Antti Hyvönen

SÄHKÖKONEEN OSALUETTELON VALIDOINTI-TYÖKALUN KEHITTÄMINEN TIETÄMYSLÄHTÖISEN SUUNNITTELUN AVULLA

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2011

TIIVISTELMÄ

Hyvönen, Antti Erkki
Sähkökoneen osaluettelon validointityökalun kehittäminen tietämyslähtöisen suunnittelun avulla
Jyväskylä: Jyväskylän yliopisto, 2011, 84 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaajat: Sakkinen, Markku, Kivioja, Juha

ABB Oy Moottorit ja Generaattorit -liiketoimintayksikkö valmistaa sähkömoottoreita ja generaattoreita vaihteleviin asiakastarpeisiin. Koneet suunnitellaan osista, jotka muodostetaan puurakenteeksi tuotetiedon hallintajärjestelmään (PDM – Product Data Management). Puurakennetta kutsutaan yleisesti osaluetteloksi. Osaluettelon suunnittelu on monimutkainen prosessi ja sen sisältämä tietämys on jakaantunut moniin tietolähteisiin. ABB Oy:n tarpeena on kehittää järjestelmä, joka kokoaa hajanaisen tietämyksen ja validoi suunniteltavan osaluettelon tämän tietämyksen avulla.

Tässä tutkimuksessa on perehdytty tietämyslähtöiseen suunnitteluun ja sen menetelmiin. Tietämyslähtöinen suunnittelu tähtää tuotesuunnittelun tietämyksen tehokkaampaan käyttöön. Perehtymisen avulla on selvitetty, miten osaluettelon validointityökalu voidaan toteuttaa tietämyslähtöistä suunnittelua hyväksikäyttäen.

Tutkimuksen keskeinen tulos on validointityökalun prototyyppi sekä kirjallisuudesta löydetty MOKA-menetelmä (Methodology and software tools Oriented to Knowledge based engineering Applications). Tutkimuksessa selviää, kuinka menetelmää voidaan käyttää apuna ABB:n tarpeeseen suunniteltavan validointityökalua kehityksessä. Menetelmän avulla osaluettelon sisältämää tietämystä voidaan koota yhtenäiseksi kokonaisuudeksi ja formalisoida prototyypin tarkistuskriteereiksi eli validaatioiksi. Prototyypin kehitys etenee rinnan MOKA-menetelmän kanssa, jotta prototyypin käyttö tukisi tietämyslähtöistä suunnittelua. Menetelmän käyttö vaatii kuitenkin sen sovittamista ABB:n käyttötarkoitukseen. Lopuksi tutkimuksessa annetaan kehitysehdotuksia liittyen menetelmän käyttöön sekä prototyypin jatkokehitykseen.

Asiasanat: Osaluettelo, tuotetiedon hallintajärjestelmä, Teamcenter, tietämyslähtöinen suunnittelu, tietämyslähtöiset järjestelmät.

ABSTRACT

Hyvönen, Antti Erkki
Developing a BOM validation tool for electrical machines using Knowledge Based Engineering
Jyväskylä: University of Jyväskylä, 2011, 84 p.
Information Systems, Master's Thesis
Supervisors: Sakkinen, Markku, Kivioja, Juha

ABB Business unit Motors and Generators manufactures electrical motors and generators for various customer needs. Machines comprise of parts which form a tree into a Product Data Management (PDM) system. This tree is generally called a Bill-Of-Materials (BOM). Designing a BOM is a complex process and knowledge pertaining to the BOM is dispersed into several information sources. ABB has a need for a system which can compile information from different sources and validate the BOM using this information.

This thesis familiarized with Knowledge Based Engineering (KBE) and methods for using it. KBE is oriented toward more efficient use of engineering knowledge. In accordance with this orientation, this study seeks to define how KBE can be used in developing a BOM validation tool.

The essential result of this thesis is a validation tool prototype and a methodology found from literature called MOKA (Methodology and software tools Oriented to Knowledge based engineering Applications). This thesis shows how the methodology can be used for developing the validation tool for ABB's purposes. Through this method, knowledge related to the BOM can be collected into a coherent whole and formalized into validation criterions. The prototype development proceeds according to the MOKA method so that use of this prototype would support the methodology. However, use of the method requires it to be adapted for ABB's purposes. In the conclusions, this thesis gives suggestions for further developing the method and the prototype.

Keywords: Teamcenter, BOM (Bill-Of-Material), PDM (Product Data Management), KBE (Knowledge Based Engineering), KBS (Knowledge Based Systems).

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Nykyajan teollisuusyritykset tavoittelevat yleisesti parempaa suorituskykyä menettämättä kuitenkaan laatua tuotteistaan. Tällaisten tavoitteiden saavuttamisessa jatkuvasti kehittyvä tietotekniikka näyttelee suurta roolia. Nykyään puhutaan tuotteiden koko elinkaaren kattavista ohjelmista, joiden avulla yritykset voivat tehostaa suorituskykyään.

Tietojärjestelmien kehitys tuotesuunnittelun osalta kulkee automaation suuntaan. Yritykset pyrkivät määrittelemään tuotesuunnittelunsa siten, että se mahdollistaa suunnittelun uudelleenkäytön parhaalla mahdollisella tavalla. Kun mahdollistetaan suunnittelukomponenttien uudelleenkäyttö, niin mahdollistetaan samalla helpompi automaation lisäys suunnitteluprosessin yhteyteen.

Muun muassa tuotesuunnittelun automaatiota sekä uudelleenkäyttöä on tutkittu kirjallisuudessa tietämyslähtöisen suunnittelun (KBE – Knowledge-Based Engineering) lähestymistavan kautta. Lähestymistapaa pidetään tietämyslähtöisten järjestelmien (KBS – Knowledge-Based Systems) osa-alueena. Tietämyslähtöisessä suunnittelussa komponenttien uudelleenkäytöllä on suuri merkitys. Kun yritys on omaksunut uudelleenkäyttöisen lähestymistavan, niin yrityksen on mahdollista luoda tuotevariantteja tehokkaammin muun muassa lyhemmillä toteutusiteraatioilla. Monissa yrityksissä on jo käytössä tähän tarkoitukseen luotuja tietämyslähtöisiä järjestelmiä.

Tässä tutkimuksessa on tarkoituksena selvittää, kuinka teollisuusyritykselle voidaan toteuttaa sen omaa tuotesuunnittelua tukevia järjestelmiä. Tutkimuksen lähestymistapana käytetään tietämyslähtöistä suunnittelua. Lähestymistavan avulla pyritään suunnittelemaan ja toteuttamaan prototyyppi validointityökalusta. Lähestymistavan tarkoituksena on tuoda esille yleisiä ongelmakohtia ja ratkaisuja validointityökalun kaltaisen järjestelmän suunnitteluun ja toteutukseen.

1.1 Ongelman asettelu

Tietämyslähtöinen suunnittelu on ollut tutkijoiden mielenkiintona jo vuosikymmenien ajan. Tutkijoiden tarkoituksena on ollut toteuttaa järjestelmiä, joiden avulla suunnittelutoimia on voitu siirtää tietokoneiden tehtäviksi.

Tietämyslähtöisen suunnittelun käyttö tietyn yrityksen vaatimuksiin on tapauskohtaista. Yritysten tarpeet vaihtelevat hyvin paljon, ja siksi on oleellista löytää oikeat yhteydet tietämyslähtöisen suunnittelun menetelmien ja toteutuksen välille. Toisin sanoen oikeanlaiset yhteydet on löydettävä, jotta ohjelman suunnittelussa voidaan hyödyntää tietämyslähtöisen suunnittelun menetelmiä. Lisäksi kirjallisuudesta löytyy eri menetelmiä tietämyslähtöiseen suunnitteluun. Menetelmien sopivuus on myös huomioitava ohjelman suunnittelussa.

Tietämyslähtöisen suunnittelun yksi haasteellisimmista toimista on tietämyksen keräys ja mallintaminen. Yrityksissä yleensä tietämys on hajanaista ja saattaa sijaita monissa eri tietolähteissä. Usein suuri osa tarvittavasta tietämyksestä esiintyy vain insinöörien ammattitaitona. Muun muassa tähän haasteeseen tietämyslähtöisen suunnittelun menetelmät pyrkivät tuomaan ratkaisun.

ABB Oy Moottorit ja Generaattorit -yksiköllä on jo käytössään järjestelmiä ja menetelmiä, joiden avulla moottoreiden ja generaattoreiden suunnittelua voidaan tukea. ABB Oy:n ongelmana on, ettei Tahtikone -tuoteperheen yhteyteen ole suunniteltu ohjelmaa, joka tukisi suunnittelijoita osaluettelon tarkistuksessa. Tällaisen tuoteperheen tarpeeseen on tarkoituksena suunnitella osaluettelon validointityökalu, johon kerätään suunnittelijoiden tietotaitoa tietyistä tuotesuunnitteluun liittyvistä tehtävistä. Tämän tietovaraston avulla validointityökalun on tarkoitus kyetä tarkistamaan suunniteltavan osaluettelon kelpoisuus sekä tukemaan suunnittelijoita tehtävässään.

Tässä tutkimuksessa on tarkoituksena tutkia, voidaanko kirjallisuudesta löytyviä tietämyslähtöisen suunnittelun periaatteella tehtyjä ratkaisuja käyttää hyväksi ABB Oy:n ongelman ratkaisemiseen. Eli voidaanko tietämyslähtöistä suunnittelun lähestymistapaa käyttää hyväksi validointityökalun suunnittelussa. Lähestymistavan käyttö edellyttää, että löytyy sopiva menetelmä jota voidaan soveltaa ABB:n ympäristöön.

1.2 Tutkimuksen tavoitteet

Tämän tutkimuksen tavoitteena on suunnitella validointityökalu ABB Oy:n Tahtikoneet-yksikön tarpeeseen. Validointityökalun suunnitteluun on tavoitteena käyttää tietämyslähtöistä suunnittelua. Tämän lähestymistavan kautta pyritään saamaan tukea validointityökalun toteutukseen. Validointityökalusta rakennetaan prototyyppi, jonka avulla arvioidaan miten tietämyslähtöisen suunnittelun mukainen menetelmä sopii tarpeeseen.

Tutkimuksessa on kuitenkin ensin selvitettävä, mitä tietämyslähtöisen suunnittelun käsite tarkoittaa validointityökalun suunnittelussa. Tästä voidaan johtaa kaksi tutkimukselle oleellista kysymystä:

- Millä tavoin tietämyslähtöisen suunnittelun lähestymistapaa voidaan käyttää suunniteltaessa yritykselle sopivaa järjestelmää?
- Millainen menetelmä tukee validointityökalun toteutusta ja tietämyslähtöistä lähestymistapaa niin, että suunniteltava toteutus tukee ABB:n ympäristöä ja sen kehitystä?

1.3 Tutkimuksen rakenne

Tutkimuksen rakenne koostuu tietämyslähtöisen suunnittelun tutkimisesta ja sen yhdistämisestä validointityökalun suunnitteluun. Tietämyslähtöisen suunnittelun tutkimuksella etsitään menetelmä, jota voidaan käyttää validointityökalun suunnitteluun. Tällä tavoin suunnittelu pyritään saattamaan toteutuskelpoiseksi. Tutkimuksen aikana valmistetaan prototyyppi, jonka avulla selvitetään kuinka menetelmän käyttö on onnistunut.

Tutkimuksen luvussa 2 esitellään tietämyslähtöisen suunnittelun käsite sekä tuotesuunnitteluun liittyviä käsitteitä. Lisäksi samassa luvussa esitellään ABB Oy:n organisaatio ja sen teknologiat tuotesuunnittelussa. Esitutkimuksena luvussa 3 tutkitaan järjestelmän suunnittelua sekä tietämyslähtöisen suunnitteluun käytettyjä menetelmiä. Luvun tutkimuksen avulla etsitään menetelmä, jota voidaan käyttää validointityökalun kehityksen apuna. Luvussa 4 esitellään prototyypin suunnittelua ja siltä vaadittavat toiminnallisuudet. Luku 5 sisältää kuvauksen luvussa 3 valitun menetelmän käytöstä prototyypin yhteydessä. Luvussa 6 tarkastellaan tuloksia siitä, kuinka luvun 3 menetelmä sopii prototyypin käyttöön sekä kuinka hyvin prototyyppi sopii Tahtikoneet-yksikön tarpeeseen. Luvussa 7 on tutkimuksen yhteenveto.

2 TAUSTAA

Tämän luvun tarkoituksena on selventää ABB Oy:n tuotesuunnittelulle oleellisia käsitteitä sekä selvittää, mitä tarkoittaa tietämyslähtöinen suunnittelu. Luvussa esitellään yleisesti tehdastuotteiden tuotealusta sekä tuoteperhelähtöinen suunnittelu. Tehdastuotteiden tuoteperheen suunnittelusta jatketaan tietämyslähtöisen suunnittelun määritelmään. Lopuksi esitellään ABB Oy Moottorit ja Generaattorit -liiketoimintayksikön tuotesuunnittelu ja siihen liittyvät rakenteet.

Aluksi on kuitenkin syytä tehdä ero ohjelmansuunnittelun sekä tehdastuotteen suunnittelun välille. Tässä tutkimuksessa tehdastuotteista puhuttaessa, tarkoitetaan fyysisiä tuotteita, kuten moottoreita ja generaattoreita. Yleisesti suunnittelulla voidaan tarkoittaa molempia suunnittelun lajeja. Jotta epäselvyyksiltä vältyttäisiin, tässä tutkimuksessa puhutaan aina tehdastuotteiden suunnittelusta viitattaessa näiden fyysisten tuotteiden suunnitteluun.

2.1 Markkinapaineen vaikutus tuotteiden suunnitteluun

Vaihtelevien asiakasvaatimusten johdosta monet yritykset ovat kehittäneet toimintatapojaan. Yritykset pyrkivät tarjoamaan asiakkailleen enemmän räätälöityjä tuotteita menettämättä asiakastyytyväisyyttä laadun suhteen tai lisäämättä kustannuksiaan ja läpimenoaikojaan. (Simpson, Siddique & Jiao, 2006, 1.) Tällaisen markkinapaineen alaisuudessa suunnittelun, myynnin ja markkinoinnin on kehitettävä erityistä suunnittelutietämystä sekä määriteltävä hyvät suunnittelumenetelmät. Tällaisten suunnittelumenetelmien täytyy olla joustavia ja samaan aikaan niitä täytyy voida optimoida ja verifioida. Tällaista suunnittelumenetelmää ei voida toteuttaa, jos suunnittelijoilla ei ole tarvittavaa tietämystä aihealueesta tai he eivät voi työskennellä systemaattisella tavalla. (Nayak, Chen, Simpson, 2000.)

Viime vuosikymmenien aikana koko tehdastuotteiden suunnitteluprosessin tärkein uudistaja on ollut tietokonelähtöinen tiedon prosessointi. Tietokoneavusteinen suunnittelu (CAD – Computer Aided Design) vaikuttaa jo suunnittelumetodeihin, organisaatiorakenteisiin sekä työnjakoon. Tulevaisuudessa

lisääntyy tarve tietämyslähtöisille järjestelmille (KBS – Knowledge Based Systems), joiden avulla erityistä tuotetietoa sen komponenteista ja prosesseista voidaan hankkia kattavammin. Tällaisia järjestelmiä voidaan käyttää apuna analysoinnissa, optimoinnissa sekä ratkaisujen luonnissa. (Nayak ym., 2002.)

2.2 Tuotealustat ja tuoteperheet

Kiristyvän kilpailun markkinoilla monet tehdastuotteita valmistavat yritykset käyttävät hyväkseen tuoteperheitä, jotta ne voisivat monipuolistaa tuotevalikoimaansa tai lyhentää läpimenoaikojaan (Simpson ym., 2006, 3-5).

Tuoteperheellä tarkoitetaan samankaltaisten tuotteiden joukkoa, jotka on kehitetty yleisestä tuotealustasta lisäämällä tiettyjä toimintoja/piirteitä, jotta tuote vastaisi tiettyjä asiakasvaatimuksia. Tuoteperheen jokaista tuotetta kutsutaan *tuotevariantiksi*. Jokainen tuoteperhe rakennetaan vastaamaan tiettyä markkinasegmenttiä, kun taas tuotevariantti rakennetaan vastaamaan tiettyä asiakasvaatimusten joukkoa segmentin sisällä. Kaikki tuotevariantit sisältävät joitakin yleisiä ominaisuuksia ja/tai teknologioita. Näistä yleisistä osista muodostuu tuoteperheen *tuotealusta* (Du, Jiao & Tseng, 2001, 311).

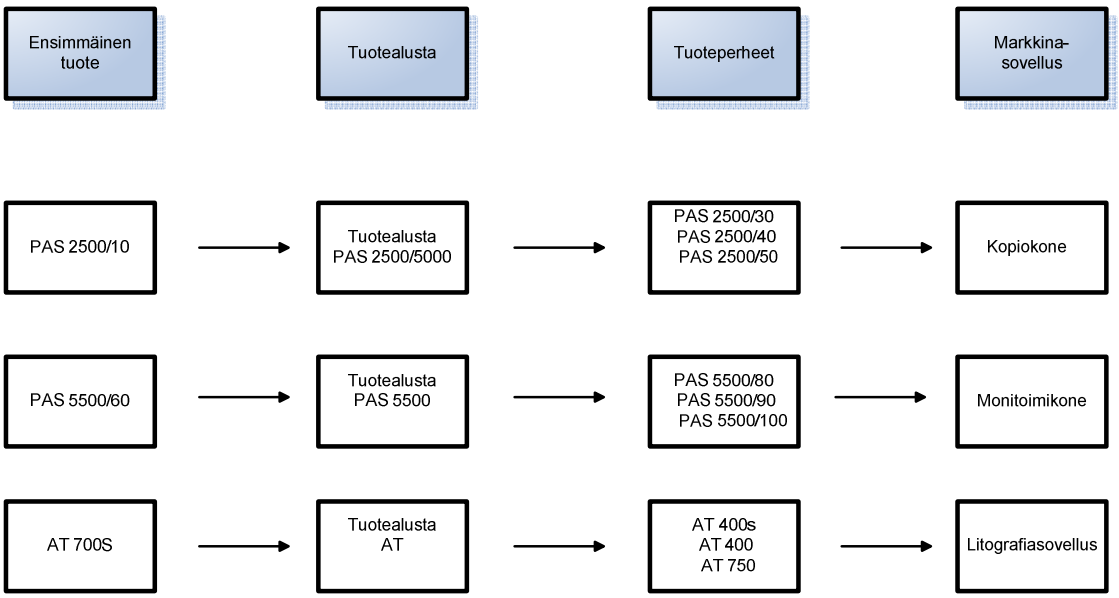
Tuotealustojen käsite on lähtöisin autoteollisuudesta. Yleisesti se tunnetaan lähestymistapana, joka yhdistää tuotteiden yleisen osan suunnittelusta sekä yleiset komponentit. Toimiva tuotealusta on edellytys toimivalle tuoteperheelle (Nayak, Chen & Simpson, 2002). Kirjallisuudessa tuotealustoille on annettu muun muassa seuraavanlaiset määritelmät:

- Joukko yleisiä komponentteja, moduuleja tai osia, joista johdetaan tuotteita, jotta ne voidaan suunnitella ja toimittaa tehokkaasti (Mayer, Lehnerd, 1997; katso myös Simpson ym., 2006, 3).
- Ominaisuuksien kokoelma (esimerkiksi komponenttien, prosessien, tiedon, ihmisten ja yhteyksien), jotka on jaettu tuotejoukkojen perusteella (Robertson, Ulrich, 1998; katso myös Simpson ym., 2006, 3).

2.3 Tuotealustalähtöinen suunnittelu

Tuotealustalähtöiseen tuoteperheen suunnitteluun liitetään yleisesti käsitteet tuotealusta, tuoteperhe sekä yksittäinen *tuote* eli tuotevariantti. Kustannus- ja aikatehokkuus sekä teknologinen etu voidaan saavuttaa, kun yritykset vaihtavat ajattelutavan yksittäisestä tuotteesta kohdistumaan tuoteperheisiin, jotka on kehitetty robusteja tuotealustoja hyväksi käyttäen. Tuotealustalähtöisestä tuoteperheen suunnittelusta voidaan saavuttaa hyötyjä monella tavalla. Potentiaalia on esimerkiksi korkeaan samankaltaisuuteen tuotteiden välillä ja erityisesti yhden sovelluksen sisällä olevien tuotteiden yhtenäistämiseen. (Hallman, Hofer & Van Vuuren, 2006, 31.)

Yleisesti tuotealustalähtöinen suunnittelu etenee kuvion 1 mukaisesti. Kuvassa havainnollistetaan Hallmanin ja muiden (2006, 38–40) mukaista tuotealustalähtöistä suunnittelua, erään yrityksen tuotteiden avulla. Tässä tuoteperhe on määritelty markkinoiden vaatimusten sekä käytettävissä olevien teknologioiden perusteella. Ensimmäinen kehitetty versio kustakin tuoteperheestä muodostaa tuotealustan kullekin seuraavalle samaan tuoteperheeseen kuuluvalle tuotteelle. Tästä eteenpäin tuotteet kehitetään tuotealustaa hyväksikäyttäen.



KUVIO 1 Esimerkki tuotealustalähtöisestä suunnittelusta. (Hallman ym., 2006, 39.)

2.4 Tietämyslähtöinen suunnittelu

Kilpailun kiristyessä yritykset pyrkivät minimoimaan kustannuksiaan tuotteiden valmistuksessa. Tämä tarkoittaa, että yritysten on kehitettävä tietopääomaansa sekä innovatiivisia ratkaisuja. Yritysten on otettava käyttöön yhä kehittyneempiä ja monimutkaisempia järjestelmiä, jotta ne voivat poistaa epävarmuustekijöitä ja turhia toistoja tehdastuotteiden suunnitteluprosessistaan. (Kulon, Broomhead & Mynors, 2005, 945.)

Yleensä tehdastuotteiden suunnittelijoiden pitää käsitellä monia muuttujia, jotka myös usein ovat ristiriidassa keskenään. Tämän lisäksi suunnittelijan täytyy hallita samalla tuotteidensa esteettinen puoli, valmistettavuus sekä toimintojen tavoitteet, jotta tuote pysyy kilpailukykyisenä. Jotta ongelmallisia muuttujia voidaan käsitellä, täytyy suunnittelijan voida kerätä ja hyödyntää tietoa, jota syntyy yrityksen monista lähteistä. Lisäksi tiukassa kilpailussa suunnittelijoiden on kyettävä tekemään kriittisiä päätöksiä ilman alkututkimuksia, joten

suunnittelijoilla on oltava paikkansapitävää tietoa sekä oikeaa osaamista oikeaan aikaan. (Ishikawa, 2003; katso myös Ammar-Khodja ym., 2008, 89.)

Suunnittelijoiden tieto voi vaihdella suunnittelijasta riippuen, joten sen käyttö ja varastointi voi olla suuri haaste. Yksi tapa tällaisen tiedon varastointiin sekä käyttöön on implementoida tietämyslähtöisiä järjestelmiä (KBS). Tällaisten järjestelmien voidaan määritellä olevan tietokoneistettuja järjestelmiä, jotka käyttävät tietyn alueen tietoa selvittääkseen ongelmia. (Ritchie & Harris, 1991.) Tietämyslähtöisen suunnittelun (KBE) voidaan katso olevan KBS-lähestymistavan osa-alue (Ammar-Khodja ym., 2008, 90).

2.4.1 Tietämyslähtöiset järjestelmät

Ensimmäisen sukupolven tietämyslähtöisten järjestelmien (KBS) katsottiin olevan asiantuntijajärjestelmiä (ES). Tällaiset järjestelmät sisältävät tietoa, jota ylläpidetään joukkona faktoja ja sääntöjä. Näitä faktoja ja sääntöjä voidaan kysellä ja manipuloida, jotta haluttuun ongelmaan saadaan ratkaisu tai selitys. (Ulengin & Topcu, 1997, 1065.)

