EIP kuvaus väliltä Teamcenter-SAP (koeajotiedot)

4.2.1 Teamcenter-SAP WebSphere-vuo

Kuvassa 31 esitetään viestin kulku väliohjelmassa kun siirto tehdään Teamcenterin ja SAP:in välillä. WebSphere sisältää valmiit solmut SAP:in integrointia varten.

Kuva 31: Viestin kulku WebSphere väliohjelmassa (Teamcenter-SAP)

FileInput: solmu konfiguroidaan lukemaan FTP-palvelimen määriteltyä hakemistoa tietyin väliajoin.

Compute: solmussa toteutetaan ESQL-kielellä muunnos. Tästä saadaan tuloksena kentät jotka tarvitaan test_data-objektin luomiseksi SAP:iin.

SAPRequest (create): käynnistää SAP:in BAPI:n, (engl. Business Application Programming Interface) jonka avulla objekti luodaan konkreettisesti. SAP palauttaa objektin ID:n.

Compute: SAP:in palauttaman ID:n avulla rakennetaan viesti, jonka avulla test_data-objektia voidaan päivittää.

SAPRequest (update): lähetetään edellä generoitu viesti SAP:iin jolloin objekti päivitetään sisältämään testiarvot kenttiinsä.

4.3 Datamigraatio

4.3.1 Tuotemallien valintasääntöjen siirtäminen Teamcenteriin

Atonin tietokannan pohjalta muodostettiin SQL-lause (liite I), jonka avulla saatiin valintasäännöt ulos tekstimuodossa. Haussa viimeinen IN('xxxx')-määre ottaa arvoikseen kannan sisäisiä ID-numeroita, jotka saatiin selville toisen SQL-lauseen avulla (liite I).

Haut oli tehtävä erikseen koska jälkimmäisen haun lomittaminen ensimmäisen haun IN määreeseen hidasti suoritusaikaa dramaattisesti. Hakujen tuloksena ulos tuli sääntö seuraavassa muodossa:

RATING & WINDING M16 SM 1 52388966 1 AND (MOT48 OP_NIN ZONE 2/22) AND (SPD15 OP_IN 106)

Tämän pohjalta tehtiin yksinkertainen muunnossääntö Python-kielellä (liite II), joka muodosti tietokannasta saadusta muodosta Teamcenterin ymmärtämän valintasäännön:

53044658:MOT48 != "ZONE 2/22" and 53044658:SPD15 = "106"

Kuvassa 26 tämä sääntö lisätään ensimmäisellä sinisellä legopalikalla olevalle nimikkeelle, jonka ID on 52388966. Säännössä oleva viittaus nimikkeeseen ID:llä 53044658 osoittaa varioituvaan STATOR-nimikkeeseen, jolta tämä konfiguroituva nimike ID:llä 52388966 ottaa arvonsa konfiguroituessa.

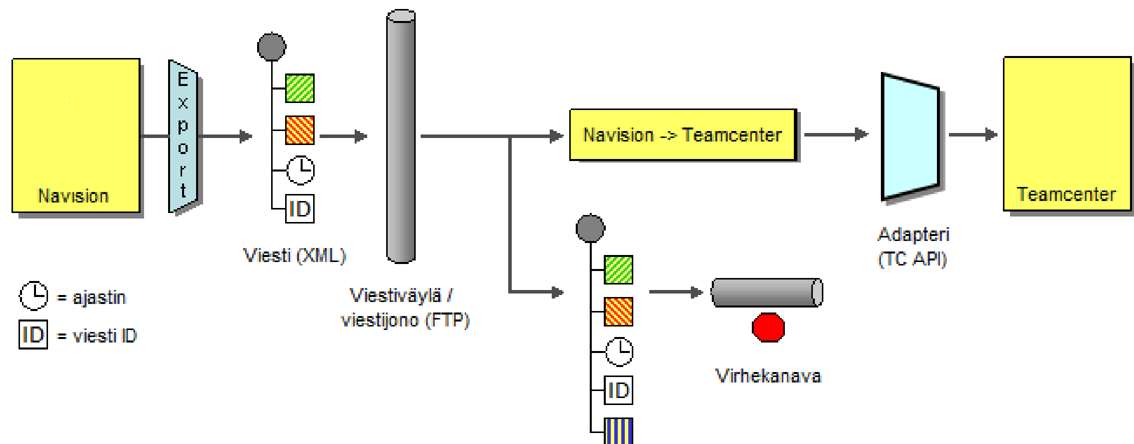
Kuvan 26 mukaisesti tuotemallin rakenne Teamcenterissä saatiin vastaamaan hyvin paljon sitä, millainen se on Atonissa. Näin eri mallien rakenteet saadaan vietyä suoraan ohjelmasta toiseen, ja valintasäännöt helposti liitettyä oikeille nimikkeille.