Ensimmäisestä sukupolvesta eteenpäin tietämyslähtöiset järjestelmät ovat kehittyneet suuntaan, jossa tietoa pyritään varastoimaan monipuolisemmin ja tiedolle pyritään saamaan kattavampi merkitys. Esimerkiksi Akerkar ja Sajja (2010) tuovat esiin kehityksen suunnan, jossa tietämyslähtöiset järjestelmät toimivat tiedon syntetisaattorina. Tässä järjestelmät käsittelevät myös moraalia, käytänteitä sekä laillisuutta.

TAULUKKO 1 Perinteisten sekä tietämyslähtöisten järjestelmien vertailu (Sajja, 2000; katso myös Akerkar & Sajja, 2010, 18).

Perinteiset tietokoneavusteiset informaatiojärjestelmät	**Tietämyslähtöiset järjestelmät**
Antaa taatun tuloksen ja keskittyy tehokkuuteen.	Lisää voimaa päätöksentekoon ja keskittyy päätöksenteon tehokkuuteen, ilman takuuta tuloksesta.
Data ja/tai informaatiolähtöinen prosessointi	Tietämys- ja/tai päätöslähtöinen prosessointi.
Avustaa päätöksentekoa -sekä rutiini transaktioita koskevia tehtäviä.	Kykenee muokkaamaa, päivittämään ja selittämään päätöksiä.
Esimerkkijärjestelmiä ovat: TPS*, MIS** yms.	Esimerkkijärjestelmiä: asiantuntijajärjestelmät, malliperusteiset järjestelmät yms.
Manipulaatio tapahtuu numeerisesti ja algoritmien avulla.	Manipulaatio tapa on pääsääntöisesti symbolinen, yhdistävä ja ei-algoritminen.
Nämä järjestelmät eivät tee virheitä.	Nämä järjestelmät oppivat virheistään.
Tarvitsee täydellisen informaation ja/tai datan.	Osittainen tai epävarma data tai informaatio riittää.
Toimii reaktiivisesti laajalla alalla.	Toimii suppealla alalla reaktiivisesti sekä proaktiivisesti.

* TPS (Transaction Processing System) – tapahtumia prosessoiva järjestelmä, joka voi esimerkiksi tarkkailla ympäristöä ja prosessoida siitä tarkoituksen mukaista informaatiota.
** MIS (Management Information System) – järjestelmä joka analysoi liiketoimintatapahtumien rutiineja ja poikkeuksia.

Myös Ammar-Khodja ja muut (2008) tuovat artikkelissaan esille suunnan, jossa KBS -kehitys kulkee kattavamman tietovarastoinnin ja sen yhtenäistämisen suuntaan. He esittelevät KBS:n eri kategorioita, jotka ajan myötä ovat keskittyneet enemmän asiantuntijatiedon varastointiin sekä näiden tietovarastojen hyödyntämiseen. He viittaavat artikkelissaan muun muassa Shroben (1988) määritelmään malliperusteisista järjestelmistä. Tällaisissa järjestelmissä pyritään etsimään kattava kohdesysteemin malli, jota hyväksi käyttäen tietokonesovellus voi toimia systeemin asiantuntijana. Malleja sekä varastoitua tietoa hyväksikäyttäen järjestelmiä voidaan suunnitella kattavammin ja nopeammin.

Yleisesti KBS on tietokoneavusteinen järjestelmä, joka käyttää ja generoi tietoa datasta, informaatiosta, muusta tiedosta. Tällaiset järjestelmät voivat ymmärtää informaatiota, jota ne käsittelevät ja voivat tehdä päätöksiä tämän perusteella. Tämä erottaa KBS perinteisistä tietokoneavusteisista järjestelmistä (Akerkar ym., 2010, 18). Perinteisten tietokoneavusteisten järjestelmien ja KBS:n erot on hahmoteltu taulukossa 1.

2.4.2 Tietämyslähtöinen suunnittelu

Tietämyslähtöisen suunnittelun (KBE) käsitettä voidaan yleisesti kuvailla suunnittelun menetelmänä, jossa tietämys tehdastuotteesta varastoidaan tehdastuotteelle ominaisella tavalla ja käytetään hyväksi tehdastuotteen suunnittelussa, analysoinnissa sekä valmistuksessa. Yksiselitteistä määritelmää KBE:stä on todellisuudessa hyvin vaikea luoda. (Ammar-Khodja ym. 2008, 91.) Epämääräiseen määrittelyyn viittaavat myös Fan ja Bermell-Garcia (2008, 272) tutkimuksessaan, jossa he sanovat KBE-teknologian edustavan hämärää tiedettä monessa yrityksessä. Heidän mielestään KBE:n kehityksen pullonkaulana on järjestelmän sisältämän tietämyksen validointi.

Ammar-Khodja ja muut (2008) täsmentävät edelliseen, että KBE yleisesti mahdollistaa ratkaisun tiettyyn suunnitteluongelmaan sekä ylläpitää aihealueen tietämystä, jota tarvitaan tiettyjen tehdastuotteeseen liittyvien suunnitteluongelmien ratkaisemiseksi. Nykyaikaisten KBE-järjestelmien he kuvaavat olevan yhdistelmä aihealueen sääntöjä sekä oliokeskeisiä mallintamismenetelmiä.

Myös Lovett, Ingman ja Bancroft (2000) kuvailevat KBE:n olevan sääntöihin pohjautuvaa suunnittelua, jossa tietyn aihealueen sääntöjen avulla kuvataan saman alueen tietämystä. Lovett ja muut (2000) tutkivat KBE:n käyttöä sen liittyessä läheisesti tietokoneavusteisen suunnittelun (CAD) yhteyteen. He painottavat KBE:n määrittelyssä sen eroa tietämyslähtöisen järjestelmän (KBS) käsitteeseen. KBS kattaa paljon laajemman soveltamisen alueen ja saattaa sisältää

hienostuneempia tietämyksen tyyppejä. KBE yleensä keskittyy parametroituun mallintamiseen sekä suunnittelun tietämykseen, joka on sidottu tuotteen malliin.

Monet KBE:n määritelmät koskevat tiedon mallintamista. Penoyer, Burnett ja Fawcett (1999, 312) pyrkivät määrittelemään KBE-järjestelmät jakamalla ne eri kategorioihin määriteltyjen piirteiden perusteella. Piirteet he kuvaavat seuraavalla tavalla:

- Tietokonejärjestelmä, jota käytetään tehdastuotteen suunnittelun yhteydessä.
- Järjestelmä, joka on kohdistettu tiettyyn tietämyksen alueeseen tietyissä ongelmatapauksissa. Lisäksi järjestelmän hallinta on eriytetty tietämyksen malleista sekä riippumaton mallinnettavan tiedon eksplisiittisestä kontekstista.
- Järjestelmä, joka tunkeutuu syvälle ongelman aihealueeseen sekä pystyy hallitsemaan yksilölliset ongelmat eikä pelkästään sellaisia yleisiä ongelmia, jotka toistuvat kaikissa ongelmatapauksissa.
- Järjestelmä, joka käyttää ongelmanratkaisuun tapojen vertailua ja loogisia sääntöjä.

Penoyer ja muut (1999, 312) jakavat KBE-järjestelmät kahteen kategoriaan sen perusteella, miten ne esittävät tietoa. Tietoa voidaan esittää joko niin, että se on ihmisen tulkittavissa tai vastaavasti tietokoneen tulkittavissa. Ihmisen tulkittavissa olevat esitysmuodot sisältävät web-pohjaiset tiedon säilytyspaikat, parhaat käytänteet, opitun tietouden ja niin edespäin. Näissä tietämys on yleensä varastoitu HTML- tai tekstimuotoon.

Tietokoneen tulkittavissa olevat tiedon esitysmuodot ovat sellaisia, missä tietämyksen on tarkoitus olla tietokoneen luettavissa ja käytettävissä. Tällaiseen tietämyksen perustuvista järjestelmistä hyvänä esimerkkinä toimivat sääntöpohjaiset asiantuntijajärjestelmät. Tällaisissa järjestelmissä käyttäjä valmistelee ongelmatapaukset, hallitsee järjestelmän suoritusta sekä käsittelee lopputulosta. Tietokone hoitaa tarvittavan tietämyksen lisäyksen ongelman ympäristöön. (Penoyer ym., 1999, 313.)

Toinen tärkeä KBE-järjestelmien jako, Penoyerin ja muiden (1999) mukaan, tapahtuu geometrisen tiedon avulla. Tässä KBE-järjestelmät jaetaan sen mukaan, kuinka yksityiskohtaista geometristä tietoa järjestelmä sisältää. Tämän jaon toiseen ääripäähän jäävät järjestelmät, jotka perustuvat muuhun kuin geometriseen tietoon. Tällaisia ovat esimerkiksi:

- Tehdastuotteen suunnitteluprosessin alussa sijaitsevat ja laadulliseen arviointiin perustuvat järjestelmät.
- Tehdastuotteen ominaisuuksiin perustuvat hinnan mallinnusjärjestelmät sekä prosessisuunnittelu- ja aikataulutusjärjestelmät.
- Formaali ja looginen rakenteen mallinnus validointiin ja verifiointiin.
- Skemaattiset toimintamallinnusjärjestelmät.

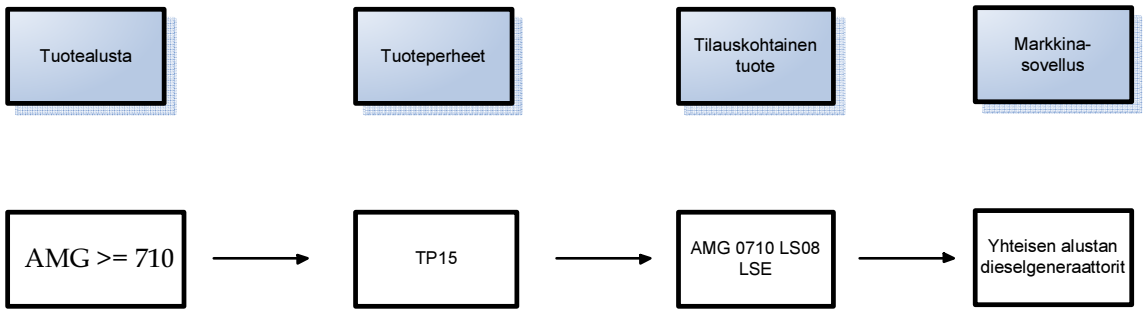
- Tietämyksen elinkaaren prosessointi, kuten esimerkiksi ylläpitoon liittyvät järjestelmät.
- Korkean tason geometrian mallintamiseen liittyvät järjestelmät.

2.5 ABB:n tuotesuunnittelu

ABB Oy on kansainvälinen sähkövoima- ja automaatioteknologiayhtymä, jonka palveluksessa työskentelee 117 000 henkilöä yli sadassa eri maassa. Ylimmän tason organisaatio koostuu viidestä eri divisioonasta ja jokainen divisioona koostuu yksiköistä. Moottorit ja Generaattorit -liiketoimintayksikkö kuuluu Sähkökäytöt ja kappaletavara-automaatio -divisioonaan. Yksikkö tarjoaa korkean hyötysuhteen moottoreita ja generaattoreita. (ABB, 2010.)

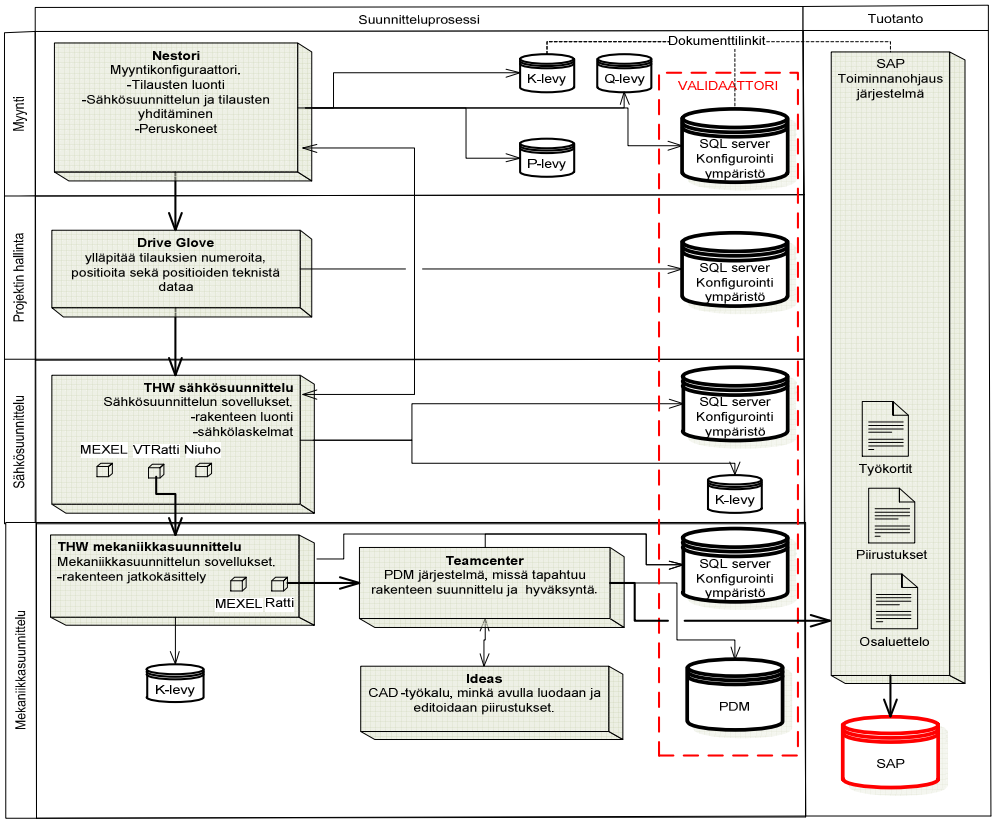
Moottorit ja generaattorit -liiketoimintayksikkö jakaantuu eri tuotteita valmistaviin osiin. Tuotteiden valmistukseen liittyvä tietämys, suunnittelu, prosessit, teknologia sekä käytänteet eroavat toisistaan eri tuotteiden mukaan. Tässä tutkimuksessa on tarkoitus kehittää validointityökalu tahtikoneiden valmistuksen yhteyteen. Tahtikone on tietyn tyyppinen sähkökone, mikä voi tarkoittaa roottoria tai generaattoria.

Tahtikoneiden suunnittelutietämys on jaettu tietämysalueisiin ja jokaiselle tietämysalueelle on nimetty omistaja, joka vastaa alueensa standardoinnista. Tietämysalueet on muodostettu käyttämällä hyväksi tuotealustalähtöistä lähestymistapaa. Tämä tarkoittaa, että tahtikoneet jakautuvat eri tuotealustoihin ja jokainen tuotealusta sisältää eri tuoteperheitä. Jokainen näistä tuoteperheistä sisältää yksittäisiä koneita eli tuotevariantteja. Jokainen tuotevariantti sisältää tuotealustalle ja tuoteperheelle ominaisia yleisiä teknologioita ja/tai ominaisuuksia. Esimerkiksi kuviossa 2 TP15 tarkoittaa isojen generaattoreiden tuoteperhettä, missä tuotevarianttia kuvaa lajimerkki ”AMG 0710 LS08 LSE”. Tässä tuotevariantti sisältää joitakin yleisiä teknologioita ja ominaisuuksia ”AMG >= 710” -tuotealustasta. Jokainen tilauskohtainen kone sisältää myös asiakaskohtaista tuoteräätälöintiä. TP15 sijoittuu markkinoilla segmenttiin, jota nimetään ”Yhteisen alustan dieselgeneraattorit” -segmentiksi.



KUVIO 2 Esimerkki ABB:n tuotealustasta.

Tahtikoneiden suunnittelussa käytettävät työkalut jakautuvat suunnitteluprosessin mukaisesti. Suunnittelussa käytetään eri sovelluksia ja tietokantoja riippuen siitä, millä tasolla suunnittelua ollaan. Kuvio 3 esittää tätä. Kuvassa ylimpänä prosessina on myynti, missä tarjous muokataan asiakkaalle sopivaksi käyttämällä hyväksi Nestori-myyntikonfiguraattoria. Vaiheen dokumentaatio kirjataan dokumenttityypin mukaan eri verkkolevyille ja tieto tilauksesta tallentuu SQL-tietokantatauluihin. Tilauksen kirjauksen yhteydessä määritellään suunniteltavalle työlle työnumero ja positiotieto. Työnumeron avulla jokainen tilauksen kone identifioidaan.



KUVIO 3 Tahtikoneiden suunnittelussa käytettävät järjestelmät.

Varsinainen sähkösuunnittelu tapahtuu THW sähkösuunnittelu -työkalulla ja sen aliohjelmilla. THW:llä tarkoitetaan ABB:llä kehitettyä ohjelmistoa, joka sisältää suunnitteluun ja myyntiin liittyviä ohjelmia. THW:n sähkösuunnittelusovellusten avulla suunniteltavalle koneelle tehdään sähkölaskelmat ja laskelmiin perustuva aktiiviosien rakenne. Prosessivaiheen tiedot päivitetään verkkolevyn dokumentaatioon sekä SQL-tietokantatauluihin.

Sähkösuunnittelusta siirrytään mekaniikkasuunnitteluun, jossa THW mekaniikkasuunnittelu -työkalun ja sen aliohjelmien avulla jatkokäsitellään rakennesuunnittelua. Jatkokäsittely voidaan suorittaa osittain automaattisesti RATTI -rakennekonfiguraattorin avulla, joka luo vakioidun osaluetelon koneelle. Mekaniikkasuunnittelua jatkokäsitellään Teamcenter -järjestelmän avulla, joka on tuotetiedon hallintaan (PDM) erikoistunut sovellus. Tämän avulla suunnittelijat muokkaavat rakennetta manuaalisesti. Suunnittelijat jatkavat suunnittelua Teamcenteriin liittyvän IDEAS ohjelman avulla. IDEAS on tietotokoneavusteiseen suunnitteluun (CAD) kehitetty ohjelmistopaketti, jonka avulla tahtikone voidaan mallintaa ja suunnitella. Teamcenterin data tallennetaan omaan tietokantaansa, josta se suunnitelun valmistuttua siirretään toiminnanohjausjärjestelmään (SAP). Tästä tiedon käsittely etenee koneen toimittajille sekä tuotantoon.

Jokainen kone koostuu sadoista materiaaleista. Nämä materiaalit jaetaan materiaali- ja alikokoonpanolistaksi eli osaluetteloksi. Suunnittelija suunnittelee osaluettelon edellä mainitun prosessin mukaisesti. Koneen koko elinkaaren ja erityisesti valmistusprosessin kannalta on hyvin tärkeää, että osaluettelo on kelvollinen eli validi.

Tällä hetkellä suunnittelijat tarkistavat rakenteen ja osaluettelon manuaalisesti ABB:n tietokannoista löytyvien suunnitteluohjeiden mukaisesti. Ohjeita on kuitenkin paljon ja niiden läpikäynti jokaisen rakennesuunnittelun aikana ei ole edullista. Tämän tutkimuksen tarkoituksena on suunnitella sähkökoneen validointityökalu kuvion 3 mukaisten suunnitteluprosessin tietokantojen yhteyteen niin, että työkalu tarkistaa tietokantojen taulujen tiedon yhtenäisyyden sekä toimii suunnittelijan apuna suunnittelussa.

2.6 Osaluettelo ABB:n ympäristössä

Osaluettelo on tuotteen komponenteista rakennettu hierarkkinen esitys, joka sisältää kriittisen informaation raakamateriaaleista, kappalemääristä, ohjeista sekä nimikkeistä. Sen päällimmäisenä tarkoituksena on määritellä äiti-lapsi-suhteet nimikkeille sekä niiden materiaaleille ja alikoonpanoille. Näiden suhteiden avulla osaluettelo tarjoaa tietoa, jonka avulla voidaan tehokkaasti suunnitella lopputuotteen valmistusprosessi sekä ylläpitää valmistukseen tarvittavien materiaalien varastosaldoja. (Romanowski ja Rakes, 2005, 249.)

ABB:llä osaluettelon komponentit eroavat yleisistä määrityksistä. Esimerkiksi osaa käytetään yleensä synonyyminä komponentille tai nimikkeelle. ABB.llä osalle on kuitenkin muodostunut oma tietty merkityksensä.

ABB:llä osaluettelon rakenne koostuu neljästä pääobjektista: *nimikkeestä*, *osasta*, *piirustuksesta* ja *materiaalista*. Näiden avulla toteutetaan hierarkkinen esitys, jonka sisältämä informaatio äiti-lapsi-suhteineen tarjoaa tarvittavat tiedot lopputuotteen valmistukseen. Osaluettelon objektien määrittely on ajan myötä muokkautunut tukeakseen eri järjestelmiä ABB:n sisällä. Esimerkiksi osaluettelo muokataan ja hyväksytään Teamcenter-järjestelmässä, mutta työnumerokohtaisen osaluettelon luonti tapahtuu RATTI-rakennekonfiguraattorilla. Tämä on yksi syy siihen, miksi ABB:n osaluettelon sisältämät komponentit ovat saaneet omat yksilölliset merkityksensä. Seuraavaksi esitellään ABB:n osaluettelon pääkomponentit ja niiden merkitykset:

- Nimikkeellä tarkoitetaan tietynlaista yksittäistä objektia osaluettelossa. Tämä komponentti voi olla jakamaton (esimerkiksi ruuvi) tai se voi olla toisista nimikkeistä koostuva alikokoonpano. Kullakin nimikkeellä on oma muuttumaton ja yksilöivä identiteettikoodinsa. Nimike sisältää sille ominaista perustietoa ja tämä tieto on yhtenäistetty DG:n (Drive Glove), Teamcenterin ja SAPin vaatimusten kanssa. (Karjalainen, 2010, 7.) Nimikkeitä, joilla on oma alirakenne, käytetään esimerkiksi alihankittavien osien yhteydessä. Toisin kuin osaa, nimikettä voidaan käyttää samalla identiteettikoodilla useissa eri tuotteissa. Tästä johtuen nimikettä kutsutaan vakioiduksi materiaaliksi/komponentiksi/kokoonpanoksi.
- Osalla taas tarkoitetaan komponenttia, jolla on myös oma identiteettikoodinsa ja joka rakentuu nimikkeistä ja muista osista koostuvista alikokoonpanoista (Karjalainen, 2010, 7). Osa on tuotteille valmistettava kokoonpano, missä kokoonpano tapahtuu joko omalla tehtaalla tai alihankkijan toimesta. Verrattuna nimikkeeseen, osa sisältää eri tietoa ja sen merkitys järjestelmissä on erilainen. Osan avulla tuoteperhekohtainen tieto siirretään tuote -eli työnumerokohtaiseksi. Esimerkiksi Teamcenterissä osan perustietolomakkeella ylläpidetään tietoa osan tunnuksesta. Osan tunnus on kiinteä ja näin ollen se ei ole yksilöivä tunnus, vaan sen voidaan kuvitella olevan yliluokka yksilöivälle tunnukselle. Osan tunnuksen avulla osalle voidaan määritellä tietyt pakolliset tiedot, jotka osan tulee sisältää. Lisäksi osan tunnusta käytetään luotaessa osaluettelon runko RATTI-rakennekonfiguraattorilla. Osan tunnusten avulla on määritelty vakiorakenne, jota hyväksikäyttäen RATTI luo työnumerokohtaisen rakenteen. Osan tunnukset ovat yleisiä tunnuksia, joten ne voivat viitata mihin tahansa työnumerorakenteeseen. RATTIn tuottama rakenne siirtyy Drive Glove -tietokantaan, josta osan tiedot siirretään muun rakenteen mukana Teamcenteriin. Teamcenterissä osalle annetaan oma yksilöivä identiteettinumero ja osan alle voidaan lisätä muita osia ja nimikkeitä työnumerokohtaisesti.

- Materiaalilla tarkoitetaan raaka-ainetta, josta osa tai nimike valmistetaan. Käytännössä materiaali ja nimike sisältävät saman perustietolomakkeen, mutta niiden käyttötarkoitus on eri. Materiaali voi esimerkiksi olla käsittelemätön metallilevy, josta muokkaamalla saadaan varsinainen nimike.