5 Parannuksia järjestelmään

5.1 Tapaus: pyyntöjen hukkuminen

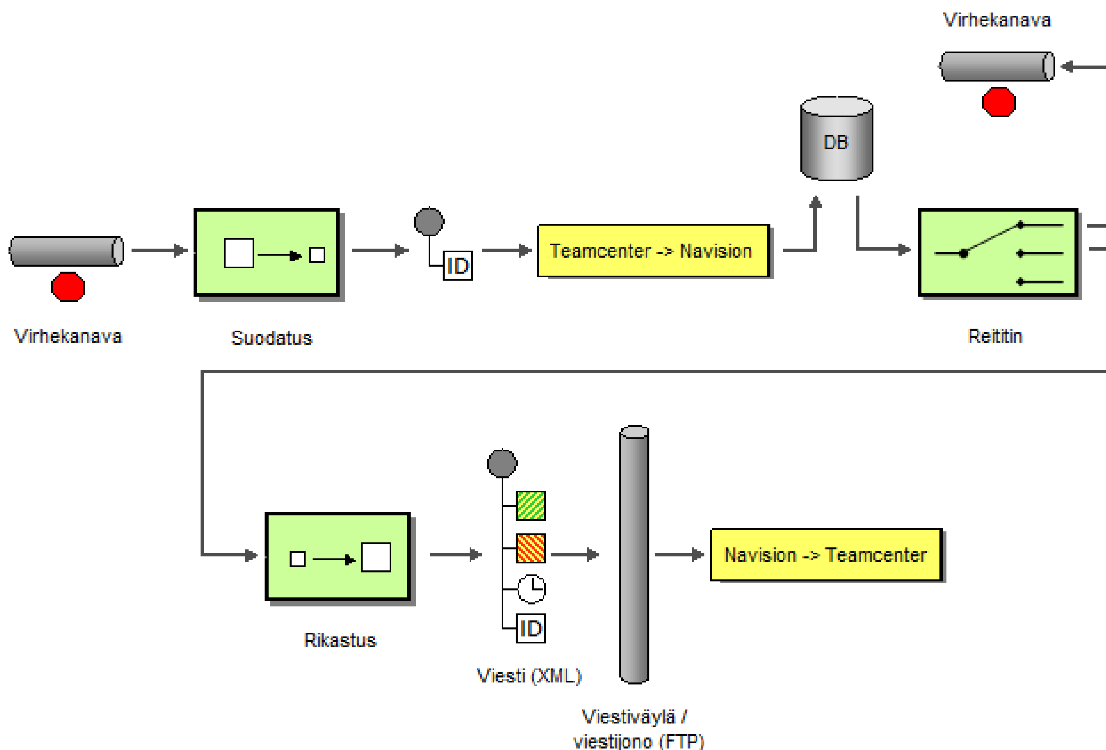
Yleinen ongelmatapaus alihankkijan lähettäessä rakenne- ja konfigurointipyyntöjä on niiden hukkuminen matkan varrella erilaisten tiedonsiirtoon liittyvien katkosten takia. Pyynnöt eivät saavu perille tai niille ei saada vastausta, eikä alihankkija pysy perässä siitä mitkä pyynnöistä ovat toteutuneet. Tähän voidaan tarjota ratkaisuksi niin sanottua *request ID*:tä joka lisätään jokaiseen rakennepyyntöön, ja joka tarkastetaan alihankkijan päässä saaduista vastauksista. Näin tiedetään mitkä pyynnöt ovat onnistuneet.

Kuva 32 esittää viestin kulun välillä Navision-Teamcenter. XML-tiedostoon generoidaan tunnistenumero (viesti ID) ja ajastin ennen kuin se viedään jonoon. Mikäli annettu aika on kulunut ennen vastausta, viedään viesti virhekanavaan. Muutoin se viedään Teamcenterille. Kuvassa 32 Navision->Teamcenter–laatikko sisältää kuvassa 27 suunnitellun logiikan.