ABB:n osaluettelon arkkitehtuuri rakentuu edellä mainittujen komponenttien suhteista. Liitteessä 1 on esimerkki ABB:llä käytettävästä osaluettelosta. Siinä ensimmäisellä rivillä esiintyy äitiosa ja sen alapuolella äitiosan alirakenne. Tämä alirakenne koostuu piirustuksista, osista sekä nimikkeistä. Alirakenteen alapuolella on lisäksi tietoja osan mitoista sekä muuta lisätietoa.

Ensimmäisellä rivillä on lueteltu osan perustietoja. Ensimmäisenä vasemmalta ilmoitetaan osan taso. Tämä kuvaa osan paikkaa koko osaluettelossa. Liitteessä on osalle annettu tasonumeroksi 2, joka tarkoittaa että se kuuluu edellisen taso 1:llä olevan osan alle. Seuraavana on osalle ilmoitettu sen identiteettinumero, revisio, nimitys sekä muuta perustietoa. Nimitys-kohdassa ilmoitetaan myös osan tunnus, joka esimerkissä on ”ZA002”. Jos osan tilalla olisi kuvattuna rakenteellinen nimike, niin nämä tietokentät olisivat samat. Sen sijaan ero osan ja nimikkeen sisältämässä tiedossa ilmenee esimerkiksi edellä mainituissa lisä- ja mittatiedot-kohdissa. Jos kuvauskohteena olisi rakenteellinen nimike, niin näitä tietoja ei olisi.

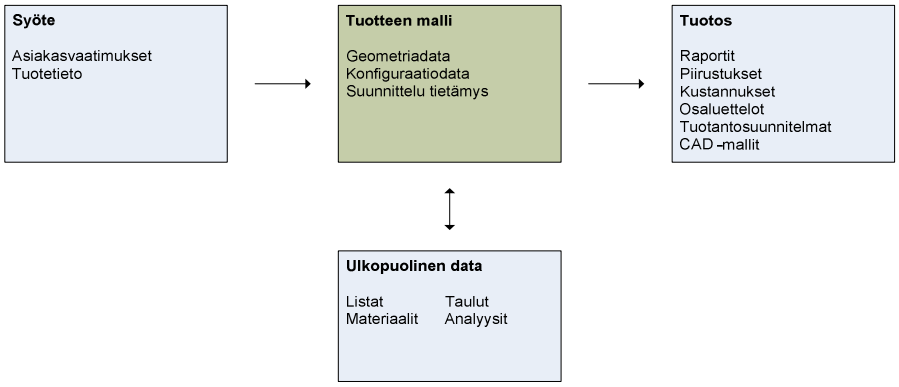
Osaluetteloa alaspäin mentäessä, kohdassa Osat/materiaalit/moduulit luetellaan äitiosan alirakenteen osat ja nimikkeet. Näille on lueteltu horisontaalisesti perustietoja ja näiden perustietokenttien nimet näkyvät osaluettelon kolmannella rivillä. Alkaen vasemmalta, ensimmäisenä ilmoitetaan osanumero, joka ilmaisee osan tai nimikkeen paikkaa. Tämä paikkatieto on äitiosakohtainen. Seuraavissa kohdissa on objekteille mainittu identiteettinumero, nimitys, lajimerkki sekä muuta perustietoa. Lajimerkki on vakio-osan yksilöivä tunnus. Alirakennelistassa ei näy materiaaleja, joita nimikkeet sisältävät. Tieto nimikkeelle kuuluvasta alirakenteesta löytyy PDM-järjestelmästä.

3 TIETÄMYSLÄHTÖINEN SUUNNITTELU KIRJALLISUUDESSA

Tässä luvussa perehdytään tietämyslähtöisen suunnittelun yhteydessä käytettyihin menetelmiin. Lisäksi esitellään tietämyslähtöiseen suunnitteluun liittyvät suositukset muun muassa arkkitehtuurista ja mallintamismenetelmistä.

3.1 Tietämyslähtöisen suunnittelun keskeisiä asioita

Tietämyslähtöistä suunnittelua voidaan kuvata yleisenä systeeminä, jonka tarkoituksena on tallentaa valmistusprosessin tietämys ja käyttää tätä tuotetietämystä iteratiivisella tavalla. Sandberg (2003) kuvailee tällaista systeemiä kuvion 4 tavalla. Kuvan systeemiin syötetään halutun tuotteen asiakasvaatimukset sekä tarvittava tuotetieto, ja niiden perusteella syntyy tuotteelle haluttu lopputulos. Lopputulos on tyypillisesti osaluettelo, piirustus, tuotantosuunnitelma tai raportti. Tämä lopputulos muodostetaan yhdistämällä asiakasvaatimukset systeemin sisältämään tietämykseen tuotteen yleisestä mallista tai muusta vastaavanlaisesta datasta. Lisäksi systeemi voi käyttää hyväkseen ulkopuolista tietoa esimerkiksi materiaaleista.



KUVIO 4 KBE yleisenä systeeminä (Sandberg, 2003, 5).

KBE-järjestelmän kaltaisen systeemin kehittäminen on iteratiivinen prosessi, jossa jokaisen iteraation aikana pyritään luomaan tai kehittämään prototyyppiä (Oldham, Kneebone, Callot, Murton & Brimble, 1998). KBE-järjestelmien kehitys iteratiivisessa ympäristössä on johtanut esimerkiksi RAD (Rapid Application Development) ja muiden tämän kaltaisten ohjelmistokehitysmallien käyttöön. (Chapman & Pinfold, 1999, 261.) RADin kaltaista ohjelmistokehitystä kuvataan luvussa 3.2.

Monien tutkijoiden mukaan KBE:n kaltaisen järjestelmän kehityksen tulisi olla luonteeltaan myös oliokeskeistä. Chapman ja Pinfold (1999, 259) kuvailevat KBE-menetelmien ilmentävän oliokeskeisen ohjelmoinnin kasvua, tekoälyä sekä CAD-teknologioita. Oliokeskeisyyden avulla mahdollistetaan tiedon esittäminen geneerisenä mallina, missä olio itsessään varastoi tiedon tehdastuotteesta ja perii muita toiminnallisuuksia yliluokaltaan.

Tietämyksen varastointi on yksi oleellinen vaihe KBE:n kaltaisten järjestelmien kehityksessä. Varastoitavalla tietämyksellä tarkoitetaan jaettavia kokemuksia, käsitteistöjä, arvoja, uskomuksia ja työtapoja. Yleensä tietämys on muodostettu tietokantaan ehtojoukoiksi, joiden avulla ohjelma voi mallintaa tietämystä. Yleisesti kerättävän tiedon täytyy olla kattavaa, jaettavissa olevaa ja uudelleenkäytettävää. Tiedon kattavuudella tarkoitetaan, kuinka yksityiskohtaista tietoa tuotteesta halutaan varastoida. Jos tietoa varastoidaan esimerkiksi implisiittisesti, niin tällöin täytyy ottaa huomioon, mikä on sen tiedon arvo joka jätetään varastoimatta. Vastaavasti täytyy huomioida, haittaako tiedon pois jättäminen ohjelman ongelmanratkaisukykyä (Sainter, Oldham, Larkin, Murton & Brimble, 2000, 2).

3.2 Nopea sovelluskehitys (RAD)

RAD (Rapid Application Development) tarkoittaa nopeaa sovelluskehitystä. Sillä on monia yhtäläisyyksiä muiden iteratiivisten kehitysmallien kanssa. Se korostaa käyttäjien mukanaoloa, prototyyppejä, uudelleenkäyttöä, automaattityökalujen käyttöä sekä pieniä kehitystiimejä. Tämän lisäksi RAD käyttää aikalaatikko-käsitettä, joka tarkoittaa tehtävien suorittamista tiettyjen ennalta suunniteltujen aikarajojen mukaan. Monissa muissa kehitysmalleissa vaatimukset ovat tiukkaan määrättyjä ja kehitysprojekti pyrkii toteuttamaan ne tietyssä arvioidussa ajassa. RAD-mallin mukaan aikarajat on määritelty alussa ja projekti pyrkii toteuttamaan sille asetetut vaatimukset aikataulun mukaisesti. Jos kuitenkin käy ilmi, ettei kaikkia vaatimuksia voida toteuttaa halutussa aikataulussa, niin vaatimuksia karsitaan pois tehtävälistalta. Näin ollen aikataulut eivät koskaan ylity. (Van Vliet, 2000, 57.)

RAD-ohjelmistokehityksen elinkaari kattaa neljä vaihetta (Van Vliet, 2000, 57):

- vaatimusten käsittely
- suunnittelu
- ohjelmiston rakennus
- käyttöönotto.

Vaatimusten käsittelyllä ja suunnittelulla on paljon yhtäläisyyksiä. Yhdessä näiden kahden vaiheen toteuttaminen vie normaalisti alle kaksi kuukautta aikaa. Aluksi pyritään keräämään oikeat vaatimukset. Jotta tämä onnistuisi, ovat sovelluksen tulevat käyttäjät prosessin avainasemassa. Vaatimukset priorisoidaan, koska on todennäköistä, että ensimmäisiin versioihin ei saada lisättyä kaikkia vaatimuksia. Näin ollen prototyyppeihin lisätään vaatimuksia prioriteettien mukaan. Vaatimusten priorisointia koskeva käsite tunnetaan nimellä *triage*. Yleisesti triage tunnetaan sotilasorganisaatioissa, joissa termi tarkoittaa haavoittuneiden ryhmittelyä vamman vakavuusasteen mukaan. (Van Vliet, 2000, 57–58.)

Ohjelman suunnitteluvaiheessa tavoitteena on tuottaa prototyyppi. Tässäkin vaiheessa käyttäjät ovat avainasemassa. Suunnittelu tapahtuu kahdessa vaiheessa: ensimmäisessä vaiheessa tuotetaan prototyyppi, joka luovutetaan käyttäjille katselmoitavaksi; toisessa vaiheessa suunnitelma viimeistellään käyttäjien kokemuksia hyväksi käyttäen. (Van Vliet, 2000, 58.)

Ohjelmiston kehitystapahtuu iteratiivisissa vaiheissa. Kehitys aloitetaan suunnitteluvaiheessa tehdyn prototyypin avulla. Kehitysvaiheessa prototyyppiä kehitetään edelleen lisäämällä siihen toiminnallisuuksia. Jokaisen iteraation jälkeen prototyyppi katselmoidaan käyttäjien toimesta, jolloin syntyy uusia tai muutettavia kehitysvaatimuksia. Tämä jatkuu kunnes prototyyppi hyväksytään käyttöönottoon. Käyttöönottovaiheessa suoritetaan viimeiset testaukset, järjestelmä asennetaan sekä käyttäjille annetaan tarpeellinen koulutus. (Van Vliet, 2000, 58.)

3.3 Tietämyslähtöisen suunnittelun menetelmiä

Insinööritietämyksellä on tapana olla hyvin monimutkaista, hajanaista sekä moniin asioihin sidoksissa olevaa. Tästä johtuen tällaisen tietämyksen mallintamisen on oltava rikasta ja hyvin rakennettua. (Klein, 2000.) Insinööritietämyksen monimutkaisuudesta johtuen tietämyksen mallintaminen on myös monimutkainen tehtävä. Suunnittelussa on otettava huomioon monet riippuvuussuhteet, jotta saadaan aikaan malli, joka on tarkka, geneerinen, kattava sekä tiivis (Sainter ym., 2000).

Tällä hetkellä löytyy hyvin vähän julkaistuja menetelmiä KBE-järjestelmien suunnitteluun. Yleisesti kirjallisuudessa on keskitytty enemmän KBS-järjestelmiin. Yleisimmin tunnettu KBS-menetelmä on nimeltään CommonKADS (Knowledge Acquisition Design System). Se tukee projektin hallin-

taa, organisaation analyysia, tietämyksen hankintaa, käsitteellistä mallintamista, käyttäjien vuorovaikutusta, järjestelmän käyttöönottoa sekä suunnittelua. (Lovett ym., 2000, 386.) KADS sisältää kattavat mallit sekä tukee vahvasti tietämykseen liittyvien tehtävien ja prosessien analysointia. KADSin mallit keskittyvät organisaation mallintamiseen, suoritettaviin tehtäviin sekä agentteihin, jotka ovat vastuussa tehtävien suorittamisesta. (Akerkar ym., 2010, 120.) Mulvennan ja Hugesin (1999; katso myös Chapman ym., 1999, 260) mielestä menetelmän täytyisi olla luonteeltaan oliokeskeinen, jotta se voisi tukea iteratiivista suunnitteluprosessia – iteratiivisuus sekä oliokeskeisyys ovat oleellisia tekijöitä KBE-järjestelmien kehityksessä. KADS ei kuitenkaan tue tämän tyylistä suunnitteluympäristöä.

Yksi harvoista KBE-järjestelmien suunnitteluun kehitetyistä menetelmistä on nimeltään MOKA (Methodology and software tools Oriented to Knowledge engineering Applications). Tämän menetelmän avulla järjestelmiä sekä tehdastuotteiden tuoteperheitä ja suunnittelua voidaan mallintaa oliokeskeisellä tavalla. MOKA-menetelmän avulla pyritään kuvaamaan tuoteperheiden rakennetta sekä sen suunnittelua oliokeskeisillä malleilla. Lisäksi menetelmä pyrkii tuomaan esille toiminnallisuuden sekä valmistettavan tuotteen suunnitteluun liittyvän käyttäytymisen. Toiminnallisuudella kuvataan geometrista tietoa sisältäviä olioita sekä tiedon organisointia. Käyttäytymisellä pyritään saamaan tekoäly toimimaan samalla tavoin, kuin tehdastuotteiden suunnittelijat toimisivat. Nämä mallinnettavat tietämyksen osat jakautuvat kahteen eri tasoon: vapaamuotoisiin ja formaaleihin malleihin. Menetelmän elinkaarimallin mukaan edettäessä mallinnus lähtee liikkeelle vapaamuotoisista ja täsmentyy formaaleiksi malleiksi. Formaalien mallien tarkoituksena on olla suoraan tietokoneen luettavissa. (Oldham ym., 1998.)

Toinen mainittava menetelmä KBE:n suunnitteluun on nimeltään KOMPRESSA (Knowledge-Oriented Methodology for the Planning and Rapid Engineering of Small-Scale Applications). Menetelmä on syntynyt osana REFIT-projektia (Revitalization of Expertise in Foundries using Information Technologies). Projektin tavoitteena on pienten ja keskisuurten yritysten kilpailukyvyn parantaminen. Keskeisimpiä asioita projektissa on vähentää järjestelmän suunnitteluun käytettävää aikaa, panostusta ja kustannuksia. (Lovett ym., 2000.) Menetelmän mallit ovat hyvin aktiviteettikeskeisiä. Lisäksi niiden avulla kuvataan tietämystä ja toiminnallisuuksia. Mallit jaetaan kahteen ryhmään: analyysi- ja suunnittelumalleihin. Analyysimallit keskittyvät organisaation prosesseihin, aktiviteetteihin ja tietämykseen. Ohjelman suunnittelumallit keskittyvät ohjelman toiminnallisuuksiin ja varastoitavaan tietämykseen. (Rasovska & Chebel-Morello, 2008).

Taulukossa 2 menetelmiä on kuvattu kuten edellä: tietämyksen rakenteen, mallien ja suunnittelun etenemisen kautta. Tietämyksen osien avulla kuvataan menetelmässä käytettävän tietämyksen mallintamiskohteita ja niistä muodostuvaa tietämysmallien rakennetta.

Rasovskan ja Chebel-Morellon (2008, luku: Methodologies of knowledge engineering) mukaan KADS kuvaa tietämystä reaalimaailman kuvausten kaut-

ta. Kuvaukset sisältävät tehdastuotteeseen liittyvät tehtävät, käyttöliittymän sekä aihealueen. Näiden kuvausten näkökulma on samanlainen erityisesti MOKAn tietämysrakenteen kuvausten kanssa. MOKAn mukaan tietämys muodostuu paitsi tietämysrakenteen, myös toimintojen ja käyttäytymisen mukaan. MOKAn tietämyksen toimintakuvauksilla on yhtäläisyyksiä KADSin käyttöliittymää kuvaavaan tietämyksen kanssa. Vastaavasti Lovettin ja muiden (2000) kehittämä KOMPRESSA muistuttaa MOKAa tietämyksen osien suhteen. KOMPRESSAn tietämyksen osat ovat kuitenkin hyvin aktiviteettipainotteisia. Lovettin ja muiden mukaan suurin ero näiden kahden menetelmän välillä tulee esille yksityiskohtaisemmassa menetelmän vertailussa ja siinä, että MOKA on keskittynyt juuri mekaanisen tuotteen ympärillä olevan tietämyksen mallintamiseen.

Taulukossa 2 kuvataan lisäksi millaisia malleja menetelmät sisältävät sekä millaisilla askelilla suunnittelu etenee. Verrattuna MOKA-menetelmään, KADSin suunnitteluosan heikkoutena ovat raskaat menetelmät. Lisäksi KADS ei ole tarkoitettu ainoastaan KBE:n kaltaisten järjestelmien suunnitteluun toisin kuin MOKA. (Lovett ym., 2000, 386; Oldham ym., 1998, 6.) Lisäksi aiemmin mainittu MOKA-mallien oliokeskeisyys on merkittävä ero verrattuna KADSiin.

TAULUKKO 2 Menetelmien vertailu

Menetelmä	**Tietämyksen osat**	**Mallit**	**Ohjelman kehityksen eteneminen**
KADS	Tehtävä Käyttöliittymä Aihealue	Organisaation tehtävät Agentit Tietämys Kommunikointi Suunnittelu	Tavoitteiden katselmukset Riskianalyysi Suunnittelu Valvonta
MOKA	Rakenne Toiminnot Käyttäytyminen	Vapaamuotoinen: Esimerkkikuvat, muuttujat, aktiviteetit, ehdot, entiteetit. Formaali: mekaaninen tuote, suunnitteluprosessi.	Tunnistus Oikeutus Tiedon hankinta Formalisointi Paketointi Välitys Esittely Käyttö
KOMPRESSA	Aktiviteetti Tietämys Toiminnot	Analyysi: tietämys, prosessi, aktiviteetti, organisaatio. Suunnittelu: toiminnot, tietämys.	Tutkimus Vaatimusten analysointi Työkalujen valinta Suunnittelu Implementointi Testaus Käyttöönotto Ylläpito

MOKAn ja KOMPRESSAN suurin ero suunnittelun etenemisen suhteen löytyy jälleen yksityiskohtaisemmalta tasolta. Lovett ja muut kuvaavat (2000,

387) KOMPRESSAN elinkaarimallia, joka sisältää pääpiirteittäin samat tehtävät kuin MOKA. KOMPRESSA-menetelmän elinkaarimalli on kuitenkin keskittynyt myös muihin ohjelman toiminnallisuuksiin kuin tietämyksen mallinnukseen ja formalisointiin liittyviin. KOMPRESSA-menetelmän käyttö on kuitenkin hyvin tapauskohtaista. Lovett ja muut mainitsevat, että heidän mallinsa antaa vain viitteen siitä mitä varsinainen prosessi saattaa sisältää. KOMPRESSA ei myöskään tue mallinnusta omalla mallinnuskielellään vaan käyttää yleisiä IDEF0 (Integration Definition for Function modeling) malleja. MOKAn yhteyteen sen sijaan on kehitetty oma mallinnuskieli MML (Moka Modeling Language).

Tämän tutkimuksen yhteyteen KADSia voidaan pitää liian raskaana. Niin kuin aiemmin tuotiin esille, KADS katsoo järjestelmäkehitystä paljon myös projektin johdon ja tulosten näkökulmasta. Niin kuin taulukosta 2 ilmenee, KADS-mallit sekä sen suunnittelun eteneminen ottaa selvästi huomioon projektin johdon näkökulman. Tästä johtuen KADS-menetelmästä voi olla hyvinkin monimutkainen ja hankalasti hallittava yksittäiseen tutkimukseen. Yleisesti KADSin soveltumattomuutta KBE:n kaltaiseen järjestelmäkehitykseen voidaan perustella KADSin painottuneisuutta KBS-järjestelmiin. Niin kuin aiemmin (2.4.2) todettiin, KBS kattaa paljon laajemman soveltamisen alueen ja saattaa sisältää hienostuneempia tietämyksen tyyppejä, kuin KBE. Tämän luvun perusteella voidaan tehdä päätelmä, että myös KADS on keskittynyt laajemmalle soveltamisen alueelle. Tällöin se ei ole käytettävyydeltään samaa luokkaa KBE:n kaltaisessa järjestelmäkehityksessä kuin esimerkiksi MOKA tai KOMPRESSA.

Tämän tutkimuksen menetelmävalintaan liittyen KOMPRESSA- ja MOKA -menetelmillä on paljon yhteisiä piirteitä kehitystyön elinkaaren näkökulmasta. Suurimmat erot syntyvät yksityiskohtaisemmalla tasolla. Tällä tasolla KOMPRESSA muuntautuu paljon tapauksen vaatimalla tavalla. Toisaalta esimerkkitapauksia KOMPRESSAN käytöstä ei ole paljon saatavilla. Lisäksi KOMPRESSA-menetelmä ei sisällä omaa mallinnuskieltään, toisin kuin MOKA.

Edellä esitettyjen näkökohtien perusteella MOKAa voidaan pitää parhaiten soveltuvana tämän tutkimuksen tarpeisiin. Menetelmä esitellään tarkemmin seuraavaksi.

3.3.1 MOKA-menetelmä

MOKA (Methodology and software tools Oriented to Knowledge engineering Applications) -menetelmä on lähtöisin ESPRIT MOKA-projektista, jolla on ollut tarkoituksena kehittää standardi menetelmä KBE-järjestelmien kehitykseen ja ylläpitoon (Sainter ym., 2000, 5).

MOKAn avulla KBE-järjestelmiä voidaan kuvata tasoina käyttämällä avuksi ehtoja, prosesseja, mallintamismenetelmiä ja määrityksiä. Se tarjoaa viitekehyksen tiedon esittämiseen ja varastointiin. Viitekehys voidaan jakaa kahteen tasoon: vapaamuotoiseen sekä formaaliin tasoon. Ensimmäinen taso on keskittynyt mallintamaan formaalia tietoa niin, että se on ymmärrettävissä ilman formaalin kielen tuntemusta. Tämän tason etuna on, että tarvittava tieto

voidaan myös validoida luotettavammin yhteistyössä tehdastuotteiden suunnitteluekspertien kanssa. Tämä kerros myös synnyttää tärkeää keskustelua eksperttien sekä järjestelmän kehittäjien välillä. Toinen taso sisältää formaalia tietoa ja se voidaan siirtää suoraan ohjelmaan. (Ammar-Khodja ym., 2008, 94.)

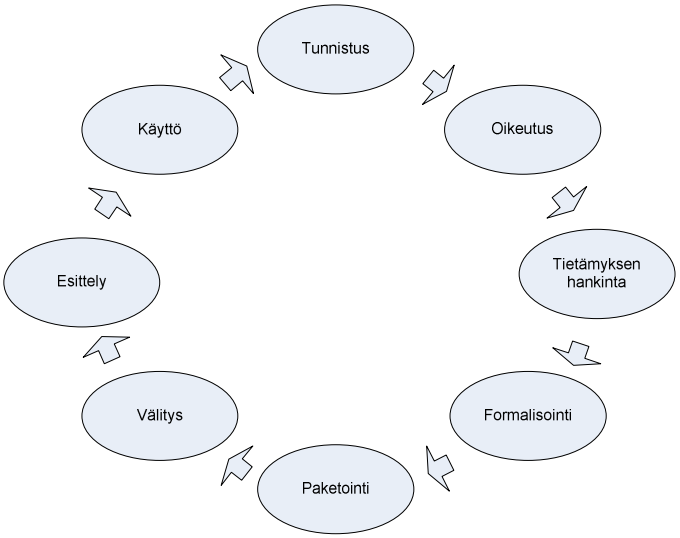
MOKA nimeää viisi geneeristä tietämyksen objektityyppiä ja määrittelee niiden yhteydet. Objektityypit kuvaavat koko KBE-järjestelmän viitekehystä. MOKA määrittelee nämä viisi objektityyppiä seuraavanlaisesti (Ammar-Khodja ym., 2008, 95):

- Esimerkkikuvat mallintavat kommentteja, kokemuksia, tiettyjä tilanteita sekä monimutkaisia selityksiä.
- Muuttujat kuvaavat tuotteen tai sen komponentin rajoituksia.
- Aktiviteetit kuvaavat ongelmanratkaisun vaiheita.
- Ehdot kuvaavat tietämystä ja niiden mukaan tehdyt päätökset ohjaavat aktiviteetteihin.
- Entiteetit kuvaavat tietämyksen elementtejä, joilla kuvataan tuotetta, sen komponentteja, asennusta, osia ja ominaisuuksia. Entiteetti voi olla rakenteellinen tai toiminnallinen.

MOKA määrittelee myös KBE-järjestelmien kehitykseen ja ylläpitoon elinkaarimallin. Kuviossa 5 esitetään elinkaari kahdeksana askeleena. Nämä askeleet on määritelty seuraavanlaisesti (Oldham ym., 1998, 3–5):

- Tunnistus: ennen kuin KBE-järjestelmään voidaan syöttää tietämystä, pitää tarvittavan tietämyksen olla tunnistettu. Tässä askeleessa tunnistetaan tietämyksen lähteet sekä työkalut ja käytänteet tiedon hankintaan ja sen esittämiseen.
- Oikeutus sisältää yrityksen tai projektin johdon hyväksynnän.
- Tietämyksenhankinta askeleen aikana kerätään tietämystä eri lähteistä. Tietämyksen avulla selvitetään esimerkiksi tuotteen rakenne ja kuvataan sitä luonnollisella kielellä. Rakenteen kattavuus ja oikeellisuus tarkistetaan käyttämällä hyväksi tietämyslähteitä. Askeleen aikana ei siis vain kerätä tarpeellista tietoa järjestelemän alueesta, vaan myös tuotetaan tietämyksen malli, josta on poistettu puhutun kielen epämääräisyydet.
- Formalisoinnissa analysoidaan tiedon hankinta-askeleen aikana tuotettu, vapaamuotoisesti rakennettu tietämys. Lisäksi tietämys muutetaan kattavammaksi ja formaalimmaksi, mutta kuitenkin niin että mallintamiskieli on riippumaton KBE-järjestelmästä.
- Paketointiaskeleessa muokataan formalisoitu tietämys sellaiseen muotoon, että se voidaan siirtää KBE-järjestelmään. Kun tietämyksen siirto järjestelmään on tapahtunut, niin järjestelmää testataan. Testauksessa selvitetään, ovatko käytetty kieli ja käyttöliittymä sopivia käyttäjille. Lisäksi testauksessa selvitetään täyttääkö järjestelmä vaaditut tehokkuuden ja suorituskyvyn vaatimukset.
- Jakelu varmistaa, että rakennettu järjestelmä on lähetetty kaikille potentiaalisille käyttäjille, jotka tulevaisuudessa voisivat siitä hyötyä.

- Esittelyaskel sisältää kaikki mahdolliset fyysiset asennukset sekä mahdolliset koulutukset.
- Käyttö tarkoittaa viimeistä askelta, jossa haluttu liiketoimintahyöty on saavutettu, järjestelmän toimiessa insinöörien toiminnan tukena. Tämä voi johtaa uusien tietämystyyppien löytymiseen tai uusien vaatimusten syntymiseen, jolloin prosessi aloitetaan alusta uutena iteraationa.



KUVIO 5 KBE järjestelmien elinkaari.

MOKA-menetelmä keskittyy tukemaan erityisesti tietämyksen hankintaa, formalisointia sekä sovelluksen paketointia. Nämä askeleet liittyvät läheisesti myös järjestelmän ylläpitoon (Oldham ym., 1998, 5).

3.3.2 MOKA-mallintamiskieli

MOKA tarjoaa myös mallintamiskielen (MML – MOKA Modeling Language), mikä on tärkeää KBE-järjestelmän kehityksessä. MML:n avulla voidaan esittää suunnittelun parhaat käytännöt luokkien, assosiaatioiden sekä attribuuttien avulla. Mallintaminen tapahtuu rakenteellisella ja loogisella tavalla. Tavoitteena on tarjota suunnittelijalle viitekehys ja ohjeistus mallintamiseen. (Brimble & Sellini, 2000, 50.)

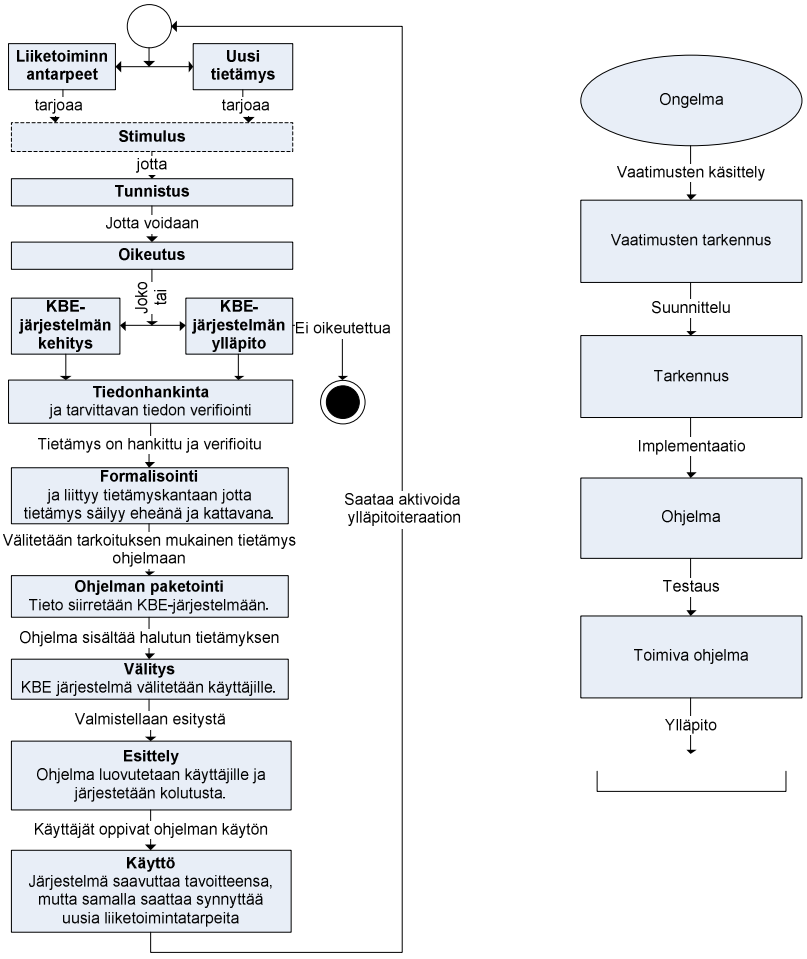
MML on laajennus yleisesti tunnetusta oliokeskeisestä mallinnuskielestä UML:stä (Unified Modeling Language). MML sisältää seuraavia lisäyksiä UML:n malleihin (Brimble ym., 2000, 50–51):

- Ennalta määrätyt näkymät – tarjoaa eri näkökulmia vallitsevaan malliin ja tietyn sisällön diagrammeihin.
- Ennalta määrätyt luokat – esittävät metaluokkia tunnistaen niiden luokkien tyypit, jotka käyttäjän tulisi luoda edelleen. Esimerkiksi <Osa> on ennalta määrätty luokka ja käyttäjien oletetaan luovan <Osa>-tyypin luokkia.

- Ennalta määrätyt luokkien attribuutit – sisältää yleisesti käytetyt attribuutit, tietylle luokalle.

3.4 KBE-järjestelmän suunnittelu

Tietämyslähtöisen suunnittelun (KBE) järjestelmille luotu MOKA-menetelmä pohjautuu kuvion 6 vasemmanpuoleiseen malliin, joka esittää KBE-järjestelmänkehityksen elinkaarta. Kuviossa 6 esitetään sama KBE-järjestelmäkehityksen elinkaari kuin aiemmin kuviossa 5. Tässä askeleiden välistä siirtymätilaa on kuvattu sanoilla tai lauseilla, jotka kuvaavat aiemman askeleen lopputuloksen siirtymistä seuraavaan.



KUVIO 6 MOKA-menetelmä ja yleinen ohjelmistokehitys (Oldham ym., 1998, 4; Van Vliet, 2000, 11).

MOKA-menetelmä antaa kuitenkin hyvin vähän viitteitä siihen, kuinka itse ohjelmaa tulisi suunnitella muiden toiminnallisten vaatimusten kuin tehdastuotteiden suunnittelua koskevan tietämyksen keräystä ja mallintamista ajatel-

len. Lisäksi MOKA ei esitä tarkasti, mitä vaatimusten keräysprosessi sisältää. Tähän ongelmaan voidaan käyttää perinteisiä ohjelmistotuotannon metodeita.

Van Vlietin (2000, 11) malli kuvion 6 oikealla puolella kuvaa yksinkertaisesti ohjelmistokehitysprosessin vaiheita, missä ongelmasta tuotetaan vaatimukset ja näitä tarkentaen päästään toimivaan ohjelmaan. Ensimmäisenä ratkaistava ongelma analysoidaan ja mallinnetaan hyvin selkeällä tavalla. Näihin vaatimuksiin perustuen tehdään ohjelman suunnitelma. Viimeisenä tulee ohjelman luomisvaihe, missä varsinainen ohjelmakoodin kirjoitus tapahtuu.
Kuvan 6 oikeanpuoleisen mallin avulla Van Vliet (2000) käy läpi suunnitteluprosessia ja esittää menetelmiä, joilla vaatimusten käsittely ja ohjelmansuunnittelu voidaan toteuttaa. Van Vlietin ja muiden suunnitteluprosessiin liittyviä menetelmiä tuodaan esille seuraavaksi.

3.4.1 Vaatimusten käsittely

Aiemmin esiteltiin MOKA-menetelmä ja sen kautta KBE-järjestelmän elinkaarta. MOKA-menetelmä keskittyy erityisesti kolmeen askeleeseen KBE-järjestelmän elinkaaresta. Nämä askeleet ovat tiedon hankinta, formalisointi ja sovelluksen paketointi. Erityisesti tiedon hankinta- sekä osittain myös formalisointiaskeleet sisältävät operaatioita, joita kirjallisuudessa on yleisesti tutkittu käsitteellä RE (Requirements Engineering) eli vaatimusten käsittely. Seuraavaksi käsitellään yleisen ohjelmistokehityksen mukaista vaatimusten käsittelyä.

Vaatimusten käsittely on ensimmäinen suuri askel ohjelmistojärjestelmän kehityksessä ja suunnittelussa. Tämän vaiheen aikana identifioidaan ja dokumentoidaan käyttäjien vaatimukset tulevasta järjestelmästä. Vaatimusten käsittely on iteratiivinen ja yhteistyössä tapahtuva ongelman analysointiprosessi, jossa dokumentoidaan vaatimukset ja varmistetaan dokumentoitujen vaatimusten oikeellisuus. (Van Vliet, 2000, 10.)

Vaatimusten käsittelyprosessi sisältää yleisesti vaatimusten selvittämisen, vaatimusanalyysin, vaatimusten tarkennuksen sekä validoinnin. Vaatimusten selvittämisvaiheen tarkoituksena on selvittää ja kerätä suunnitteluprojektin osapuolilta vaatimukset. Vaatimusten keräys tapahtuu kokouksissa, joissa ohjelmiston kehittäjät ja kohdealueen ekspertit kokoavat yhdessä vaatimukset. Vaatimusanalyysin aikana varmistutaan siitä, että kaikki osakkaat ymmärtävät vaatimukset oikealla tavalla. Lisäksi vaatimusanalyysin aikana pyritään poistamaan virheitä, aukkoja sekä muita puutteita. Vaatimusten tarkennusvaiheessa pyritään tarkentamaan vaatimuksia ohjelman toiminnallisten sekä ei-toiminnallisten vaatimusten osalta. Tässä vaiheessa tuotetaan vaatimusten käsittelyn lopputulos. Validointivaiheessa kirjoitetaan vaatimusten pohjalta testitapauksia, joiden avulla vaatimukset voidaan testata ja verifioida. Vaatimusten käsittely ei kuitenkaan ole lineaarinen prosessi, vaan se toteutuu sykleittäin. Edellä mainitut vaatimusten käsittelyyn liittyvät toimet toteutuvat lomittain, iteratiivisesti ja inkrementaalisesti. (Wiegers, 2003, 43–59.)

Vaatimukset voidaan jakaa kahteen luokkaan ja näitä kutsutaan toiminnallisiksi ja ei-toiminnallisiksi vaatimuksiksi. Toiminnalliset vaatimukset kuvaavat järjestelmältä vaadittavia toimintoja ja ei-toiminnalliset kuvaavat esimerkiksi suorituskykyä, hintaa, käyttäjien koulutuksta ja niin edespäin. (Van Vliet, 2000.) Hunter, Rios, Perez ja Vizan (2005, 685) toteavat, että kun puhutaan tietämyslähtöisten järjestelmien kehityksestä, niin löytyy kahdenlaisia toiminnallisia vaatimuksia, jotka täytyy huomioida ja dokumentoida. Ensimmäinen näistä on kehitettävän sovelluksen toiminnalliset vaatimukset. Toinen koskee tehdastuotteiden toiminnallisia vaatimuksia.

3.4.2 KBE-järjestelmän rakenne

Yleensä KBE-järjestelmän kehityksessä on kyse jostain tuotteesta, jonka suunnittelun avuksi järjestelmää kehitetään. Usein vaatimuksena onkin järjestelmän yhteensopivuus tuotteen kanssa. Tästä johtuen vaatimusten tarkennuksen tulisi sisältää jonkinlaista tuotesuunnittelun tietoa, mutta niin ettei se ole tarpeetonta. Tällöin ei jouduta tilanteeseen, jossa luotaisiin turhia rajoitteita tulevalle ohjelmalle. (Wiegers, 2003, 304.)

Esittelemällä järjestelmän tulevaa arkkitehtuuria vaatimusten käsittelyn yhteydessä asiakkaille annetaan laajempi näkökulma suunniteltavasta ohjelmasta. Tämän avulla pystytään edelleen varmentamaan vaatimukset paremmin sekä säätämään niiden täsmällisyyttä. Arkkitehtuuri on erityisen kriittisessä asemassa silloin kun järjestelmä sisältää sekä laitetason että ohjelmakomponentteja tai suunniteltava ohjelma on hyvin monimutkainen. (Wiegers, 2003, 304.) Oletettavasti edellä mainitut asiat pätevät myös KBE-järjestelmiin.

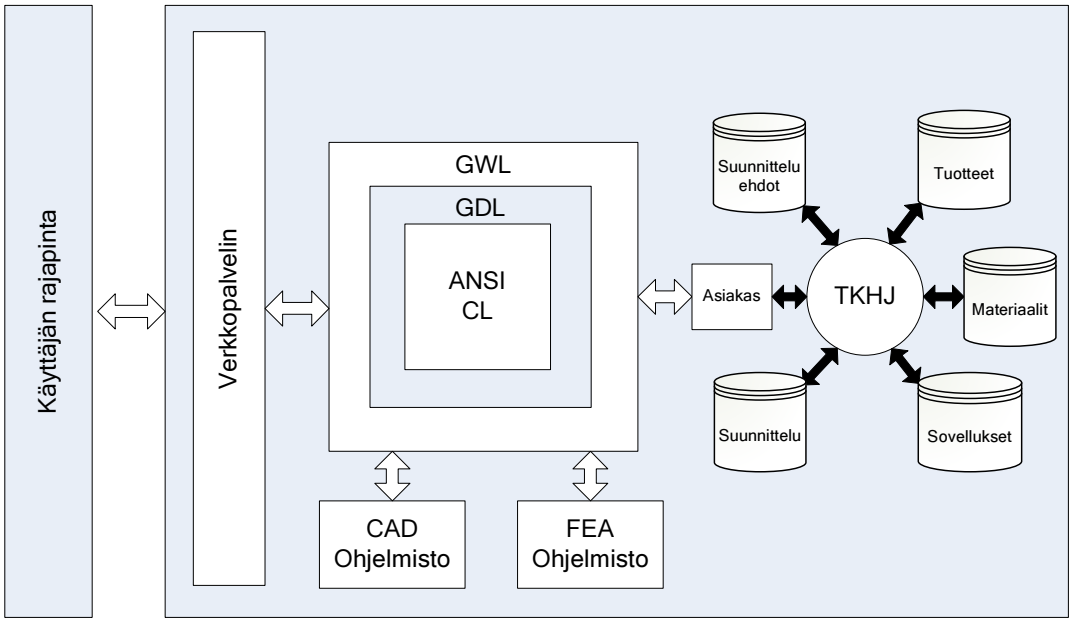
Kulon ja muut (2005) ovat tutkineet KBE-järjestelmän suunnittelua ympäristöön, jossa tarkoituksena ei ole täysin automatisoida suunnitteluprosessia, vaan luoda KBE järjestelmä, joka tukee suunnittelua. He esittelevät tapauskohtaisen arkkitehtuurimallin, missä KBE-järjestelmä kehitetään internet-perusteisesti. Kuviossa 7 on Kulonin arkkitehtuurimalli, jossa näkyvät järjestelmään kuuluvat eri komponentit ja niiden suhteet. Vasemmalla puolella kuvassa on käyttöliittymä ja oikealla puolella kuvataan tapahtumia KBE-verkkopalvelimen sisäpuolella. Tämän KBE-järjestelmän ydinkomponentteina toimivat GDL (General-purpose Declarative Language) ja GWL (Generative Web Language). GDL ja GWL ovat Genworks Internationalin kehittämiä tietämyslähtöisiä ohjelmointiympäristöjä. GDL ja GWL yhdessä yhdistävät sovelluksen logiikan sekä tietokannat. GWL on verkkopalvelimen puolella toimiva käyttöliittymä, jonka avulla hallitaan dataa sekä muita ohjelmistotyökaluja. Päätehtävät, joita kuvan GWL-kerros hallitsee, ovat

- tietokantahakujen hallinta
- tiedostojen siirron helpottaminen käyttäjän ja CAD-paketin tai FEA-ohjelmiston välillä
- KBE-järjestelmän toimitusten suorittaminen.

GDL perustuu oliokeskeisten kielten mallinnukseen ja sitä käytetään monimutkaisten objektien ja järjestelmien mallinnuksessa. Se sisältää myös

oliokeskeisen ANSI Common Lisp–kielen, joka on muunnos perinteisestä Lisp-kielestä. Yleisesti GDL:n avulla voidaan rakentaa web-pohjaisia sovelluksia, jotka käsittelevät dataa ja luovat niistä graafisia esityksiä (Genworks, 2011).

Kuvion 7 FEA-ohjelmisto (Finite Element Analysis) on Kulonin tutkimustapaukseen liittyvä ohjelmisto, jonka avulla suunnittelijat voivat simuloida suunnitelmiaan. CAD-ohjelmistot tarkoittavat tietokoneavusteisia suunnitteluohjelmia kuten AutoCAD. Kuvan esimerkissä nämä järjestelmät voivat käyttää esimerkiksi GDL-kielellä toteutettuja graafisia esityksiä tai ottaa vastaan Common Lisp kielellä toteutettuja datapaketteja.



KUVIO 7 KBE arkkitehtuuri (Kulon ym., 2005, 948).

Kulonin malli kuviossa 7 kertoo, kuinka KBE-järjestelmä voi liittyä muiden suunnittelussa käytettävien ohjelmistojen yhteyteen. Lisäksi kuvasta erottuu KBE-järjestelmän sekä tietokannan yhteys. Hou, Liu, He, Zhao ja Wang (2010) tutkimuksessaan esittävät vastaavanlaisen mallin KBE-järjestelmän arkkitehtuurista. Heidänkin mallissaan käy selvästi ilmi tietokannan sekä KBE:n ydinkomponentin erottelu.

3.5 Kirjallisuuskatsauksen yhteenveto ja päätelmiä

Edellä käsiteltiin KBE-järjestelmän kehitykseen liittyviä asioita. Pyrkimyksenä oli tehdä katselmus kirjallisuuteen siitä, millaisia tietämyslähtöisen suunnittelun menetelmiä on käytetty järjestelmien suunnittelussa ja millä tavoin . Erityi-

sesti tavoitteena oli tutkia, kuinka tietämyslähtöistä suunnittelua voitaisiin käyttää hyväksi suunniteltaessa ohjelmaa ABB Oy:n tarpeeseen.

Luvun 3 alussa tuotiin esille joitakin keskeisiä asioita tietämyslähtöisestä suunnittelusta. Tässä esiteltiin suositeltavia lähtökohtia järjestelmän mallinnukseen, tietämyksen keräykseen sekä suunnitteluprosessiin. Luvussa käy ilmi, että KBE-mallien tulisi tukea oliokeskeistä lähestymistapaa. Lisäksi KBE-järjestelmiä suositellaan suunniteltavaksi iteratiivisesti sekä prototyyppejä luomalla. Tällä tavoin suunnittelu tukee parhaalla mahdollisella tavalla eksperttien tietämyksen hankintaa ja varastointia. Suunnitteluprosessin läpivientiin on tarjolla monenlaisia menetelmiä.

Toisin kuin KBS:ään, KBE-järjestelmien suunnitteluun ei ole kehitetty kovinkaan monta menetelmää. Yleisimmin tunnettu ja tutkimuksissa noteerattu on MOKA. Tämä menetelmä tarjoaa viitekehyksen tietämyksen hankintaan ja implementointiin sekä lisäksi mallintamiskielen MML. MOKA keskittyy erityisesti tietämyksen hankintaan ja sen muuntamiseen formaaliin muotoon, mutta se ei anna selvää mallia tietoa käyttävän sovelluksen kehittämiseen. Verrattuna KOMPRESSAan tai KADSiin MOKA-menetelmän voidaan katsoa sopivan paremmin tämän tutkimuksen yhteyteen. Ohjelman yleiseen suunnitteluun löytyy kirjallisuudessa paljon malleja ja menetelmiä. Luvussa tarkasteltiin erityisesti Van Vlietin (2000) ja Wiegersin (2003) mukaan suunnittelun keskeisiä asioita, kuten vaatimusten käsittelyä.

Kirjallisuuskatsauksen perusteella nousee päällimmäisenä esille, että tietämyslähtöinen suunnittelu keskittyy erityisesti tietämyksen mallintamiseen sekä tietämysvaraston ja tietämystä käyttävän ohjelman yhdistämiseen. KBE:tä koskevat tutkimukset eivät tuo esille tietämystä käsittelevän ohjelman suunnittelua koskevia menetelmiä. Tämä tarkoittaa, että itse ohjelma voidaan suunnitella perinteisiä menetelmiä mukaillen niin, että menetelmät tukevat KBE -lähtöistä suunnittelua prosessien, mallien sekä menetelmien osalta. Ohjelmiston suunnittelu ABB Oy:n tarpeeseen tarkoittaa silloin, että:

- suunnitteluprosessi on iteratiivinen, missä tehdastuotteen suunnittelijat sekä järjestelmän suunnittelija toimivat aktiivisessa yhteistyössä,
- suunnitteluprojekti on prototyyppilähtöinen, eli suunnittelun aikana tuotetaan ja kehitetään prototyyppiä,
- suunnittelumallit ovat oliokeskeisillä kielillä toteutettuja,
- suunnitelman tulee antaa tehdastuotteen suunnittelijoille selkeyttävä rakennekuvaus jo alkuvaiheessa.

4 VALIDOINTITYÖKALUN KEHITYS

Luvussa 3 esiteltiin MOKA-menelmää sekä yleistä ohjelmistokehitystä. Tuloksena tästä luvusta mainittiin, ettei MOKA-menetelmä tuo selvästi esille muuta suunnittelua kuin tehdastuotteiden keräystä ja mallintamista koskevaa.