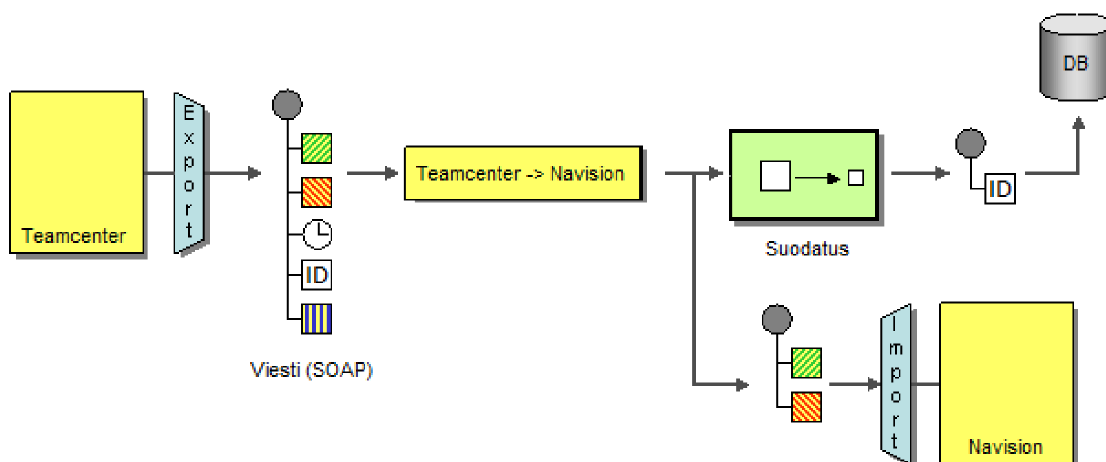
Kuva 32: Rakennepyyntö ja viesti ID

Kuvassa 33 käydään läpi miten viesti etenee virhekanavaa pitkin. Viestistä suodatetaan ID talteen, jota käydään vertaamassa tietokantaan. Tietokannassa pidetään yllä tietoa siitä, kuinka monta kertaa kyseistä nimikettä on pyydetty. Mikäli pyyntöjä on tehty kolme kertaa, ohjataan viesti jälleen virhekanavaan, jonka jälkeen se merkataan epäonnistuneeksi. Jos pyyntökertoja on vielä jäljellä, ohjataan viesti rikastimen läpi uudelleen Teamcenterille.



Kuva 33: Viestin reitti virhekanavaa pitkin

Mikäli kuvan 32 reiteistä ei päädytä virhekanavaan, palaa vastaus kuvan 34 mukaisesti alihankkijalle. Vastaukseen lisätään korrelaatio ID, joka vastaa järjestelmään pyynnön mukana tullutta viesti ID:tä. Vastauksesta suodatetaan ensin ID pois, jota vertaillaan tietokannassa oleviin tietoihin. Näin saadaan pyyntö oikein kuitatuksi. Itse moottorirakenne otetaan viestistä ja viedään Navisioniin, jonka jälkeen prosessi on valmis.