Tässä luvussa perehdytään validointityökalun kehitykseen. MOKA-elinkaaren mukaisesti päätös validointityökalun kehityksestä tehdään varhaisessa vaiheessa. Tässä vaiheessa tehdään päätös siitä, tarkoittaako kehitys olemassa olevan järjestelmän jatkokehitystä vai uuden järjestelmän luontia. ABB:n ympäristössä on alustavasti mahdollisuus näihin molempiin vaihtoehtoihin. Jotta päätös voidaan tehdä, tarvitaan tietoa toteuttavan validointityökalun vaatimuksista. Tästä johtuen päätöksentekovaiheeseen käytetään hyväksi RADin mukaisia vaatimusten käsittelyprosessia sekä Wiegersin (2003) mallia dokumenttien rakentamisesta. Lisäksi määritellään tietämyksen keräystä koskeva menetelmä ICARE, koska oletettavasti vaatimusten keräys liittyy oleellisesti validointityökalun toimintoihin ja tätä kautta myös siis vaatimuksiin. Validointityökalun kehityksen edetessä vaatimusten käsittelyä ja tietämyksen keräysmenetelmää voidaan käyttää hyväksi jatkamalla kehitystä ohjelmistokehitysmallin ja MOKA-menetelmän mukaisesti.

4.1 Vaatimusten käsittely

RAD-kehitysmalli lähtee liikkeelle vaatimusten käsittelyvaiheesta. Vaiheen tarkoituksena on kerätä käyttäjien vaatimukset sekä tekniset vaatimukset. Vaatimusten keräykseen käytetään apuna Wiegersin luomia dokumenttipohjia.

RAD-kehitysmallin mukaisesti käyttäjien vaatimuksia kartoitetaan yhdessä tahtikonesuunnittelijoiden kanssa. Kartoitus tapahtuu viikoittaisissa kokouksissa, joista ensimmäisessä kehitetään visio validointityökalun suunnittelusta ja toteutuksesta. Visio kirjataan vaatimusten käsittelydokumenttiin. Tässä tutkimuksessa dokumenttimallina käytetään Wiegersin (2003, 9) ”Vision and Scope” –dokumenttia. Vaatimusten käsittelydokumentti sisältää liiketoimintavaati-

mukset, vision, ohjelman laajuuden ja rajoitukset. Käyttäjien toimintavaatimuksia lähdetään kartoittamaan seuraavien kysymysten avulla:

- Mistä saadaan kriteeristö? Kriteerien hankkiminen ei kuulu tämän tutkimuksen piiriin. Toteutuksen kannalta voidaan kuitenkin miettiä, voidaanko esimerkiksi validointikriteerit ottaa referenssityöltä.
- Millaiseen muotoon tarkistuksen tulos tuotetaan? Tähän liittyen täytyy ottaa huomioon millaiseen käyttötarkoitukseen tulostiedostoa käytetään.
- Voidaanko työkalun toiminnot jakaa tuoteperheisiin? Tämä tarkoittaa, että työkalun kriteerejä voitaisiin luoda tuoteperheittäin.
- Mihin tarkistus osaluettelossa voidaan kohdistaa? Tässä on kiinnitettävä huomiota yksittäisen osan metatiedon yhtenäisyyteen. Esimerkiksi pysyykö osan nimi aina samana.
- Mitkä ovat muut ohjelmalta halutut toiminnallisuudet?

Vaatimusten käsittelydokumenttia tarkennetaan seuraavissa kokouksissa. Tarkennuksissa dokumenttipohjana käytetään hyväksi Wiegersin (2003, 9) SRS-dokumenttia (Software Requirements Specification). Tätä dokumenttia mukaillen ohjelman vaatimuksiin täsmennetään

- yleiset toiminnallisuudet
- käyttäjät
- ohjelman ympäristö
- dokumentointi
- olettamukset ja riippuvuussuhteet
- ulkoisten rajapintojen vaatimukset
- tulevaisuuden näkymät
- ei-toiminnalliset vaatimukset.

Tämän tutkimuksen vaatimusten käsittelyvaiheessa kirjattiin oleellisimpina asioina menestystekijöitä sekä toimintavaatimuksia. Menestystekijöitä kirjattiin seuraavalla tavalla:

1. Tuotteen tarkistusprosessin tulisi nopeuta.
2. Tuotteen tarkistusprosessin tulisi helpottua.
3. Validointityökalun tulisi toimia mahdollisimman paljon osana olemassa olevia prosesseja ja järjestelmiä.

Tavoiteltavia toiminnallisuuksia listattiin seuraavalla tavalla:

1. Ohjelma varastoi suunnittelutietämystä ehtoina ja sääntöinä.
2. Ohjelma antaa suunnittelijoille raportin tarkistettavasta rakenteesta ehtojen ja sääntöjen perusteella.
3. Ohjelma antaa mahdollisuuden suunnittelijoille lisätä ehtoja ja sääntöjä tietämysvarastoon.

4. Ohjelma antaa suunnittelijoille mahdollisuuden tulostaa raportti.
5. Ohjelma antaa suunnittelijoille mahdollisuuden oman suunnittelutyön seurantaan. Tämä tarkoittaa esimerkiksi, että validointityökalu sisältää listan johon suunnittelija voi merkitä tarkistetut osat ja nimikkeet.
6. Ohjelma antaa mahdollisuuden verrata referenssitöiden rakenteita suunniteltavaan rakenteeseen.

Vaatimukset pyritään priorisoimaan niin, että tärkeimmät vaatimukset tulevat varmasti toteutetuksi. Vaatimuksia täsmennetään suunnittelijoiden kanssa projektin edetessä. Projekti etenee tuottamalla prototyyppi ja kehittämällä sitä. Prototyyppiä pyritään kehittämään lisäämällä toiminnallisuuksia ja esittelemällä nämä lisätyt toiminnallisuudet suunnittelijoille. Edellisen listan toiminnallisuudet on kirjattu prioriteettijärjestykseen.

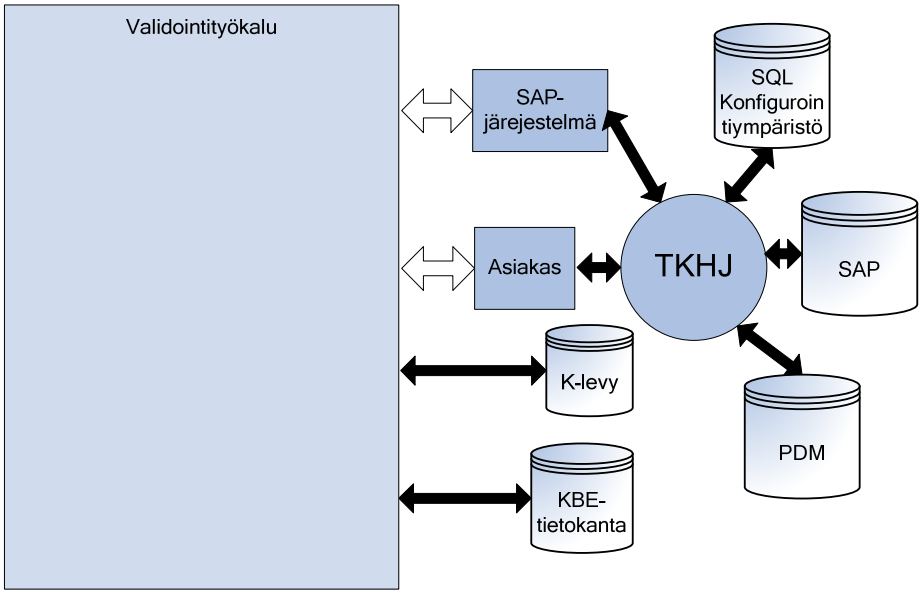
Lisäksi tärkeäksi asiaksi määriteltiin ohjelman ympäristö ja riippuvuussuhteet, erityisesti tietokantojen osalta. Tietokantojen määrittely, niiden tekniset vaatimukset sekä käyttöoikeudet on määritelty seuraavassa kohdassa.

4.2 Tietokannat

Tietokannat vaikuttavat oleellisesti validointityökalun suunnitteluun. Validointityökalun suunnittelussa on otettava huomioon tietokantojen käyttömahdollisuudet teknisestä näkökulmasta sekä käyttöoikeuksien kannalta. Kuviossa 8 on esitelty validointityökaluun liittyvät tietokannat. Tietokantojen rajapinnat on selvitetty osana vaatimusten käsittelyprosessia. Kuvion sisältö selitetään seuraavaksi:

- *K-levy* on verkkoasema joka ylläpitää normaaleja Windowsissa käytettäviä tiedostohakemistoja. Tiedostohakemistot ylläpitävät tietoa suunnittelun piirustuksista. Eri järjestelmät kuten DG (Drive glove), Teamcenter ja SAP käyttävät hakemistoa linkkien avulla. Käytännössä tämä tarkoittaa, että järjestelmät ylläpitävät linkkiä ja kun piirustus halutaan nähtäväksi, se aukeaa piirto- tai kuvaohjelmalla suoraan K-levyltä.
- *TKHJ*:llä tarkoitetaan Microsoft SQL Server 2005 relaatiotietokannan hallintajärjestelmää (RTKHJ tai relaatio-TKHJ). Se mahdollistaa muun muassa XML-tuen ja sovelluspalvelimena toimimisen. Lisäksi palvelinta voidaan käsitellä useiden työkalujen kautta, kuten esimerkiksi Visual Basicin, Management Studio sekä Configuration Manager (Brown, 2006). MS SQL Server 2005 toimii SQL konfigurointiympäristön, SAPin sekä Teamcenterin tietokantana.
- *Asiakas* havainnollistaa käyttöoikeuksia MS SQL Serverille. Koska MS SQL Serveristä käytetään useita rinnakkaisasennuksia (instansseja), niin käyttöoikeudet on määriteltävä kolmijakoisesti. Ensin on määriteltävä käyttäjälle tai käyttäjäryhmälle haluttu instanssioikeus. Instanssioikeudella tarkoitetaan fyysisen tai virtuaalisen palvelinlaitteiston käyttöoikeutta.

Tämän jälkeen määritellään käyttäjille tai tietokantaa käyttävälle ohjelmalle sen tarvitsema luku- ja/tai kirjoitusoikeus. Viimeisenä täsmennetään luku- ja kirjoitusoikeuksia skeemojen avulla (Schema). Skeemalla tarkoitetaan tässä käyttäjäkohtaista listaa, jonka avulla voidaan määrittää tietylle käyttäjäryhmälle tietyt sallitut objektit, kuten taulut ja proseduurit.



KUVIO 8 Validointityökalun tietokantayhteydet.

- *PDM-tietokannalla* tarkoitetaan ABB Oy Moottorit ja Generaattorit –yksikössä Teamcenter 2007 tuotetiedon hallintajärjestelmää ja siihen liittyvää tietokantaa. Teamcenterin tietokanta sisältää varsinaisen suunnittelutiedon eli piirustukset, osat, nimikkeet sekä osaluettelot. Teamcenterin arkkitehtuuri koostuu kahdesta kerroksesta. Ylimmällä tasolla toimiva asiakaskerros sisältää sovellukset, joiden avulla käyttäjä käyttää Teamcenteriä. Tämän kerroksen ohjelmointikielenä on käytetty Javaa. Alemmalla tasolla on palvelinkerros, jonka ydinmoduulina toimii POM (Persistent Object Manager). POM-moduuli toimii korkeammalla abstraktiotasolla kuin tietokannan hallintajärjestelmä (TKHJ). POM-moduulin avulla yhdistetään tietokannan hallintajärjestelmä sekä sovellukset, jotka käyttävät dataa. (UGS, 18) Tiedon haku tietokannasta voidaan toteuttaa POMin kautta tai vaihtoehtoisesti suoraan SQL-lauseilla TKHJ:n kautta.
- *SAP-tietokanta* ylläpitää toiminnanohjaukseen liittyvää tietoa. Tietokantaa ei voida lukea suoraan SQL-lauseilla, vaan sitä on mahdollista käyttää vain SAP-käyttöliittymän kautta. SAP sisältää monia työkaluja tiedon tarkasteluun ja hakuun tietokannasta. Ulkopuoliset sovellukset voivat käyttää käyttöliittymän toimintoja BAPI (Business Application Interface) ActiveX-komponenttien avulla. BAPIlla tarkoitetaan liiketoimintaobjekteja, joilla suoritetaan tiettyjä liiketoiminnan tehtäviä. BAPI ActiveX-komponentit mahdollistavat RFC-funktioiden (Remote Function Call) käytön, joilla ulkopuoliset järjestelmät voivat suorittaa SAPin liiketoimin-

taobjekteihin liittyviä funktioita. SAPin BAPI ActiveX-komponentteja voidaan käyttää esimerkiksi Visual Basic- tai Delphi-kielien yhteydessä. (SAP, 2011.)

- KBE-tietokanta on validointityökalun oma tietokanta. Tietokannassa ylläpidetään tietoa validaatioista eli tarkistuskriteereistä ja niihin liittyvää mahdollista muuta tietoa. Parhaassa tapauksessa tietokannan tieto käsittää kaiken sähkökoneen suunnitteluun tarvittavan tietämyksen.

4.3 Mahdolliset validointityökalun toteutustavat

ABB:n ympäristö mahdollistaa kolme varteenotettavaa tapaa toteuttaa validointityökalu. Toteutus voidaan tehdä alusta loppuun omaksi itsenäiseksi sovellukseksi. Vaihtoehtoisesti voidaan käyttää olemassa olevaa CORSU-ohjelmaa ja muokata tätä ohjelmaa Tahtikoneet-yksikön käyttötarpeita vastaavaksi. Kolmantena vaihtoehtona on rakentaa validointityökalu käytössä olevan PDM-järjestelmän yhteyteen. PDM-järjestelmänä ABB:lla on Teamcenter-järjestelmä, joka voi toimia ohjelmistoalustana toteutettavalle validointityökalulle.

Taulukossa 3 on kuvattu toteutustapojen oletettuja hyötyjä ja haittoja. Taulukossa ensimmäisenä on listattu uuden itsenäisen sovelluksen hyödyt ja haitat. Etuna uudelle sovellukselle on, että se voidaan suunnitella tarkalleen Tahtikoneet-yksikön käyttötarkoitukseen. Näin ollen sovellus voidaan suunnitella tarkalleen MOKA-menetelmää mukaillen ja sen sääntöjä noudattaen. Haittapuolena tällaisella toteutustavalla on, että sen suunnittelu ja toteutus vaati paljon resursseja sekä aikaa. Lisäksi PDM-tietokannan käyttö suoraan SQL-kyselyillä voi muodostua ongelmaksi, johtuen tietokannan monimutkaisesta rakenteesta.

CORSU-ohjelman etuna voidaan olettaa olevan, että se sisältää huolellisesti suunnitellun ja valmiiksi toiminnassa olevan toimintaperiaatteen. Tämä toimintaperiaate on hyvin samantapainen, kuin Tahtikoneet-yksikössä halutaan validointityökalulta. Lisäksi validaatioiden luominen on toteutettu tässä yksinkertaisella, mutta kattavalla tavalla. Kokonaisuudessaan CORSUn toiminnallisuudet antavat hyvin kattavan pohjan neljään ensimmäiseen toiminnalliseen vaatimukseen. Ohjelman haittapuolena voidaan olettaa olevan, ettei se sellaisenaan täysin sovellu Tahtikoneet-yksikön käyttötarpeisiin. Ohjelmaa joudutaan muokkaamaan ja tätä vaikeuttaa se, ettei ohjelmasta löydy kattavaa suunnittelun dokumentaatiota. Lisäksi tässäkin toteutusvaihtoehdossa joudutaan käyttämään monimutkaista Teamcenter-tietokantaa suoraan SQL-kyselyiden avulla. Hakua helpottaa kuitenkin se, että CORSUn käyttöön on toteutettu SQL-funktio, joka hakee tarpeellista dataa Teamcenter-tietokannan tauluista. Funktiota voidaan käyttää myös Tahtikoneet-yksikön toteutuksessa.

Teamcenterin etuna voidaan olettaa olevan sen keskeisyys tehdastuotteen suunnittelussa. Suurin osa validoitavasta tiedosta kerätään tämän järjestelmän tietokannasta. Teamcenteriin on mahdollista luoda omia sovelluksia ITK:n (Integration ToolKit) avulla (UGS, 16). Tällöin haut tietokannasta voidaan toteuttaa sen POM-rajapinnan (Persistent Object Manager) kautta. Rajapinnan

käyttö helpottaa tiedon hakua monimutkaisesta tietokannasta. Teamcenter-lähtöinen toteutus tukee parhaalla tavalla vaatimusten käsittelyvaiheessa todettuja menestystekijöitä. Tämä vaihtoehto on ainoa, joka ei luo lisää käytettäviä järjestelmiä. Tällä tarkoitetaan, että validointi voidaan toteuttaa samassa ympäristössä, missä pääosin myös suunnittelu tapahtuu.

TAULUKKO 3 Toteutustapojen vertailu

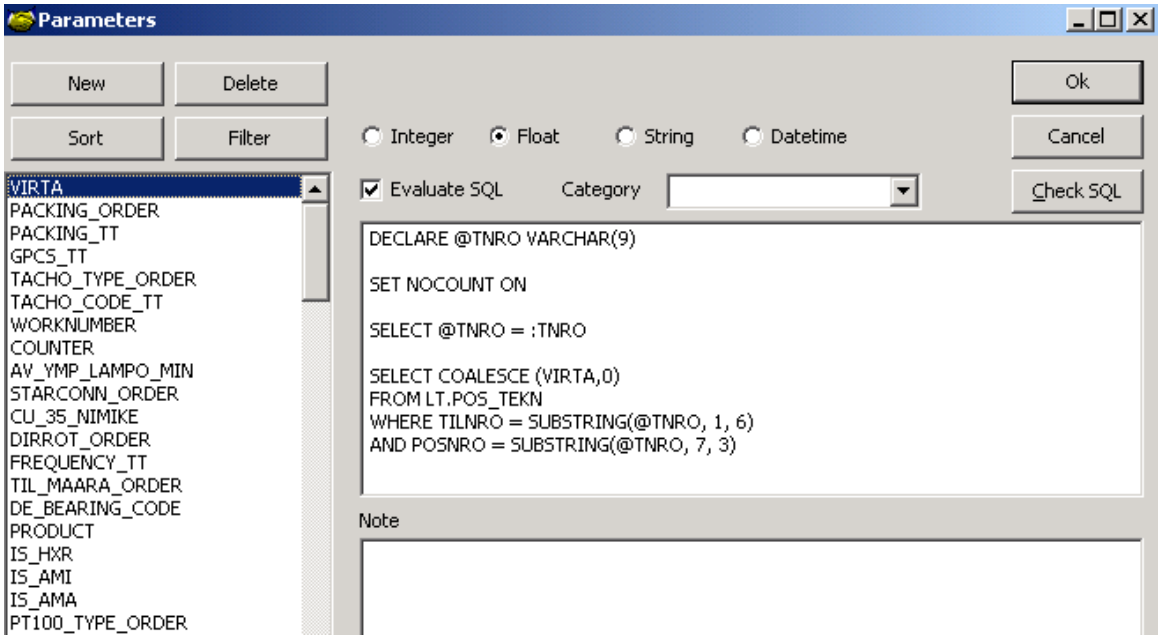
Toteutustapa	**Edut**	**Haitat**
Uusi itsenäinen sovellus	Voidaan suunnitella tarkalleen käyttötarkoitukseen sopivaksi	Työläs
		PDM-tietokannan ulkopuolinen käyttö
CORSU	Valmis ohjelma, jolla voidaan tehdä tarkistuksia osaluetteloon	Sovellusta ei ole dokumentoitu tarkasti
	Yksinkertaisuus	Ei sovellu sellaisenaan Tahtikoneet-yksikön käyttötarpeeseen
		PDM-tietokannan ulkopuolinen käyttö
Teamcenter	Suuri osa tiedosta luetaan PDM-tietokannasta	Työläs ja haastava
	Validointi tapahtuu samassa ympäristössä kuin suurin osa rakenteen suunnittelusta	Suorituskyky

4.4 Referenssiohjelma CORSU

ABB:llä on Induktiokoneet -yksikössä olemassa järjestelmä, joka tarkistaa eri palvelimilta tiedon yhtenäisyyden. Lisäksi järjestelmä tarkistaa osaluettelon kelvollisuuden eli validoi rakenteen.

Osarakenteen validointi tapahtuu ohjelmaan syötettyjen ennakkotietojen avulla. Näiden tietojen avulla ohjelma osaa esimerkiksi tarkistaa, että suunniteltu rakenne sisältää tiettyjen ehtojen vallitessa tietyn osan. Tarkistuskriteerejä voidaan syöttää ohjelmaan sen oman käyttöliittymän kautta. Jokaiseen kriteeriin liittyy jokin parametri, joka ylläpitää tietoa tietyn tietokannan taulun yksittäisestä arvosta. Näitä parametreja käytetään esimerkiksi muodostamalla kriteeriksi ehto, joka tarkistaa täyttääkö parametri annetun ennakkoehdon.

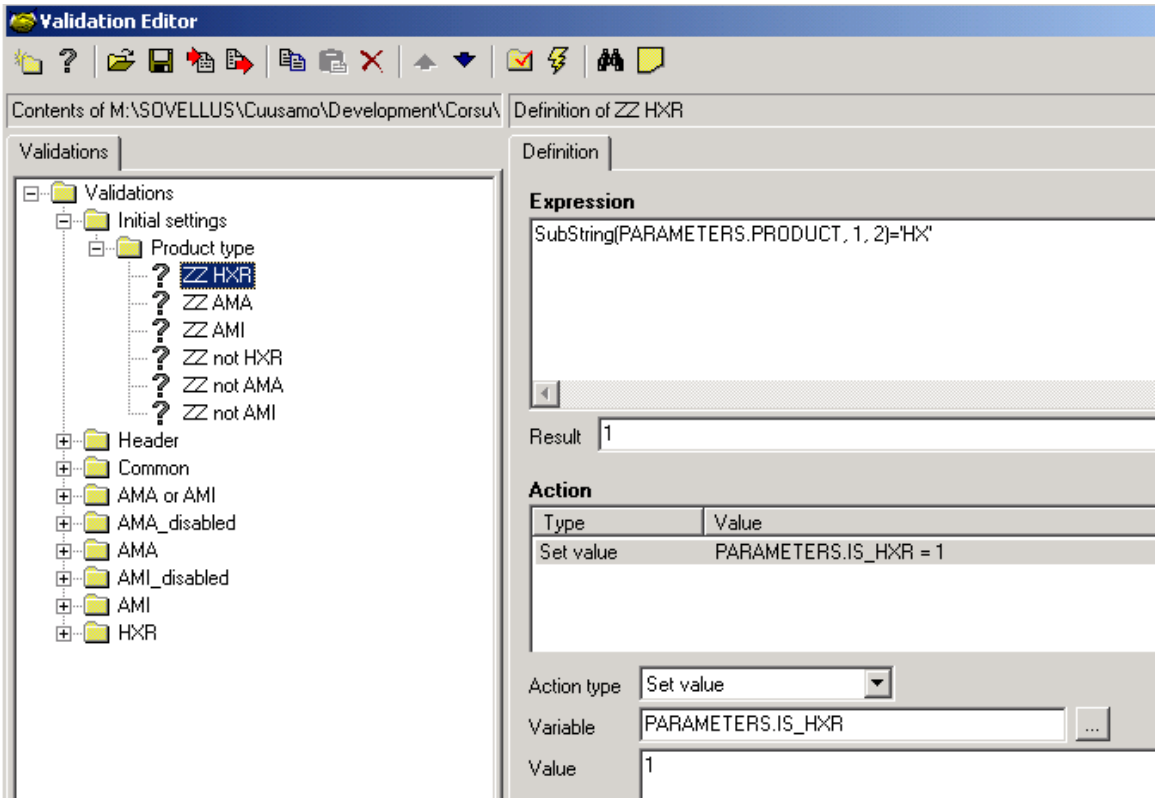
Parametreja luodaan SQL-lauseilla, CORSUn käyttöliittymän kautta, Parameters-lomakkeella. Kuviossa 9 on esitetty Parameters-lomake. Parametri luodaan ja nimetään vasemman yläkulman New-painikkeesta. Tämän jälkeen parametrille määritellään tyyppi ja kirjoitetaan SQL-lause. Luodut parametrit ilmaantuvat vasemman reunan valikkoon. SQL-lauseita voidaan luoda tietokantojen määritysten mukaisesti. Lisäksi parametreja voidaan käyttää toistensa lapsiparametreina. Tällä tarkoitetaan, että SQL-lauseen kriteerinä voidaan käyttää toista parametria kirjoittamalla parametrin nimen eteen ”:”-merkki.



KUVIO 9 CORSU-parametrien ylläpitokaavake.