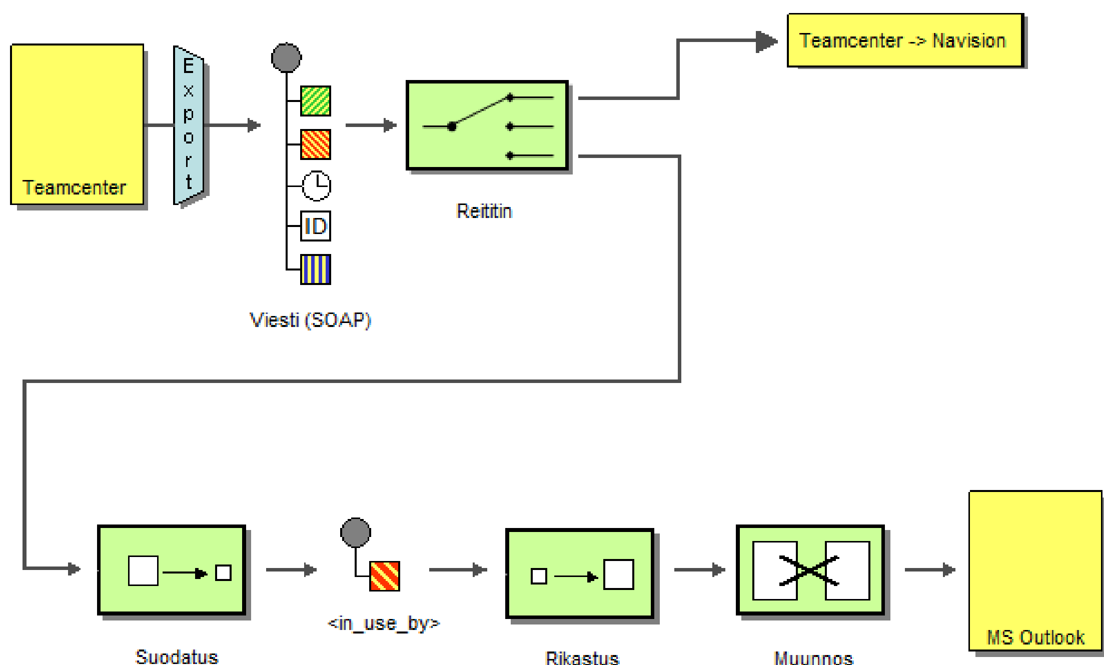


Kuva 34: Vastauksen reitti

5.2 Tapaus: nimike varattuna

Pyyntöjen hukkumista yleisempi, mutta ei niin vakava ongelma on nimikkeiden unohtaminen varattuun tilaan. Kun suunnittelija tekee muutoksia nimikkeeseen, täytyy se ensin varata kyseiselle käyttäjälle. Tänä aikana muut käyttäjät eivät voi tehdä nimikkeelle kuin *SELECT* hakuja tietokantaan. Tämä johtaa myös siihen, ettei tehdyt pyynnöt onnistu, vaan pyynnön lähettäjän täytyy pyytää nimikkeen varannutta vapauttamaan se. Tästä taas voi seurata pitkäkin katkos toimitusputkeen. Ratkaisuksi suunnitellaan toteutus, jossa varattua nimikettä pyydettäessä ilmoitetaan oikealle henkilölle asiasta sähköpostitse.

Kuvassa 35 esitetään miten vastaus lähtee Teamcenteristä, ja miten otetaan nimikkeen varaus huomioon. Reititin valitsee määränpään viestin sisällön perusteella. Tässä voidaan tutkia XML-tiedoston <in_use_by>-elementtiä. Mikäli elementti on tyhjä, lähetetään vastaus normaalia reittiä pitkin alihankkijalle. Mikäli nimike on käytössä, suodatetaan siitä nimikkeen varanneen henkilön käyttäjänimi, ja rikastetaan viesti vielä itse sähköpostiviestin sisällöllä. Tämän jälkeen muunnetaan viesti C#-kieliseksi ohjelmaksi, jonka avulla voidaan kutsua Outlook-sähköpostisovellusta lähettämään haluttu viesti.



Kuva 35: Sähköpostin käyttö kun nimike on varattuna

6 Arviointi

Tässä luvussa käydään läpi työn päävaiheet, miten niissä onnistuttiin ja mitä on vielä tehtävä.

6.1 Tuotemalli ja valintasäännöt

Työ aloitettiin määrittelemällä tuotemallille rakenne Teamcenteriin. Tämä saatiin tehtyä vanhan Aton-mallin pohjalta. Atonin tietokannasta tuotiin SQL-haulla konfiguraattorin valintasäännöt tekstimuodossa, joista tallennettiin varmuuskopio tulevaisuutta varten. Näitä sääntöjä tutkimalla tehtiin ensin C++-kielellä muuntaja, joka muokkasi säännöt Teamcenterin ymmärtämään muotoon. Tämä ratkaisu kuitenkin hylättiin, koska tutustuin työssä myös Python-kieleen, ja totesin sen olevan huomattavasti parempi tämän tyyppiselle tekstimuunnokselle. Tuoterakenteen määrittelyn jälkeen tehtiin Teamcenteriin kokonainen tuoteperhe moottorimallista SM16, ja tästä tehtiin konfiguroituva rakenne. Tähän rakenteeseen liitettiin Pythonilla muokatut säännöt, ja todettiin tuloksen konfiguroituvan oikein.

Tulevaisuudessa Atonista viedään tuotemallit Teamcenteriin, jolloin saadaan kaikille malleille geneerinen rakenne. Näihin rakenteisiin ajetaan vielä valintasäännöt, jolloin koko moottorikatalogi saadaan konfiguroitua Teamcenterissä.

6.2 Integraatio ja muunnossääntö

Sovellusten väliselle viestiliikenteelle saatiin määriteltyä vuot teoreettisella tasolla, niin integraation suunnittelumalleilla kuin WebSphere MessageQueuen solmuilla. Suunnittelumalleilla kuvattiin myös mahdollisia ratkaisuja järjestelmässä esiintyviin ongelmiin, joita ovat konfigurointipyyntöjen häviäminen tietokatkoksen vuoksi, sekä nimikkeen jääminen varattuun tilaan. Nämä toteutettiin myös vain teoreettisella tasolla.

PLMXML-XML-muunnosta aloitettiin toteuttamaan ensin XSLT:n avulla. Työn edetessä saatiin kuitenkin tieto, ettei Teamcenterin tapauksessa enää käytetäkään XSLT:tä PLMXML-tiedoston rakenteen ollessa liian raskas XSLT-muunnokselle. Tekniikaksi oli valittu palvelimella toimiva Python-ohjelma, joten toteutuksessa siirryttiin käyttämään sitä. Rakennemuunnos saatiin toimimaan testeissä oikein.

Seuraavaksi on tehtävä muunnos alihankkijan XML-pyynnöstä Teamcenterin SOAP-muotoon, jolloin suunniteltu Navision-Teamcenter-vuo antaisi toivottavasti oikean rakenteen, ja muokkaisi siitä oikeanlaisen XML:n. Myös itse vuot täytyy testata tuotantoympäristössä. Tällä hetkellä Teamcenter ei tuota täydellisiä moottorirakenteita PLMXML-muodossa, joten tämä rutiini täytyy saada valmiiksi. Myös erilaiset liitetiedostot, kuten suunnittelupiirrustukset on vielä saatava tuoduksi Teamcenteristä, jonka jälkeen sääntö voidaan päivittää ottamaan ne huomioon.

LÄHTEET

[1] Konecranes vuosikertomus 2010 [WWW-dokumentti] [Viitattu: 01.11.2011]. Saatavilla: <http://www.konecranes.com/files/attachments/ir/annual_reports/kc_2010_fi_v03.pdf>

[2] David S. Linthicum. 2003. Enterprise Application Integration. 5. painos. Addison-Wesley Professional. 400 sivua.

[3] Rahul Sharma, Beth Stearns, Tony Ng. 2003. J2EE Connector Architecture and Enterprise Application Integration. 2. painos. Prentice Hall. 416 sivua.

[4] John Morris. 2009. Practical Data Migration. 2. painos. British Informatics Society Ltd. 228 sivua.

[5] Qusay H. Mahmoud. 2004. Middleware for Communications. 1. painos. Wiley Publishing, Inc. 522 sivua.

[6] Erik T. Ray. 2003. Learning XML. 2. painos. O'Reilly Media. 432 sivua.

[7] Siemens PLM Software. 2011. PLMXML White Paper. [WWW-dokumentti] [Viitattu: 07.11.2011]. Saatavilla: <http://www.plm.automation.siemens.com/en_us/products/open/plmxml/sdk.shtml#lightview%26uri=tcm:1023-11521%26title=Open%20Product%20 Lifecycle%20Data%20Sharing%20Using%20XM%20-%20PLM%20Components%20 White%20Paper%20-%201247%26docType=.pdf>

[8] Kogent Learning Solutions, Inc. 2011. SAP ABAP Handbook. 2. painos. Jones & Bartlett Publishers, Inc. 950 sivua.

[9] David S. Linthicum. 2003. Next Generation Application Integration: From Simple Information to Web Services. 2. painos. Addison-Wesley Professional. 512 sivua.

[10] Jeff Davies, David Schorow, Samrat Ray, David Rieber. 2008. The Definitive Guide to SOA: Oracle Service Bus. 2. painos. Apress. 550 sivua.