Tarkistuskriteerit luodaan CORSUn kuvion 10 esittämällä Validation editor-lomakkeella. Validaatiot voidaan ryhmitellä kansioiden avulla ja kansioita luodaan vasemman yläkulman Kansio-painikkeella. Itse validaatiot luodaan viereisestä kysymysmerkistä. Validaation sääntö kirjoitetaan Expression-kenttään. Tässä CORSU saadaan tunnistamaan parametrit kirjoittamalla ensin ”PARAMETERS.” ja tämän perään halutun parametrin nimi. Näin voidaan toteuttaa ehto, ja jos ehto täyttyy, voidaan CORSU määrätä suorittamaan virheviestin kirjoittaminen tulokseen. Vaihtoehtoisesti CORSU voidaan käskeä asettamaan johonkin parametriin jokin tietty arvo. Kuvion esimerkissä asetetaan arvo ”1” parametriin ”IS_HXR”, jos annettu ehto täyttyy. Haluttu toiminto näkyy Action-kentässä.

CORSUun voidaan lisäksi lisätä lisätietoja yläpalkin oikean reunan lomakekuvakkeesta. Kuvakkeesta painamalla aukeaa uusi lomake, johon voidaan vapaasti syöttää lisätietoja.



KUVIO 10 CORSU-validaatioiden hallinta.

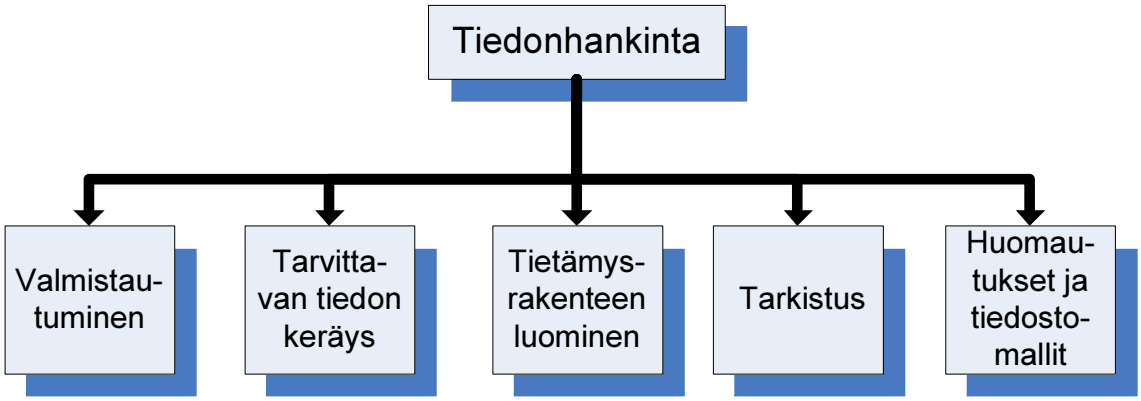
4.5 Tietämyksen hankinta MOKA-menetelmällä lyhyesti

MOKA-menetelmän elinkaari kuvattiin kohdassa 3.3.1. Tutkimusongelman mukaisesti ensimmäinen oleellinen vaihe on tiedonhankinta. Tutkimuksessa ei oteta kantaa tietämyksen keräykseen, mutta MOKA-menetelmän mukainen tietämyksen hankinta sisältää tietämyksen keräyksen ja rakenteellistammisen. Tiedon rakentaminen liittyy oleellisesti validaatioiden luomiseen ja tätä kautta koko ohjelman toiminnallisuuteen. MOKA-menetelmä kokonaisuudessaan esitellään Stokesin teoksessa *"Managing Engineering Knowledge"* (2001).

Tiedon hankinta käsittää raakatiedon keräyksen ja sen muuttamisen vapaamuotoiseksi malliksi. Kuviossa 11 kuvataan vaiheen alirakennetta, missä se koostuu eri vaiheista seuraavalla tavalla (Stokes, 2001, 72-174.):

- Valmistautumisvaiheen aikana perehdytään tietämyslähteiden, esimerkiksi eksperttien, dokumenttien ja tietokantojen, saatavuuteen ja käyttäytymiseen. Lopputuloksena tulisi olla suunnitelma keskusteluista sekä dokumenttien ja tietokantojen käsittelystä. Esimerkiksi tietokantojen ja dokumenttien käsittelyä suunniteltaessa tulisi ottaa huomioon niihin liittyvät turvallisuus- ja tekniset vaatimukset.
- Tietämyksenkeräysvaiheen aikana kerätään raakatieto. Keräystapa riippuu kerättävästä tietämystyypistä; eksperteillä, dokumenteilla ja tietokannoilla on omat menetelmänsä. Tietämyksen keräys on iteratii-

vinen prosessi, missä iteraatiot saattavat sisältää myös muita MOKA-elinkaaren vaiheita kuten formalisoinnin.
- Tietämysrakenteen luomisen aikana tiedolle annetaan standardimuoto, joka on eksperttien ja ohjelman suunnittelijoiden ymmärrettävissä. Rakenteen avulla helpotetaan tietämyksen uudelleenkäyttöä ja ylläpitoa. Rakenteisuus voidaan kuvata käyttämällä MOKAn tarjoamia ICARE-lomakkeita (Illustration Constraint Activity Rule Entity).
- Tarkistusvaiheen tarkoituksena on tarkistaa rakenteen kohdat, jotka ovat jääneet epäselviksi. Kaikki epäselvät tapaukset tulisi käydä läpi tehdastuote-eksperttien kanssa. Tarkistukseen kuuluu roolien, rakenteisuuden, tarpeellisuuden, oikeellisuuden ja ristiriitaisuuksien tarkistus.
- Huomautukset ja tiedostomallit -vaiheen aikana siirretään tuotetut mallit tietämyskantaan.



KUVIO 11 Tiedonhankintavaiheen alirakenne (Stokes, 2001, 72).

Koko tiedonhankintavaiheen tuloksena kerätään raakatietämys ja muunnetaan se vapaamuotoiseksi malliksi. Vapaamuotoinen malli sisältää seuraavat kolme kokonaisuutta (Stokes, 2001, 154–155.):

- Keskenään linkitetyt ICARE-lomakkeet sisältävät kukin tiettyä tietämyksen osaa. Lomakkeet sisältävät viitteitä toisiinsa ja muodostavat näin rakenteen koko tietämykselle. Viitteiden avulla lomakkeiden sisältämää tieto voidaan kuvata erilaisten taulukoiden avulla. ICAREa kuvataan tarkemmin luvussa 5.2.1.
- Terminologian määrittelyn avulla huolehditaan siitä, että mallien termistö on ymmärrettävää tehdastuote-eksperteille sekä ohjelmiston suunnittelijoille. Näin helpotetaan ylläpitoa sekä jatkokehitystä.
- Tietämyskirjan avulla ylläpidetään lokia, joka esimerkiksi varastoi tiedon siitä miten tietämysmalli on rakennettu ja mitä raakatietämystä siihen on käytetty.

4.6 Tietämysrakenteen muodostus ICARE-lomakkeiden avulla

Edellisessä kohdassa kuvattiin koko tietämyksen hankintavaiheen kulku. Tutkimuksen kannalta oleellisia asioita ovat ne jotka vaikuttavat validointityökalun toimintoihin ja käyttöön. Koska validointityökalu käyttää ja ylläpitää suuria määriä tietoa, niin oletettavasti tietämysrakenne mahdollistaa helpomman validointityökalun käytön, ylläpidon sekä kehityksen.

Tietämysrakenteen muodostus koostuu kokonaisuudessaan edellisen luvun tietämysrakenteen luonnista sekä sopivuuden tarkistuksesta. Sopivuuden tarkistus on syytä ottaa huomioon, koska rakenne ei ole valmis ennen kuin se on hyväksytty tehdastuotteen suunnittelijoiden tai vastaavien toimesta (Stokes, 2001, 155). Sopivuuden tarkistuksen rutiininomaisen luonteen takia sen täsmällisempi tarkastelu jätetään pois tästä tutkimuksesta. Sen sijaan seuraavaksi esitellään tietämyksen rakenteellistaminen ICARE-lomakkeiden avulla.

ICARE-lomakkeet ovat osa vapaamuotoista mallinnusta, missä käytetään hyväksi lomakkeita, joista jokainen sisältää tietyn tietämystyypin mukaista tietoa ja jotka ovat viittausten avulla yhteydessä toisiinsa. Tietämystyypit ovat seuraavat (Stokes, 2001, 100):

- Kuvaukset (Illustrations): ylläpitää tietoa esimerkeistä tai muusta tiedosta, mitä ei ole tarkoitus koodata KBE-järjestelmään.
- Rajoitteet (Constraints): ylläpitää tietoa entiteettien rajoitteista.
- Aktiviteetit (Activities): ylläpitää tietoa suunnitteluprosessin vaiheiden selityksistä.
- Säännöt (Rules): ylläpitää tietoa keinoista kuinka aktiviteetit saatetaan järjestelmään sekä muuta tietämystä suunnitteluprosessista.
- Entiteetit (Entity): ylläpitää tietoa tehdastuotteen koostavista objekteista kuten osista ja nimikkeistä.

4.6.1 Entiteetit-lomake

Liitteessä 2 on esimerkki ABB käyttötarkoitukseen sovelletusta entiteetit-lomakkeesta. MOKA-menetelmän mukaisesti jokaisen entiteetin tulisi olla oma lomakkeensa, mutta tässä tapauksessa vastaavat tiedot kaikista entiteeteistä on koottu Excel-taulukoksi. Lomake ylläpitää tietoa seuraavalla tavalla:

- *Nimi* käsittää entiteetin nimen.
- *Referenssi* ilmoittaa tunnuksen – esimerkiksi ”0000000”, jonka avulla kyseiseen entiteettiin voidaan viitata.
- *Tyyppi* kuvaa entiteetin tehtävää osaluettelossa. Se koostuu kolmesta eri vaihtoehdosta: *rakenteelliset* entiteetit kuvaavat tuoteperheen osia, alikokoonpanoja jne.; *toiminnalliset* entiteetit kuvaavat tuotevariantin tai sen alikokoonpanon toiminnallisuuksia; *käyttäytyminen* kuvaa entiteetit, jotka määrittävät tuotevariantin tai kokoonpanon käyttäytymistä, valittaessa jokin tietty rakenteellinen tai toiminnallinen entiteetti.

- *Viittaukset muihin entiteetteihin* sisältää viittaukset käyttäytymis- ja toiminnallisiin entiteetteihin sekä *äiti-* ja *lapsientiteetteihin*. Äiti- ja lapsientiteetit tarkoittavat osaluettelon mukaisia alirakennesuhteita. Eli jos osalla on alirakenne, niin sen sisältämät osat on mainittu Lapsi-kohdassa. Vastaavasti jos osa kuuluu jonkin muun osan alirakenteeseen, niin sitä ylemmällä tasolla olevan osan referenssi on mainittu kohdassa Äiti.
- *Konteksti* kohdassa kerrotaan esimerkiksi koskeeko entiteetti tiettyä tuoteperhettä vai voidaanko entiteettiä käyttää yleisesti kaikissa tuoteperheissä.
- *Muut liittyvät rajoitteet, säännöt ja kuvaukset* pitävät sisällään viittaukset muihin lomakkeisiin.

Liitteessä 2 on esimerkkejä eri entiteeteistä ja niiden sisältämästä tiedosta. Esimerkiksi osa ”0000000” eli pääkokoonpano toimii yhtenä entiteettinä. Sen äitientiteetti on osa ”TNRO” ja lapsientiteetti osa ”ZA001”. Tällä tarkoitetaan, että pääkokoonpano kuuluu osan ”TNRO” alle ja vastaavasti osa ”ZA001” pääkokoonpanon alle. Pääkokoonpanon tyyppinä on rakenteellinen, eli se toimii yhtenä alikokoonpanona osaluettelolle. Pääkokoonpanon referenssinä käytetään osan tunnusta – niin kuin muidenkin rakenteellisten entiteettien, jotta referenssi indikoisi tuoteperhekohtaista tai vakio-osaa; ei siis työnumerokohtaista. Pääkokoonpanolla on lisäksi myös viittaus käyttäytymistyypin entiteettiin ”K1”. Tässä ”K1” on referenssi entiteetille ja tämä kertoo, että osan on tultava kaikkiin koneisiin. Pääkokoonpanon kontekstikohdan ilmoitus ”Yleinen” sen sijaan kertoo, että osaa käytetään yleisesti eli entiteetti soveltuu kaikille tuoteperheille sellaisenaan.

4.6.2 Säännöt-lomake

Liitteessä 3 on esimerkki Säännöt-lomakkeesta. Se on toteutettu samalla tavoin MOKAan pohjautuen kuin Entiteetit-lomake. MOKA-menetelmän mukaisesti Säännöt-lomake sisältää samat sarakkeet kuin Entiteetit-lomake. Näiden lisäksi lomake sisältää *tavoitteet*-sarakkeen. Tässä sarakkeessa kuvataan sääntö tekstimuotoisena kuvauksena. MOKA-menetelmästä poiketen kontekstisarakkeeseen on lisätty ilmoitus siitä, tuleeko säännön rikkomisesta virhe, varoitus vai ilmoitus. Lisäksi lomakkeen loppuun on lisätty tieto siitä, mihin tietokantaan sääntö liittyy. Toisin sanoen mistä tietokannasta sääntö tulee tarkistaa.

Liitteen 3 lomakkeen toimintaa voidaan kuvata esimerkiksi säännön ”R1” mukaan. Sääntö ”R1” eli tekstien tarkistus tarkistaa löytyykö pääkokoonpano-osaan liittyvistä teksteistä merkkijonoa”VIRHE VIRHE”. Tuo merkkijono syntyy RATTI-rakennekonfiguraation suorituksen aikana, tiettyjen sääntöjen täyttyessä. Sääntö koskee kaikkia tuoteperheitä ja entiteettiä eli osaa ”0000000”. Merkkijono tulee tarkistaa Teamcenter-tietokannasta.

4.6.3 Aktiviteetit-lomake

Aktiviteetit-lomakkeen tarkoitus on ylläpitää tietoa tehdastuotteen suunnitteluprosesseista. Lomakkeet nimetään ja niihin viitataan referenssien kautta samalla tavalla kuin aiemmissa lomakkeissa. Viittausten kautta lomakkeen sisältämä tietämys voidaan liittää entiteetteihin, sääntöihin ja kuvauksiin. Aktiviteetit-lomakkeen sisältämää prosessitietämystä ylläpidetään seuraavalla tavalla (Stokes, 2001, 119—124.):

- *Laukaisinkenttä* sisältää tietoa prosessin aktivoivasta toimesta.
- *Syötteet* sisältävät tietoa entiteeteistä, joita tarvitaan prosessin suorittamiseen.
- *Tuotos* sisältää prosessin aikana tuotetut entiteetit.
- *Riskitekijät* kuvaavat prosessin epäonnistumiseen johtavia tekijöitä. Kentän sisältämä tieto mahdollistaa sääntöjen luomisen riskitekijöiden varalle.
- *Syötteiden* vaatimuksilla tarkoitetaan muita aktiviteetteja, joita tarvitaan kyseessä olevan aktiviteetin suorittamiseen.
- *Tavoitteilla* tarkoitetaan haluttuja tuloksia, joita aktiviteetin aikana tulisi syntyä.
- *Kuvaus*-kohdassa yhdistetään edellä mainittujen kenttien tieto ymmärrettäväksi kuvaukseksi aktiviteetista.

4.6.4 Rajoitteet-lomake

Rajoitteet ovat vaatimuksia, jotka tehdastuotteen on täytettävä. Ne kohdistuvat entiteetteihin tai niiden muuttujiin ja voivat koskea entiteettejä yleisesti tai vain tiettyjä entiteettejä. Rajoitteiden voidaan kuvitella olevan Säännöt-lomakkeen sisältämän tiedon kuvauksia(Stokes, 2001, 125.):

Rajoitteet-lomakkeelle viitataan samaan tapaan referenssien avulla kuin muihinkin edellä mainittuihin. Lomake koostuu seuraavista kentistä — kenttien sisältö ei poikkea aiemmin selitetyistä vastaavista kentistä (Stokes, 2001, 126.):

- nimi
- referenssi
- tavoite
- konteksti
- rajoitteen kuvaus
- liitetyt entiteetit, kuvaukset, säännöt.

4.6.5 Kuvaukset-lomake

Kuvaus-lomake sisältää samat sarakkeet ja niiden käyttöperiaatteet kuin Entiteetit-lomake, pois lukien viittaukset muihin entiteetteihin. Lomakkeelle voidaan tallentaa lisätietoa entiteeteistä, toiminnallisuuksista, säännöistä tai muuttujista.

Liitteessä 4 on esimerkki Kuvaus-lomakkeen käytöstä. Siinä esimerkiksi referenssi ”KU1” eli mekaniikkasuunnittelun piirustukset ylläpitää tietoa tietyistä piirustuksista. Tämä tieto liitetään sääntöihin ”R8” ja ”R9” sekä entiteettiin ”AA020”. Tällä tavoin edellä mainituissa lomakkeissa voidaan ylläpitää tietoa, jota ei sellaisenaan voida koodata tietämyskantaan. Liitteen 4 tapauksessa piirustusten nimet eivät ole täsmällisiä vaan suuntaa antavia.

4.6.6 ICARE-lomakkeiden käyttö

ICARE-lomakkeiden avulla voidaan standardoida tietämysvaraston sisällön koonti, jakamalla tietämys osiin. Tietämysosat liitetään viittausten avulla toisiinsa ja alkuperäiseen tietämyslähteeseen. Näin ylläpidetään ja yhdistetään systemaattisella tavalla suunnittelun kulkua. (Stokes, 2001, 101.)

Kun tietämyksen määrä kasvaa, lomakkeiden ja niiden sisältämien viittausten tulkinta voi olla sellaisenaan lähestulkoon mahdotonta. Tulkintaa voidaan helpottaa käyttämällä yhteenvetotaulukoita. MOKA tarjoaa mallin myös tällaisille taulukoille.

ABB:n ympäristössä ICAREn malli validaatioiden keräyksestä tukee esimerkiksi CORSUn kaltaisen ohjelman käyttöä. CORSUn validaatiot voidaan jakaa kansioiden avulla selkeästi yleisiin, tuoteperhe- ja tuotealustakohtaisiin validointiluokkiin. Se mihin luokkaan mikäkin validaatio kuuluu, voidaan selvittää ICAREn rakenteen avulla. Tällöin validaatioiden keräys selkeytyy ja niiden ylläpito helpottuu. Esimerkiksi validaatioita luotaessa voidaan ehkäistä tilanne, missä samoja validaatioita joudutaan luomaan uudestaan eri osien alle. Käytettäessä ICARE-lomaketta voidaan yksi validaatio liittää – referenssien ja kontekstimääritysten avulla – useille eri osille ja tätä kautta eri tuoteperheille ja tuotealustoille.

Koska validaatiot koostuvat eri tietolähteistä, eri ihmisten tietämyksestä ja ne liittyvät eri prosesseihin, on järkevää ylläpitää tietoa myös näistä tietämysalueista. ICARE-lomakkeet antavat tähän mahdollisuuden aktiviteetti-, rajoitesekä kuvauslomakkeiden avulla. Tällä tavoin ICARE samalla pakottaa miettimään validaatioiden suhdetta muuhun tietämykseen. Tällä menetelmällä validointityökalu voi samalla toimia tietämyksen yhdistävänä keskuksena.

4.7 Luvun yhteenveto sekä päätös kehitystavasta

Luvussa 4 käsiteltiin validointityökalun kehitystä tarkoituksena saattaa kehitys sellaiseen pisteeseen, jossa voidaan tehdä perusteltu päätös validointityökalun toteutustavasta.

Vaatimusten käsittelyn yhteydessä selvitettiin validointityökalun tärkeimmät menestystekijät ja toiminnallisuudet. Lisäksi tämän yhteydessä selvitettiin tietokantoihin liittyvät rajoitteet ja mahdollisuudet. Näitä tekijöitä käytettiin valittujen toteutustapojen vertailussa. Vertailussa kävi ilmi, että yhteyksien toteutusmahdollisuuksissa on eroja.

Teamcenterin etuna on, että se mahdollistaa POM-rajapinnan käytön. Tällöin validointityökalu käyttäisi Teamcenterin omaa rajapintaa tiedon hakuihin PDM-tietokannasta. Tietokannan päivitykset eivät vaikuttaisi validointityökalun toimintaan samalla tavalla, kuin jos tieto haetaan suoraan SQL-tauluista. Tiedon haku olisi myös selkeämpää POM-rajapinnan kautta, koska PDM-tietokanta muodostuu hyvin monimutkaisesta tietorakenteesta. Lisäksi yhdeksi menestystekijäksi mainittiin yhtenäisyys. Toteuttamalla validointityökalu Teamcenteriin suunnittelijoilla olisi mahdollisuus toimittaa validoinnit samassa ympäristössä, jossa he tekevät suurimman osan suunnittelusta.

CORSUn suurimpana etuna on, että se on tuotannossa toimiva ohjelma. CORSU suorittaa ABB:llä tarkastuksia, jotka vastaavat tämän tutkimuksen validointityökalulta vaadittavia tehtäviä. Lisäksi sitä voidaan muokata Tahtikoneet-yksikköön sopivaksi. Toisin kuin Teamcenter-pohjaisessa toteutuksessa, CORSU ei käytä POM-rajapintaa. Sen käyttötarkoitukseen on kuitenkin toteutettu SQL-hakufunktio, joka osaa koostaa PDM-tietokannasta kriittisen tiedon taulun. Samaa funktiota voidaan käyttää myös Tahtikoneet-yksikön tarpeisiin.

Täysin uuden sovelluksen luomisessa on etuna, että se voidaan räätälöidä täysin tahtikoneiden suunnittelijoiden tarpeisiin. Verrattuna CORSUun tämä etu tarkoittaa lähinnä käyttöliittymän suunnittelua sekä validointien muodostamista. Luvussa esiteltiin menetelmä myös validointien muodostamiseen. Esittelyn ohessa huomattiin, että menetelmää voidaan käyttää myös CORSUn kaltaisen ohjelman yhteydessä. Koska tämän tutkimuksen pääpaino on tietämyslähtöisessä suunnittelussa, niin toteutustapaa valittaessa validaatioiden muodostaminen on hyvin ratkaisevassa asemassa. Jos CORSUun voidaan yhdistää tietämyslähtöisen suunnittelun menetelmä, ei ole syytä jättää käyttämättä muita, toimiviksi todettuja CORSUn ominaisuuksia.

CORSUn käyttäminen tämän tutkimuksen lähtökohtana validointityökalun toteutukselle antaa vaihtoehdoista parhaat lähtökohdat keskittyä tietämyslähtöisen suunnittelun menetelmien käyttöön. Teamcenter-lähtöisellä toteutuksella voitaisiin saavuttaa sellaisia hyötyjä mitkä eivät muilla toteutustavoilla ole mahdollisia, mutta samalla se asettaa myös paljon haasteita. Valittaessa toteutuksen pohjaksi CORSU, saadaan samalla valmiit ratkaisut suureen osaan käyttöliittymään ja toiminnallisuuksiin liittyvistä haasteista.

5 TIETÄMYKSEN SIIRTO VALIDOINTITYÖKALUN PROTOTYYPPIIN

Tässä luvussa esitellään ensin ABB:n Tahtikoneet -yksikköön toteutettavaa prototyyppiä ja siihen toteutettavia toiminnallisuuksia. Tämän jälkeen tutkitaan, kuinka kerätty suunnittelutietämys voidaan paketoida prototyyppiin, MOKAn mukaisten formaalien mallien avustuksella.

5.1 CORSU-prototyyppi tuoteperheelle

Tähän tutkimukseen kehitetään validointityökalun prototyyppi CORSU-ohjelman pohjalta. CORSU kattaa tärkeimmän validointityökalulta vaaditun toiminnallisuuden. CORSUn avulla voidaan luoda validaatioita, joiden avulla tarkistetaan osaluettelosta halutut tiedot. Prototyypin laajuus kattaa osaluetteloon liittyvät validaatiot, yhden tuoteperheen osalta.