[11] Mulesoft. 2011. Understanding Enterprise Application Integration - The Benefits of ESB for EAI. [WWW-dokumentti] [Viitattu: 07.11.2011]. Saatavilla: <http://www.mulesoft.org/enterprise-application-integration-eai-and-esb>

[12] Best Price Computers. Enterprise Application Integration. [WWW-dokumentti] [Viitattu: 07.11.2011]. Saatavilla: <http://www.bestpricecomputers.co.uk/glossary/enterprise-application-integration.htm>

[13] Michael Rosen, Boris Lublinsky, Kevin T. Smith. 2008. Applied SOA: Service-Oriented Architecture and Design Strategies. 1. painos. Wiley Publishing, Inc. 696 sivua.

[14] Fabio Casati, Dimitrios Georgakopoulos, Ming-Chien Shan. 2001. Technologies for E-services, Volume 2. 1. painos. Springer. 219 sivua.

[15] Sven Casteleyn, Florian Daniel, Peter Dolog, Maristella Matera. 2009. Engineering Web Applications. 1. painos. Springer. 362 sivua.

[16] Frederick Hirsch, John Kemp, Jani Ilkka. 2006. Mobile Web Services: Architecture and Implementation. 1. painos. Wiley Publishing, Inc. 338 sivua.

[17] James Snell, Doug Tidwell, Pavel Kulchenko. 2002. Programming Web Services with SOAP. 1. painos. O'Reilly Media. 264 sivua.

[18] Gregor Hohpe, Bobby Woolf. 2003. Enterprise Integration Patterns. [WWW-dokumentti] [Viitattu: 30.01.2012]. Saatavilla: <http://www.eaipatterns.com/eaipatterns.html>

[19] Alexis Leon. 2008. Enterprise Resource Planning. 2. painos. Tata McGraw-Hill Education Private, Ltd. 388 sivua.

[20] Steve M Hanson, Sanjay Nagchowdhury. 2008. Message Modeling and Parsing Enhancements in WebSphere Message Broker V6.1. [WWW-dokumentti] [Viitattu: 12.02.2012]. Saatavilla: <http://www.ibm.com/developerworks/websphere/library/techarticles/0810_hanson/0810_hanson.html>

LIITE I: SQL haku valintasäännöille

SELECT variant.item_code "variant_code", variant.item_ver, tgtitem.item_code "target item", tgtitem.item_ver, sys_connect_by_path('('||header_code||' '||rule.operator_id||' '||val.value||')',' AND ')"rulepath"
FROM mst.mst_item_main variant, mst.mst_variant_node node, mst.mst_rule_str rule, mst.mst_variant_value val, mst.mst_variant_action action, mst.mst_item_str tgt, mst.mst_item_main tgtitem
WHERE variant.item_id=node.item_id **AND** rule.node_parent_id=node.node_id **AND** val.rule_id=rule.rule_id **AND** rule.node_id=action.node_id **(+) AND** action.action_operation='OP_SELECT' **AND** action.action_param1=tgt.str_id **(+) AND** tgt.str_deleted='N' **AND** tgt.item_id=tgtitem.item_id **(+) AND** tgtitem.item_deleted= 'N'
CONNECT BY PRIOR rule.node_id=rule.node_parent_id **START WITH** rule.node_parent_id **IN**('xxxx')

SELECT node.node_id
FROM mst_rule_str rule, mst_variant_node node
WHERE rule.node_parent_id=node.node_id
AND node.item_id=(**SELECT** item_id **FROM** mst_item_main **WHERE** item_code='RATING & WINDING M16 SM')
MINUS
SELECT node.node_id
FROM mst_rule_str rule, mst_variant_node node
WHERE rule.node_id=node.node_id
AND node.item_id=(**SELECT** item_id **FROM** mst_item_main **WHERE** item_code='RATING & WINDING M16 SM')