Tutkimuksen aikana validaatioihin liittyvät toiminnallisuudet ovat tarkentuneet iteratiivisten kehitysjaksojen aikana. Näiden aikana on käynyt ilmi, ettei CORSU sellaisenaan kykene toteuttamaan kaikkia vaadittuja validaatiota. Muutoksena joudutaan kehittämään tarkistus, joka tarkistaa rakenteen oikeellisuuden PDM-tietokannasta. Lisäksi tarvitaan tietokantojen yhtäpitävyyden tarkistus. PDM-tietokannassa esiintyvät osat, nimikkeet, piirustukset tulee löytyä myös muista tietokannoista.

Tiedon haussa SAP-yhteys vaatii aiemmasta poikkeavaa käsittelyä. Siksi prototyyppiin on luotava komponentti, joka osaa hakea tietoa SAPista. Muut muutetut toiminnallisuudet liittyvät prototyypin käytettävyyteen, esimerkiksi tarkistustuloksen tallentaminen sekä aiemman tarkistuksen lataaminen uudelleentarkistusta varten.

5.1.1 Osaluettelon rakenteen tarkistus

CORSUn alkuperäisen toiminnallisuuden avulla rakenteen tarkistusta on hyvin vaikea toteuttaa ja osittain myös mahdotonta. Tämä johtuu alkuperäisestä toiminnallisuudesta, missä jokaiselle rakenteesta tarkistettavalle osalle on määriteltävä oma parametrinsa. Tästä johtuen esimerkiksi useiden eri tuoteperheiden mukaiset rakennetarkistukset ovat hyvin työläitä. Jos tarkistettavat rakenteet muuttuvat usein, ylläpidolle koituva kuorma kasvaa hyvin suureksi.

Prototyyppiin kehitetään toiminnallisuus, joka lukee CORSUn hakemistosta tekstitiedoston. Tiedosto sisältää tarkistettavan rakenteen. Rakenne muodostetaan tiedostoon kahtena sarakkeena. Ensimmäinen sarake sisältää tiedon tarkistettavan osan tasosta ja toinen sarake sisältää osan tunnuksen. Osan taso kertoo sen paikan osaluettelossa. 1 on korkeimmalla tasolla ja 8 alimpana hierarkiassa. Jos osan taso on esimerkiksi 2, niin se kuuluu tason 1 alle. Teamcenter-tietokannassa kullakin osalla on tieto omasta tasostaan ja alaosista. Rakennetieto on siis luettavissa vain peräkkäisinä tasoina. CORSU lukee tiedostoa riveittäin ylhäältä alaspäin ja vertaa rivejä toisiinsa seuraavalla tavalla:

- Jos ensimmäisen rivin taso on sama kuin toisen, niin tarkistetaan löytyykö ensimmäisen rivin osa Teamcenter-tietokannasta. Tämän jälkeen siirrytään yhtä riviä alemmas.
- Kun rivien tasonumerot kasvavat lineaarisesti yksi taso kerrallaan, luetaan rivit muistiin kunnes seuraavan rivin tasonumero ei kasva yhdellä. Riviä, johon luku pysähtyy, ei lueta muistiin. Lopuksi tarkistetaan muistin rivit ja siirrytään yhtä riviä alemmas eli riville, jota ei luettu muistiin.
- Jos toisen luetun rivin taso on ainakin kahta tasoa suurempi kuin ensimmäisen, niin tarkistetaan vain ensimmäisen luetun rivin osa ja siirrytään yhtä riviä alemmas. Tällaisessa tapauksessa rivien väliltä puuttuu tietoa muista tasoista. Teamcenterissä tieto tallennetaan lineaarisesti yksi taso kerrallaan. Jokaisella osalla on yksi yhtä tasoa pienempi äitiosa. Osilla voi kuitenkin olla useampia lapsiosia. Näin ollen CORSU ei voi tietää, mitkä osat kuuluvat tarkistettavien tasojen väliin. Toisin sanoen, CORSU ei tässä tapauksessa voi tietää, mille osalle toisena tarkistettu rivi kuuluu.
- Jos ensimmäisen tarkistettavan rivin taso on suurempi kuin toisen, niin luetaan aiempia rivejä ylöspäin kunnes löytyy taso, joka on yhtä pienempi. Luettava rivi ei voi kuulua listan seuraavien osien alle. Näin ollen ensimmäisenä luettava rivi kuuluu sen edellisen osan alle, joka on yhtä tasoa pienempi.
- Jos toinen luettu rivi on tyhjä, tarkistetaan ensimmäinen luettu rivi ja lopetetaan tarkistus.

Jos tarkistuksen aikana on tapahtunut virhe, prototyyppi kirjoittaa tuloksen raporttiin. Tuloksessa mainitaan viimeisen virheellisen osan tunnus.

Osaluettelon rakennetarkistuksen lisäksi tarvitaan tarkistus eri osien alla oleville piirustuksille ja nimikkeille. Tässä syöttötiedoksi annetaan vain haluttu

osa, jonka avulla tarkistetaan osan alta eri tietokannoista löytyvien nimikkeiden ja piirustusten viittaukset. Tietokannoista tulee osan alta löytyä samat viittaukset. Alkuperäisen toiminnallisuuden avulla on mahdotonta tehdä tällaista tarkistusta.

Osat, joiden alta tarkistetaan piirustukset ja nimikkeet, luetaan eri tekstitiedostosta. Tämä tekstitiedosto jaetaan samaan tapaa kahteen sarakkeeseen. Ensimmäisessä sarakkeessa ilmoitetaan osan tunnus ja toisessa tieto siitä, tarkistetaanko nimikettä vai piirustusta. Tiedostoa luetaan riveittäin alhaalta ylöspäin. Tarkistus aloitetaan Teamcenter-kannasta. Tekstitiedoston riviltä haetaan luetun osan alta löytyvät nimikkeet tai piirustukset. Haetun listan nimikkeet tarkistetaan DGn ja SAPin tietokannoista sekä K-levyltä. Jos tarkistuksen aikana on tapahtunut virhe, prototyyppi kirjoittaa tuloksen raporttiin. Tällöin tuloksena mainitaan viimeinen osa, jonka alta löytyneet piirustukset tai nimikkeet eivät ole löytyneet kaikista tietokannoista. Lisäksi raportissa mainitaan puuttuva nimike ja tietokanta.

5.1.2 Tietokantayhteyksien toteutus

Pääosin tietokantahakujen toteutus onnistuu SQL-lauseita hyväksikäyttäen. SQL-lauseet voidaan luoda suoraan prototyyppiin sen käyttöliittymästä aivan kuten alkuperäisessä CORSUssakin. Hakutoiminnot toimivat Tahtikoneet-yksikössä, mikäli prototyyppiin on määritelty oikeat tietokannat sekä oikeudet näihin. Tietokantojen määritys prototyypissä tapahtuu konfigurointitiedoston avulla.

Poikkeuksen tietokantahakujen toteutukseen tekee SAP-järjestelmä (ks. kohta 4.2). SAPin tapauksessa hakuja ei voida tehdä suoraan tietokantaan. Haut on toteutettava SAP-käyttöliittymän kautta ja tarkemmin RFC-funktioita hyväksikäyttäen. Funktioiden käyttö vaatii asennetun SAP-järjestelmän koneelle. SAP-asennuksen mukana tulevat *SAP Logon Control* sekä *SAP Remote Function Call Control* –komponentit mahdollistavat RFC-funktioiden käytön.

Yksinkertaisimmillaan SAP-yhteyden luonti Delphillä tapahtuu SAP Logon Control –komponentin avulla. Yhteys SAPiin toteutetaan luomalla tämän komponentin sisältämän TSAPLogonControl–luokan olio ja kutsumalla tälle oliolle newConnection funktiota. Esimerkki yhteyden luonnista näyttää Delphikielellä seuraavalta:

```
function OpenSapConnection: variant;
begin
   FSAPLogon := TSAPLogonControl.Create(self);
   Connection := FSAPLogon.newConnection;
   if not Connection.LogOn(0, true) then
    ShowMessage('SAPiin kirjautumisessa tapahtui virhe')
   else begin
    Result := Connection;
   end;
end;
```

Esimerkissä luodaan yhteys ”Connection”-nimiseen muuttujaan. Varsinainen kirjautuminen SAPiin tapahtuu LogOn–funktion avulla. Tämä palauttaa boolean-arvon True, jos kirjautuminen onnistui. Suoritettaessa LogOn–funktiota TSAPLogonControl–luokka kysyy käyttäjätunnusta ja salasanaa.

Tietoja SAPista voidaan pyytää *SAP Remote Function Call Control* -komponentin tarjoamien toimintojen avulla. Tämä komponentti mahdollistaa SAPin omien funktioiden käytön. Yksinkertainen tapa saada tarvittavaa tietoa SAPista on käyttää SAPin RFC_READ_TABLE-funktiota. Tämän funktion avulla voidaan lukea useita eri tauluja SAPista. Funktion käyttö tapahtuu luomalla aluksi TSAPFunctions–luokan olio ja sijoittamalla edellä luotu SAP-yhteys sen Connection-ominaisuuteen. Tämän jälkeen TSAPFunctions.add-funktion avulla voidaan sijoittaa automatisointirajapinta (IDispatch) muuttujaan. Alla olevassa esimerkissä muuttuja on luotu nimellä ”Funct”. Sijoituksen jälkeen muuttujan (”Funct”) avulla voidaan käyttää RFC_READ_TABLE-funktiota. Jotta SAPista saataisiin haluttu data ulos, täytyy RFC_READ_TABLE-funktiolle määritellä myös haettava taulu, taulun sarake sekä ehdot haun rajaamiseksi. Haun suoritus tapahtuu call-funktion avulla, joka palauttaa haun onnistuessa boolean-arvon True. Seuraavassa esimerkissä on kuvattu Delphi-kielellä RFC-funktion käyttöä:

```
FSAPFunctions := TSAPFunctions.Create(self);
FSAPFunctions.Connection := OpenSapConnection;
Funct := FSAPFunctions.add('RFC_READ_TABLE');
Funct.exports('QUERY_TABLE').value:=TableName;
Funct.Tables.item('FIELDS').AppendRow.Value('FIELDNAME')
:=FieldName;
Funct.Tables.Item('OPTIONS').AppendRow.Value('TEXT') := Options;
if not Funct.call then
  raise Exception.Create(Funct.exception)
```

Esimerkin mukaisen toteutuksen avulla RFC-funktion vaatimat tiedot voidaan syöttää CORSUn käyttöliittymän Parameters-ikkunassa. Tiedoista jokainen syötetään omalle rivilleen, eikä tyhjiä rivejä sallita. Tiedot syötetään allekkain järjestyksessä: taulu, sarake ja rajoite. Lisäksi rajoitteita määritettäessä on käytettävä merkkioperaattoreita. Tämä tarkoittaa, että esimerkiksi ”=”-operaattorin sijasta on käytettävä ”EQ”-operaattoria. Tiedot tauluista ja sarakkeista on mahdollista selvittää SAPista, mutta se vaatii SAPin käytön hallintaa sekä käyttöoikeuksia tarvittaviin SAP-transaktioihin.

5.1.3 Tallennus, lataus ja suunnittelun seuranta

Vaatimusmäärittelyn mukaan prototyypin on kyettävä tallentamaan ja lataaman raportti validaatioiden suorituksesta. Alkuperäisessä CORSUssa validaatioraportti voidaan tallentaa tekstitiedostoksi. Tiedostoa ei kuitenkaan voida ladata takaisin ohjelmaan. Tämä johtuu CORSUn alkuperäisestä toiminnallisuudesta. CORSU suorittaa aina kaikki siihen asennetut tarkistukset ja näin ol-

len latautukselle ei ole mitään tarvetta. Tekstitiedosto voidaan avata suoraan hakemistosta tai suorittaa uusi tarkistus CORSUssa.

Vaatimusmäärittelyn mukaan validointityökalulta halutaan myös toiminnallisuus, jonka avulla suunnittelijat voivat seurata työtään. CORSUn on näin ollen kyettävä tallentamaan työn seuranta jollain tapaa. Tähän liittyen CORSUun toteutetaan valinta, jonka avulla suunnittelija voi päättää, mitä validaatioita tarkistetaan. Tähän voidaan hyödyntää CORSUn alkuperäistä toiminnallisuutta, missä validaatiot voidaan jakaa kansioihin. Toteuttamalla kansioihin tallennettava tarkistusehto voidaan validaatioita tarkistaa osittain. Tarkistusehto kertoo onko kansiota tarkistettu. Näin suunnittelijat voivat tarkistaa yhden tai useamman työhönsä liittyvän osakokonaisuuden kerrallaan. Tällöin validaatioiden tarkistus mukailee suunnittelijan työtä ja ylläpitää karkeaa tietoa työn etenemisestä.

Tarkistusehto toteutetaan jokaiselle kansiolle tulevalla valitsimella. Jos valitsimeen laitetaan merkki, voidaan tarkistuksen aloitusvaiheessa valita, tarkistetaanko kyseinen kansio. Tarkistuksen aloitusvaiheessa käyttäjältä kysytään, mitkä kansioista tarkistetaan. Koska tarkistettavia kansioita voi olla tuntematon määrä, niin aloitusvaiheen tarkistusvalinnat täytyy luoda dynaamisesti. Dynaamiset kansioiden tarkistusvalinnat tallennetaan samaan tiedostoon kuin tarkistuksen tulosraporttikin. Tieto tarkistetuista ja tarkistukseen valituista kansioista kirjoitetaan raporttiin samalla tavoin kuin tarkistuksetkin.

Lataus suoritetaan suoraan tekstimuotoisesta raportista lukemalla. Prototyyppi lukee raportista, mitkä kansiot ovat tarkistukseen valittavia ja mitkä näistä on tarkistettu. Kun käyttäjä painaa latauspainiketta, prototyyppi kysyy luettavaa raporttia. Kun oikea raportti on syötetty ohjelmaan, se näyttää alkunäkymän, missä näkyvät dynaamisesti luotavat valinnat tarkistettavista kansioista. Prototyyppi valitsee automaattisesti ne valinnat, jotka on jo aikaisemmin tarkistettu.

5.1.4 Piirustusten tarkistukset

Piirustuksia halutaan tarkistaa tavalla, jossa lähtötilanteessa ei tiedetä piirustuksen identiteettinumeroa. Halutussa tapauksessa piirustuksen identiteettinumero selvitetään Teamcenterin tietokannasta osan tunnuksen avulla. Tämän jälkeen tietokannasta haetun piirustusnumeron tulee löytyä K-levyltä, joka toimii piirustusten säilytyspaikkana. Alkuperäisen CORSUn toiminnallisuudella tarkistuksen toteutus ei ole mahdollista. Tämä johtuu siitä, että alkuperäisessä CORSUssa jokaisen tarkistuksen on kohdistuttava täsmälliseen tietoon. Piirustusta tarkistettaessa olisi siis tiedettävä piirustuksen numero, jotta tarkistus voitaisiin toteuttaa.

Prototyyppiin tehdään toiminnallisuus, jonka avulla piirustusten tarkistus onnistuu osan tunnuksen avulla. Validaatioksi voidaan alkuperäiseen tapaan kirjoittaa piirustuksen numeron hakeva parametri. Parametrin eteen kirjoitetaan '@' –merkki, jolloin prototyyppi tietää tarkistuksen kohdistuvan piirustuk-

seen. Näin ollen ohjelma olettaa, että parametrin toteuttama haku palauttaa halutun piirustuksen numeron, osan tunnuksen perusteella. Kun piirustuksen numero on selvillä, ohjelma tarkistaa löytyykö numeron mukainen piirustus K-levyltä. Jos piirustus löytyy, parametri viimeiseksi arvokseen piirustuksen numeron. Jos piirustusta ei löydy, parametrin arvo on ”0”.

Toteutus vertailee vain piirustuksen numeroa. Se ei siis ota huomioon piirustuksen revisiota. Teamcenterin haku ei tällä hetkellä salli revisiokohtaista hakua.

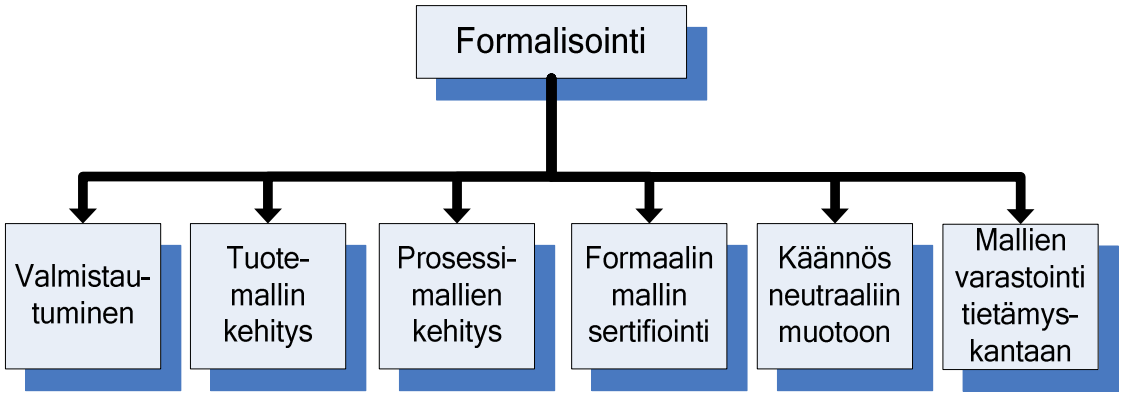
5.2 Tietämyksen formalisointi

Formalisointivaiheen tarkoituksena on esittää luvussa 4 esiteltyä vapaamuotoista mallia tarkempi kuvaus tietämysobjekteista ja niiden yhteyksistä. Tässä luvussa esitellään MOKA-menetelmän mukainen formalisointi sekä miten se liittyy kehitettävään prototyyppiin. Formalisoinnin vaatimukset otetaan huomioon kehitettäessä prototyyppiä.

MOKAn mukainen formalisointivaihe voidaan jakaa alirakenteeksi. Tällaista alirakennetta on kuvattu kuviossa 12. Askelia voidaan kuvata seuraavalla tavalla (Stokes, 2001, 187):

- *Valmistautumisvaihe* liittyy projektin hallintaan. Sen aikana katselmoidaan projektisuunnitelmaa ja verrataan siinä arvioitua työmäärää ICAREn tarjoamaan tietoon tulevan työn laajuudesta. Vaiheen aikana tarkastellaan lisäksi ICAREn pohjalta luotuja metamalleja. Metamallit voidaan kuvata MOKAn tarjoaman MML-kielen avulla.
- *Tuotemallien kehitysvaiheen* aikana luodaan Entiteetit- ja Rajoitteet-lomakkeiden avulla mallit jokaisesta tuotteesta. Mallien pohjana käytetään metamalleja, ja mallit voidaan rakentaa käyttämällä MML-kieltä. Lomakkeiden tarkoituksena on tarjota mallien muuttujille arvot. MML-mallien tarkoitus on tarjota ICAREn tarjoamaa täsmällisempi kuvaus rakenteesta ja sen komponenttien suhteesta. Vaiheen syötteenä toimii tuote tai sen osa ja tuloksena syntyy formaali malli tuotteesta.
- *Prosessimallien kehitysvaiheessa* luodaan tuotemalleja vastaava malli suunnittelun prosesseista. Nämä mallit pohjautuvat ICAREn Aktiviteetit- ja Säännöt-lomakkeisiin. Prosessimallien tarkoituksena on kuvata aktiviteettien ja sääntöjen suhteet tarkemmin. Mallien tarkoituksena on kuvata myös prosessien läpivienti, mihin voidaan käyttää UML:n aktiviteettikaaviota. Tämän askeleen syötteenä toimivat vapaamuotoiset mallit ja tuloksena syntyy formaali suunnitteluprosessin kuvaus.
- *Sertifioinnin* aikana varmistutaan siitä, että kaikkia ICARE-lomakkeita on käytetty formaaleissa kuvauksissa. Koko tietämyksen rakennusprosessi on luonteeltaan iteratiivinen. Näin ollen vaiheen tarkoituksena on myös varmistua siitä, että formaalit mallit edustavat ajan tasalla olevaa tietoa.

- MOKA suosittelee, että formaalit mallit esitettäisiin jollain neutraalilla kielellä. Tiedon tulisi olla siirrettävissä mahdollisimman helposti KBE-järjestelmään. Mallien tulisi myös olla mahdollisimman riippumattomia järjestelmästä. MOKA suosittelee mallien kääntämistä XML-muotoon. Tähän tarkoitukseen MOKA tarjoaa *MOKA Prototype Tool* -työkalun, jonka avulla mallit voidaan kääntää XML-muotoiseksi. XML:ää voidaan pitää tarpeeksi yleisenä kielenä, jonka avulla mallien sisältämää tietoa voidaan siirtää moniin järjestelmiin.
- Mallien varastointiaskeleen tarkoituksena on sitoa mallit helposti ylläpidettäväksi kokonaisuudeksi. Tietämyskannan täytyy sisältää tarvittava tieto mallin sijainnista, sen versiosta ja niin edespäin.



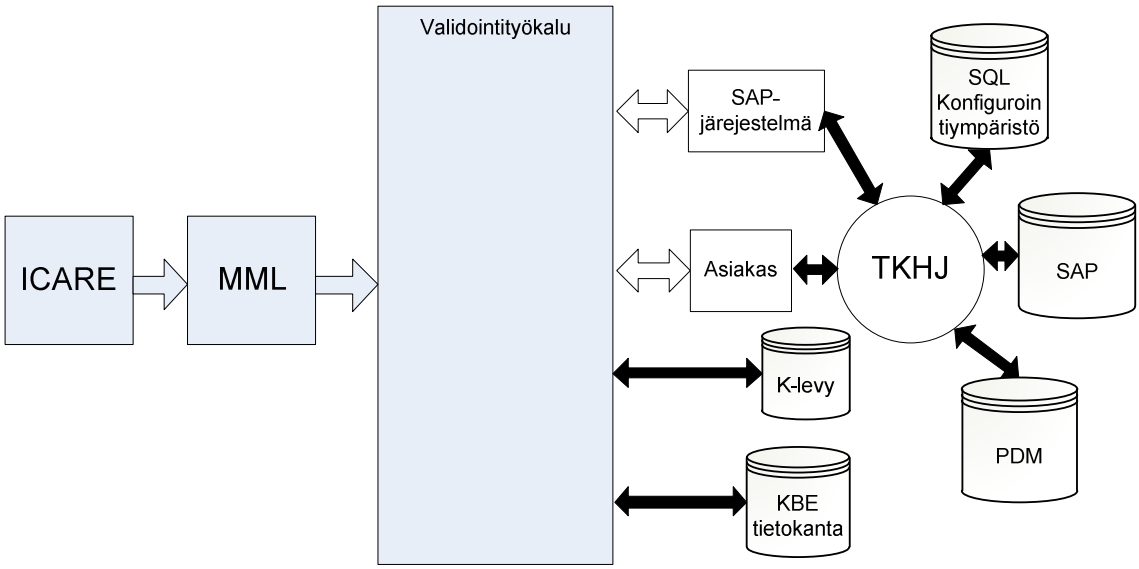
KUVIO 12 Formalisointiaskeleet (Stokes, 2001, 186).

Formalisointi liittyy oleellisesti validaatiotiedon siirtoon malleista järjestelmään. Tämän tutkimuksen tapauksessa formalisoidun tiedon päämääränä on validointityökalu. Kuviossa 13 kuvataan, miten formalisointi liittyy tämän tutkimuksen ympäristöön. Kuviossa tietämys kerätään aluksi ICARE-lomakkeisiin, ja niistä mallinnusta jatketaan formaaliin muotoon MML:n avulla.

Formaalien mallien tulee vastata CORSUn käyttöä. CORSUn tarkistukset perustuvat validaatioihin ja niiden parametreihin. Jotta mallista saataisiin tarvittava hyöty, formalisoitujen mallien on kyettävä myös parametrien mallintamiseen. CORSUn tapauksessa validaatioissa tarvittavaa tietoa ei syötetä ohjelmaan mallien mukana. Tieto haetaan aina tapauskohtaisesti eli jokaiselle osaluettelon tarkistukselle kerrallaan. Haut toteutetaan parametrien avulla, joten ne ovat oleellinen osa validaatioita. Määrittelemällä selkeä malli parametreille selkeytetään niiden käyttöä. Formaaleilla malleilla voidaan mallintaa parametrien suhdetta osiin ja validaatioihin selkeämmin kuin vapaamuotoisilla malleilla. Vapaamuotoisissa malleissa parametrien käyttö kuvataan vain viittausten avulla. Formaali mallinnus mahdollistaa selkeän kuvauksen osien, parametrien ja validaatioiden suhteista kokonaisuutena.