LIITE II: Python lähdekoodi valintasääntöjen muuntajalle

```
import string
import re

file=open('rules.txt')
tcrules=[]
rawobjects=[]

class rule(object):
  def __init__(self):
    object.__init__(self)
    self.target=''
    self.variant=''
    self.rule=[]
    self.parts=[]
    self.INs=[]
    self.EQs=[]
  def chopRules(self):
    for i in self.rule:
      regex=re.compile("\([ a-zA-Z0-9_/-]*\)")
      self.parts.append(regex.findall(i))

def parseRules():
  rawrules=file.readlines()
  for rawrule in rawrules:
    parsed_rawrule=rawrule.split('\t')
    if (len(rawobjects)==0 or len(rawobjects[len(rawobjects)-1].rule)==0) or
        parsed_rawrule[4!]!=rawobjects[len(rawobjects)-1].target:
      rawobj=rule()
      rawobj.variant=parsed_rawrule[0]
      rawobj.target=parsed_rawrule[4!]
      rawobj.rule.append(parsed_rawrule[3!])
      rawobjects.append(rawobj)
    else:
      rawobjects[len(rawobjects)-1].rule.append(parsed_rawrule[3!])

if __name__ == "__main__":
  parseRules()

  for object in rawobjects:
    object.chopRules()

  for object in rawobjects:
    for part in object.parts:
      for piece in part
        IN=piece.find('IN')
        NIN=piece.find('NIN')
        if IN!=-1:
          IN_attribute=piece[1:IN+2]
          for p in object.parts:
            for k in p:
              if k.find(IN_attribute)!=-1:
                if [IN_attribute, k[len(IN_attribute)+2:-1]] not in
                   object.INs:
                   object.INs.append([IN_attribute, k[len(IN_attribute)+2:-1]])
        else:
          if piece[1:-1] not in object.EQs:
            object.EQs.append(piece[1:-1])


   for object in rawobjects:
       old_inn=''
       rule_part01=''
       rule_part02=''
```

```
  for eq in object.EQs:
    eq=eq.replace(' OP_EQ ', '="')
    eq=eq + '"'
    rule_part01+=object.variant + ':' + eq + ' and '
  for inn in object.INs:
    inn[0]=inn[0].replace(' OP_IN', '="')
    inn[0]=inn[0].replace(' OP_NIN', '!="')
    if old_inn=='':
      rule_part02+='('
    if inn[0]!=old_inn and old_inn!='':
      rule_part02=rule_part02[:-4] + ') and ('
    rule_part02+=object.variant + ':' + inn[0] + inn[2!] + '" or '
    old_inn=inn[0]
  rule_part01=rule_part01[:-5]
  if len(rule_part02)!=0:
    rule_part02=rule_part02[:-4] + ')'
    if len(rule_part01)!=0:
      rule_part02=' and ' + rule_part02
  print object.target + ' ' + rule_part01 + rule_part02
file.close()
```

LIITE III: Python lähdekoodi rakennemuunnokselle

```
import xml.dom.minidom as minidom
import string

doc=minidom.parse('plmxml.xml')
item_list=[]

class item(object):
  def __init__(self):
    object.__init__(self)
    self.plmxml_id=''
    self.plmxml_parentid=''
    self.item_root='<item>'
    self.item_code=''
    self.item_ver=''
    self.item_type=''
    self.item_group=''
    self.item_status=''
    self.item_deleted=''
    self.item_current=''
    self.item_language=''
    self.item_magnitude=''
    self.item_desc1=''
    self.item_desc2=''
    self.item_desc3=''
    self.item_desc4=''
    self.item_desc5=''
    self.item_info=''
    self.item_delivery_time=''
    self.item_handled=''
    self.item_create=''
    self.item_create_date=''
    self.item_modify=''
    self.item_modify_date=''
    self.item_root_end='</item>'

def getformRef(instancedID):
  ProductRevision=doc.getElementsByTagName('ProductRevision')

  for rev in ProductRevision:
    revisionID=rev.getAttributeNode('id')
    if instancedID=='#' + revisionID.nodeValue:
      children=rev.childNodes
      for child in children:
        if child.nodeType!=child.TEXT_NODE:
          formRef=child.getAttributeNode('formRef')
          if formRef:
            return formRef.nodeValue
            break

def getInstance(id, folk):
  occurrences=doc.getElementsByTagName('Occurrence')
  forms=doc.getElementsByTagName('Form')

  for node in occurrences:
    occurrenceID=node.getAttributeNode('id')
    if occurrenceID.nodeValue==id:
      instancedRef=node.getAttributeNode('instancedRef')
      if instancedRef:
        for form in forms:
          formID=form.getAttributeNode('id')
          if "#" + formID.nodeValue==getformRef(instancedRef.nodeValue):
            nimike=item()
            KCID=form.getAttributeNode('name').nodeValue
```

```
        nimike.item_code=KCID[:8]
        nimike.item_ver=KCID[9:10]
        nimike.plmxml_id=id
        nimike.plmxml_parentid=folk
        children=form.childNodes
        for child in children:
         if child.nodeType!=child.TEXT_NODE:
          uservalues=child.childNodes
           for values in uservalues:
            if values.nodeType!=values.TEXT_NODE:
             if values.getAttributeNode('title').nodeValue=='cs3_group':
              nimike.item_group=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_language':
              nimike.item_language=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_magnitude':
              nimike.item_magnitude=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_specification':
              nimike.item_desc1=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_description':
              nimike.item_desc2=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_maingroup':
              nimike.item_desc3=values.getAttributeNode('value').nodeValue
             if values.getAttributeNode('title').nodeValue=='cs3_info':
              nimike.item_info=values.getAttributeNode('value').nodeValue
        item_list.append(nimike)

def getStructure(ID):
  occurrences=doc.getElementsByTagName('Occurrence')

  for node in occurrences:
    occurrenceRefs=node.getAttributeNode('occurrenceRefs')
    id=node.getAttributeNode('id')
    instancedRef=node.getAttributeNode('instancedRef')

    if occurrenceRefs and id.nodeValue==ID:
      parent=node.getAttributeNode('parentRef')
      if parent and len(parent.nodeValue)==0:
        parent='NULL'
      elif parent:
        parent=parent.nodeValue
      getInstance(ID, parent)
      sub_ids=occurrenceRefs.nodeValue.split()
      for sub_id in sub_ids:
        getStructure(sub_id)
    elif id.nodeValue==ID:
      parent=node.getAttributeNode('parentRef')
      getInstance(ID, parent.nodeValue)

def getRoot(xml):
  root=doc.documentElement
  motor=doc.getElementsByTagName('ProductView')

  for node in motor:
    primaryOccurrenceRef=node.getAttributeNode('primaryOccurrenceRef')
    motor_id=primaryOccurrenceRef.nodeValue

  return motor_id

if __name__ == "__main__":
  topLevel=getRoot('plmxml.xml')
  firstLevel=getStructure(topLevel)
  getStructure(firstLevel)
  for item in item_list:
    print item.item_root
    print '<item_code>' + item.item_code + '</item_code>'
    print '<item_ver>' + item.item_ver + '</item_ver>'
    print '<item_type>' + item.item_type + '</item_type>'
    print '<item_group>' + item.item_group + '</item_group>'
```

```
print '<item_status>' + item.item_status + '</item_status>'
print '<item_deleted>' + item.item_deleted + '</item_deleted>'
print '<item_current>' + item.item_current + '</item_current>'
print '<item_language>' + item.item_language + '</item_language>'
print '<item_magnitude>' + item.item_magnitude + '</item_magnitude>'
print '<item_desc1>' + item.item_desc1 + '</item_desc1>'
print '<item_desc2>' + item.item_desc2 + '</item_desc2>'
print '<item_desc3>' + item.item_desc3 + '</item_desc3>'
print '<item_desc4>' + item.item_desc4 + '</item_desc4>'
print '<item_desc5>' + item.item_desc5 + '</item_desc5>'
print '<item_info>' + item.item_info + '</item_info>'
print '<item_delivery_time>'+item.item_delivery_time+
       '</item_delivery_time>'
print '<item_handled>' + item.item_handled + '</item_handled>'
print '<item_create>' + item.item_create + '</item_create>'
print '<item_create_date>' + item.item_create_date + '</item_create_date>'
print '<item_modify>' + item.item_modify + '</item_modify>'
print '<item_modify_date>' + item.item_modify_date + '</item_modify_date>'
print item.item_root_end
print ''

if item.plmxml_parentid:
  for parent in item_list:
    if '#' + parent.plmxml_id==item.plmxml_parentid:
      print '<item_structure>'
      print '<parent_code>' + parent.item_code + '</parent_code>'
      print '<parent_version>' + parent.item_ver + '</parent_version>'
      print '<item_code>' + item.item_code + '</item_code>'
      print '<item_version>' + item.item_ver + '</item_version>'
      print '<str_partnro>' + '</str_partnro>'
      print '<str_type>' + '</str_type>'
      print '<str_deleted>' + '</str_deleted>'
      print '<str_start_date>' + '</str_start_date>'
      print '<str_stop_date>' + '</str_stop_date>'
      print '<str_pcs>' + '</str_pcs>'
      print '<str_qty>' + '</str_qty>'
      print '<str_length>' + '</str_length>'
      print '<str_width>' + '</str_width>'
      print '</item_structure>'
      print ''
```