Parhaassa tapauksessa formalisoitu tieto voidaan siirtää suoraan järjestelmään. Tämä tarkoittaa, että tietoa käyttävä järjestelmä osaa lukea formaalia kieltä. MOKAn tapauksessa on tarjolla työkaluja, jotka kykenevät muuntamaan

MML-kielen mukaiset mallit XML-muotoon. Tietoa lukeva järjestelmä voidaan kehittää lukemaan XML-kieltä.



KUVIO 13 Tiedon siirto CORSUun.

5.2.1 Mallinnus MML-kielellä

MML pohjautuu UML-mallintamiskieleen. Lisänä UML:ään MML sisältää ennalta määrätyt näkymät, luokat sekä attribuutit. Näkymillä tarkoitetaan tuotteen metamalleja. ICAREn sisältämä tieto liitetään halutun näkymän mukaiseen metamalliin luokkien, attribuuttien ja näiden assosiaatioiden avulla. Näin muodostetaan formaali malli tuotteesta tai sen osasta. Näkymiä on viisi erilaista, joista jokainen sisältää tietyt ennalta määrätyt luokat (Stokes, 2001, 179—195):

- *Rakenteellinen näkymä* mallintaa tuotteen tai sen osan rakennetta. Rakenne koostuu osista ja niiden kokoonpanoista sekä ominaisuuksista.
- *Toiminnallisella näkymällä* mallinnetaan tuotteen toimintoja ja näiden rakennetta. Rakenne koostuu toimintotavoista ja niihin liittyvistä tekniikoista.
- *Käyttäytymisnäkymällä* tarkoitetaan tilakaavioita tuotteen valmistuksen eri vaiheista. Tilakaavio sisältää tilasiirtymien sekä rajoitteiden mallinnuksen.
- *Teknologianäkymä* sisältää materiaalinkäsittelyn sekä tuotannon prosessit eri järjestelmissä.
- *Representaationäkymä* sisältää geometrisen tiedon, mutta tähän voidaan sisällyttää myös muuta tuotteen muotoa tai kokoa koskevaa tietoa.

Näkymiä voidaan käyttää erikseen tai yhdistämällä eri näkymiä toisiinsa. Yhdistämällä saadaan kattavampi kuvaus, mutta usein näin ne myös kasvavat liian suuriksi kokonaisuuksiksi. Näin ollen tuotekuvauksia käsitellään vain harvoin yhtenä kokonaisuutena. Jokaiselle näkymällä on määrätty tietyt luokat, joiden avulla näkymiä mallinnetaan. Luokkien avulla määrätään myös niiden yhteys vapaamuotoiseen malliin. Näkymät, luokat, attribuutit ja näiden yhteydet on määritelty Stokesin ”Managing Engineering Knowledge” –oppaassa (2001).

Esimerkkinä voidaan luoda rakenteellinen näkymä ABB:n validaatioista. Kuviossa 14 havainnollistetaan rakenteellista näkymää. MOKAn mukaan rakenteellinen näkymä sisältää luokat

- tuote
- kokoonpano
- osa
- ominaisuus
- yhdistetyt ominaisuudet.

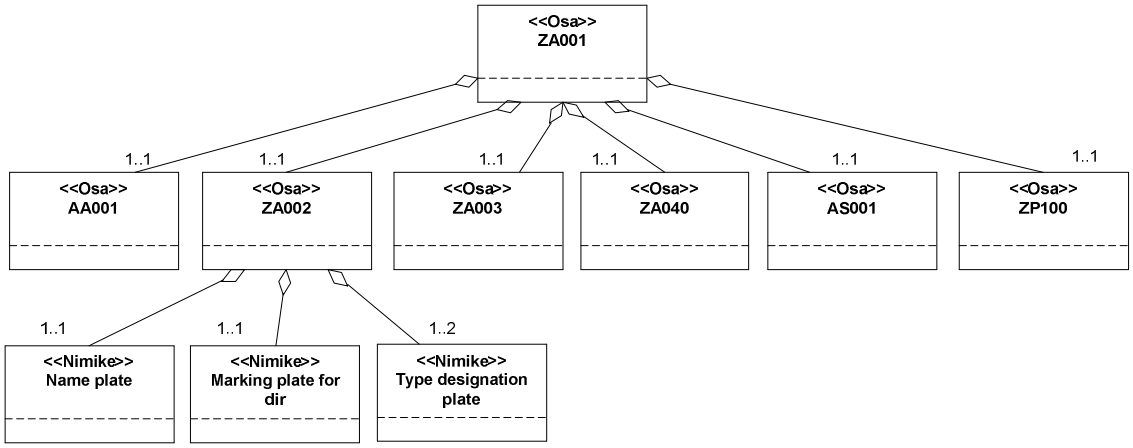
Lisäksi MOKAn mukaan rakenteellinen näkymä havainnollistaa Entiteetit-lomakkeen tarjoamaa tietoa. Kuvion 14 rakenteellinen näkymä on koostettu liitteen 2 sisältämän tiedon mukaan. Kuvio ei sisällä yhteyksiä muihin näkymiin, vaan mallintaa vain pääkokoonpano-osan rakenteen osittain. Kuviossa tuotteena on pakkauksen ja maalauksen alikokoonpanon rakenne. Johtuen ABB:n osaluettelon rakenteesta luokkien merkitykset on syytä muuttaa ABB:lle sopiviksi. MML:n mukainen osa vastaa lähinnä ABB:n nimikettä. MML:n kokoonpano taas muistuttaa ABB:n osaa (ks. kohta 2.6).

Kuviossa 14 ABB:n rakenteellinen näkymä lähtee liikkeelle ta ”ZA001”, joka koostuu alikokoonpanoista. Jokainen liitteessä 2 esiintyvä osa koostuu muista osista tai nimikkeistä. Kuvion esimerkissä on kuitenkin kuvattu vain osaa ”ZA002”. Tämä osa ilmentää liitteen 1 osaluettelon mukaista rakennetta, mutta sitäkin vain osittain. Kuviossa esimerkkiin on valittu ”ZA002” osan rakenteeksi yksi ”Name plate” –osa, yksi ”Marking plate for dir” -osa ja kaksi ”Type designation for plate”-osaa.

Kuvion 14 notaatio on MML-kielen määritysten mukaisesti peritty UML-kielestä. Näiden mukaan suorakulmiot edustavat luokkia, jotka on asetettu mallintamaan tuotteeseen liittyviä objekteja. Luokka on jaettu kahteen osaan katkoviivalla. Näistä ylemmällä tasolla kerrotaan, mikä objektityyppi on kyseessä sekä tarkempi nimitys tälle objektille. Alemmalla tasolla kuvataan luokkaan liittyviä attribuutteja. Kuviossa käytetään objektityyppeinä ABB:n osaluettelon mukaisia osia ja nimikkeitä.

Luokat on yhdistetty rakenteeksi viivoilla, joiden päässä on ”salmiakki” (vinoneliö). UML:n mukaisesti salmiakilla havainnollistetaan koostetta, joka on erityinen assosiaatio. Salmiakin puoleinen luokka kuvaa kokonaisuutta ja toisen pään puoleinen luokka kokonaisuuden osaa. Lisäksi assosiaation kumpaankin päähän voidaan liittää kardinaalisuus eli kertautumismerkintä. (Rumbaugh, Jacobson & Booch, 1999, 159.) Kuvion tapauksessa tämä ilmaisee, kuinka monta assosiaation yhdistämää alaosaa tai nimikettä liittyy kokonaisuutta kuvaavaan

osaan. Assosiaation vasemmanpuoleinen luku ilmaisee, kuinka monta luokkaa vähintään kustakin luokasta osallistuu assosiaatioon. Vastaavasti oikeanpuoleinen numero kertoo enimmäismäärän.



KUVIO 14 Esimerkki MML:n luokkakuvauksesta.

5.2.2 CORSUn validaatioiden ja parametrien mallinnus MML:n avulla

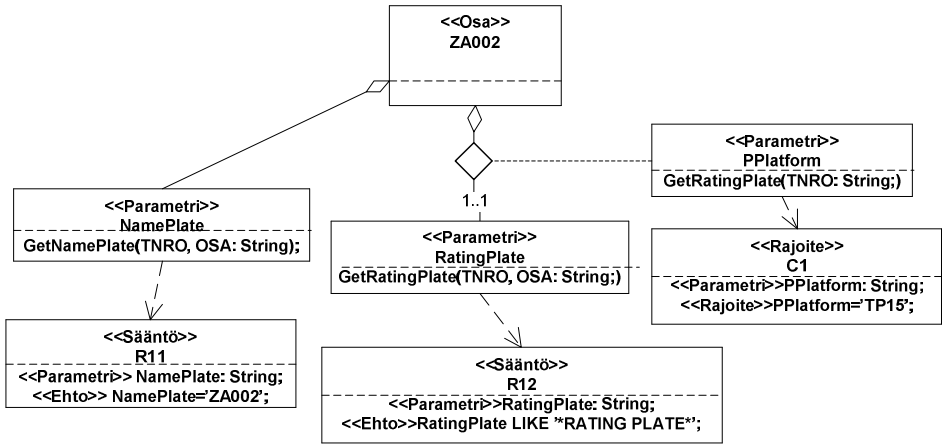
CORSUn tapauksessa tiedon formalisoinnin tulisi tarkoittaa ennen kaikkea parametreihin liittyvien hakujen sekä validaatioihin liittyvien ehtojen formalisointia. Tämä johtuu CORSUn toimintatavasta, missä tarkistukset suoritetaan parametrien ja validaatioiden avulla. CORSUn tapauksessa on ensisijaisen tärkeää mallintaa juuri parametrien ja validaatioiden suhdetta. Vain tällä tavoin voidaan ylläpitää tarvittavaa tietoa CORSUn validaatioista.

Kuviossa 15 havainnollistetaan esimerkin avulla, kuinka CORSUn käyttöön voidaan sovittaa validaatioita formalisoivaa MML-mallinnusta. Kuviossa mallinnetaan osaan ”ZA002” liittyviä validaatioita. Osa kuvaa ICAREn yhtä Entiteetit-lomakkeelta löytyvää ja samalla nimellä esiintyvää entiteettiä (Liite 2). Osaan liittyvät säännöt, rajoitteet sekä niiden yhteydet esitetään myös ICARE-lomakkeella (Liite 3). CORSUssa validaatiot rakennetaan parametreista. Validaatioita ja parametreja voidaan mallintaa luomalla näille omat luokat MML:n luokkakuvauksen mukaisesti.

Kuviossa 15 parametreille on luotu oma luokka, joka sisältää hakufunktion. Tämän funktion avulla voidaan mallintaa CORSUn parametreille määriteltäviä hakuja. Osa ”ZA002” koostuu parametriluokista ”NamePlate” sekä ”RatingPlate”, eli yhteen osaan saattaa liittyä useampi parametri. CORSUn validaatioita kuvataan sääntöluokkien avulla. Sääntöluokat sisältävät tarkistukset, jotka CORSUn tulee suorittaa. Tarkistukset tapahtuvat parametriluokilta saatujen arvojen avulla. Tässä tutkimuksessa tätä havainnollistetaan UML:n riippuvuussuhteena. UML-kielessä riippuvuus kuvataan katkoviivanuolen avulla (Rumbaugh ym., 1999, 166). Säännön sisältämää parametria käytetään säännön todentavana ehtona. Sääntö ”R11” sisältää parametrin ”NamePlate”, joka saa ar-

vonsa samannimisen parametriluokan ”GetNamePlate” -funktiolta. Funktio suorittaa haun tietokannasta, mikä tässä tapauksessa tarkoittaa hakua osatunnuksen ja työnumeron perusteella. Haku palauttaa arvon ”R11”-säännön parametrille, jota hyväksikäyttämällä sääntöluokka voi tarkistaa ehdon. Jos ehto täyttyy, sääntöluokka — tai yleisemmin CORSU — suorittaa sille määrätyn toimen, esimerkiksi kirjoittaa raporttiin virheilmoituksen.

Kuvion 15 ”R12”-sääntö muodostetaan samalla tavalla kuten edellä to ”R11”. Erona aiempaan on, että tämän ehdon muodostuksessa käytetään osan ja parametrin välissä rajoitetta. MML:n mukaisesti rajoitetta kuvataan kärjellään seisovan neliön avulla. Neliöön voidaan lisätä suora viittaus rajoiteluokkaan katkoviivan avulla ja rajoitteen tiedot voidaan nähdä luokkaan kirjoitetuista attribuuteista. Neliön avulla selkeytetään erityisesti tilanteita, joissa vaihtoehtoja on enemmän kuin kaksi. Kuvion tapauksessa rajoitteenkin muodostamisessa käytetään parametriluokkaa. Rajoite käyttää oman ehtonsa todentamiseen parametrilta saamaansa arvoa. Tätä kuvataan samaan tapaan kuin edellä riippuvuussuhteena, katkoviivanuolen avulla. Käytännössä rajoite määrittää, suorittaako CORSU kyseisen tarkistuksen. Näin rajoitteet mahdollistavat validaatioiden ryhmittelyn tuoteperhekohtaisesti. Kuvion mallissa rajoite tarkistaa, kuuluuko tarkistettava työnumero ”TP15”-tuoteperheeseen.



KUVIO 15 Esimerkki formaalista mallista CORSUn validaatioille.

Edellä kuvattu malli on vain karkea esimerkki siitä, kuinka MML-malleja voidaan sovittaa CORSUn käyttöön. Tällaisten mallien sovittaminen vaatii tarkempaa jatkotutkimusta ja tarkastelua niiden tarpeellisuudesta. Huomioitakoon kuitenkin, että MOKA menetelmän mukainen kattava rakenteen mallinnus on raskas prosessi. Formaaleja malleja voidaan käyttää kuitenkin myös pelkkään parametrien ja validaatioiden mallinnukseen. Kattavampi mallien määrittely jätetään tässä tutkimuksessa tekemättä.

5.3 Paketointi

Paketoinnilla tarkoitetaan tietämyksen muokkaamista muotoon, jossa se voidaan siirtää suoraan KBE-järjestelmään. MOKA suosittelee formalisoidun tietämyksen muuntamista XML-muotoon MOKAn määrittelemän DTD:n (Document Type Definition) avulla XML-muotoinen tieto voidaan lukea suoraan järjestelmään. Lisäksi MOKAn tarjoaman MOKA prototype tool –työkalun avulla voidaan formaalit mallit tallentaa suoraan XML-muotoon. (Stokes, 2001, 189.)

Tässä tutkimuksessa ei ole käytössä MOKA prototype tool:in kaltaista ohjelmaa, joten MML-mallien kääntäminen XML-kieleen jätetään tekemättä. Sen sijaan on syytä kiinnittää huomiota siihen, kuinka formaalit mallit voidaan tällä hetkellä paketoida CORSU-ohjelmaan. Tällä tarkoitetaan validaatioiden ja parametrien siirtämistä ICARE- ja MML-malleista CORSUun käsin.

Luvussa 4.6 esiteltiin, kuinka validaatioita sekä parametreja voidaan luoda CORSUssa. Siinä luominen tapahtuu käsin. MOKA-menetelmän mukaisia malleja voidaan käyttää hyväksi myös käsin tapahtuvassa tiedon siirrossa. Tällöin validaatioiden tekijä tulkitsee MML-malleja, kirjoittaa mallien perusteella parametrien SQL-lauseet sekä muodostaa parametrien avulla validaatioita. Luvussa 5.2 todettiin, että MML-mallien sovittaminen vaatii tarkempaa tutkimusta. Tästä syystä seuraavaksi esitetään, kuinka paketointi parametreihin ja validaatiohin tapahtuu lähtökohtaisesti ICARE-lomakkeiden avulla. Esittelyssä tuodaan kuitenkin esille MML-kuvausten käytön hyödyt.

5.3.1 CORSU-parametrien paketointi

Parametrien paketoinnissa on otettava huomioon, mihin osaan ja validaatioon parametri liittyy. Osa ja siihen liittyvä validaatio voidaan täsmentää koskemaan tiettyä tuoteperhekohtaista rakennetta rajoitteiden avulla. Tätä kuvataan CORSUn osalta tarkemmin kohdassa 5.3.2. Jotta parametrien ja mallien väliset viittauksia voidaan ylläpitää, täytyy parametri liittää entiteettiin eli osaan. Kohdassa 5.2.2 käytiin läpi, kuinka tämä tapahtuu formaalien mallien avulla. Tässä kohdassa kuvataan kuinka viittauksia voidaan ylläpitää CORSUssa. Lisäksi esitellään miten mallien tietoa voidaan syöttää CORSUun.

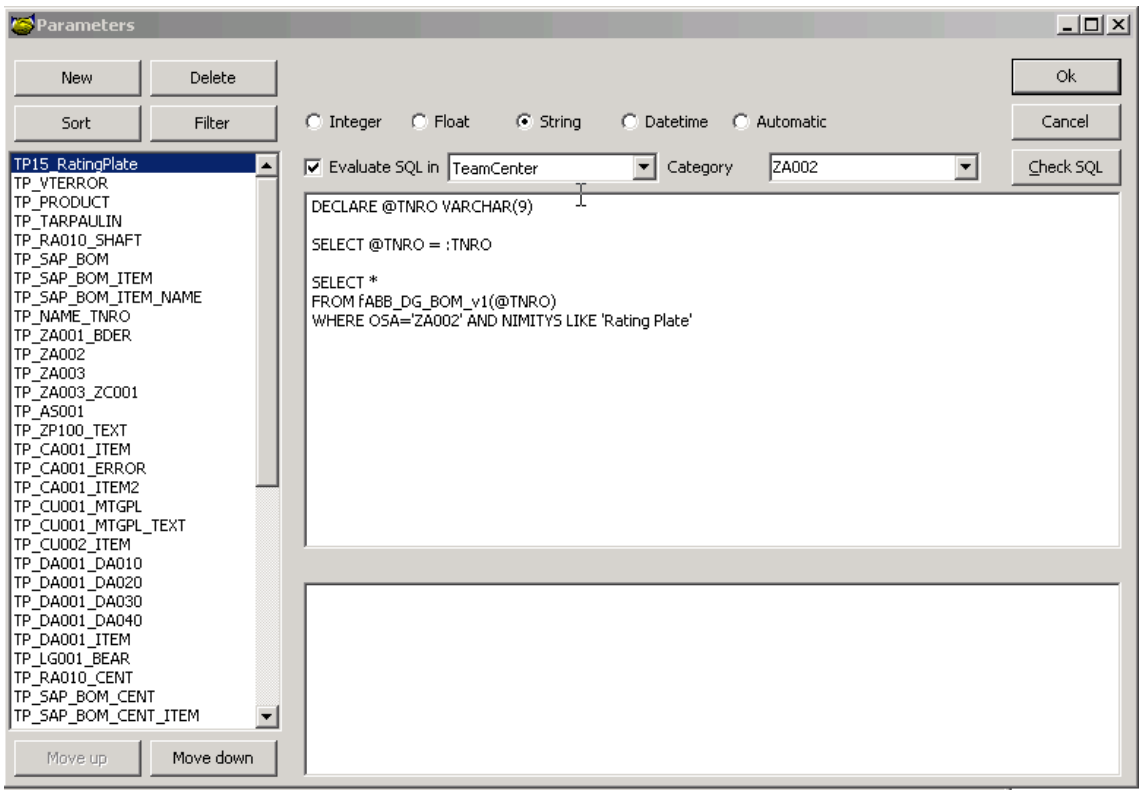
Parametrien määrä saattaa kasvaa hyvin suureksi ja niistä jokainen suorittaa haun tietokannasta. Kun parametrien käyttö on selkeää, voidaan estää turhien parametrien luonti. Turhilla parametreilla tarkoitetaan kahta tai useampaa parametria, jotka suorittavat käytännössä saman haun. Tällaisia tilanteita syntyy, koska validaatiot saattavat tarvita samaa tietoa. Ilman mallinnusta parametreja luodaan yksi validaatio kerrallaan ja tällöin saman haun sisältävät parametrit eivät erotu massasta. Turhilla parametreilla synnytetään turhaa kuormaa tietokantoihin sekä hidastetaan CORSUn toimintaa.

CORSU mahdollistaa parametrien jaon ryhmiin ”Category”-valintalistan avulla. Valintalistaan voidaan luoda vapaasti ryhmiä, joihin parametreja liite-

tään. Valintalistaa voidaan käyttää hyödyksi parametrien ja MOKA-mallien välisissä viittauksissa. Valintalistan avulla parametrit voidaan ryhmitellä sen osan mukaan, jonka yhteydessä niitä käytetään. Tällä tavoin parametriin liitetään osan viite, joka toimii suorana viittauksena ICAREn Entiteetit-lomakkeelle. Parametreja voidaan tämän jälkeen suodattaa käyttöliittymän avulla.

Rakentamalla validaatiot ICARE-lomakkeiden avulla saadaan aikaan selkeä rakenne validaatioille. ICAREssa jokainen validaatio liitetään entiteettiin eli osaan. Osat muodostavat referenssien avulla lomakkeille osaluettelon mukaisen rakenteen. Tällä tavoin samoja validaatioita voidaan helposti mallintaa useissa osissa.

Kuviossa 16 näytetään, kuinka CORSUn parametreille voidaan rakentaa viittaus ICAREn malleihin. Kuviossa osalle ”ZA002” luodaan parametri, jonka nimeksi annetaan ”TP15_RatingPlate”. Parametri lisätään ryhmään, joka nimetään osan ”ZA002” mukaan. Tällöin parametri viittaa kaikkiin Entiteetit-lomakkeen ”ZA002”-osan validaatioihin.



KUVIO 16 CORSU-parametrien paketointi.

Entiteetit-lomakkeelta voidaan tuoda myös muuta tietoa, jonka avulla parametrit voidaan ryhmitellä tarkemmin. Kuten edellä, tarkennustieto voidaan liittää parametrin nimeen. Jokaisen parametrin nimeen voidaan liittää tuoteperheen tunnus, mikäli parametrin toiminta liittyy vain tiettyyn tuoteperheeseen. Kuvion esimerkissä parametrin nimen eteen on lisätty etuliite ”TP15”, koska parametri liittyy samannimiseen tuoteperheeseen. Tämä ilmenee myös liitteen 3, säännön ”R12” kohdalta, sarakkeesta ”Konteksti”.

Sovitettujen ICARE-lomakkeiden avulla voidaan määrittää myös tietokanta, josta haku tapahtuu. Liittessä 3 kuvattujen tietokantasarakkeiden avulla voidaan nähdä, että säännölle ”R12” tarvitaan haku ”TC”-tietokannasta (Teamcenter). Kuvion 16 esimerkistä nähdään, kuinka ”Database”-kohdan alasvetovalikolla määrätään parametrille tietty tietokanta.

ICARE-lomakkeet eivät kuitenkaan sovellu suoraan parametrien erittelemiseen, koska parametreille ei ole määritelty omaa lomakettaan. Käytettäessä alkuperäisiä lomakkeita parametrit joudutaan määrittelemään jokaiselle validaatiolle erikseen. Tieto validaatioon tarvittavasta hausta eli parametrista voidaan liittää esimerkiksi Säännöt-lomakkeen kuvaussarakkeeseen. Tällä tavoin on kuitenkin hyvin vaikea esittää saman parametrin käyttöä eri validaatioiden yhteydessä. CORSUn tapaukseen olisikin järkevää luoda oma lomakkeensa tarvittaville parametreille ja liittää nämä parametrit referenssien avulla entiteetteihin ja sääntöihin. Toinen vaihtoehto on määritellä MML:n mukainen malli, kuten luvussa 5.3 esiteltiin. Tällöin parametrien erottaminen säännöistä voisi tapahtua vasta formaaleja malleja luotaessa.

5.3.2 CORSU-validaatioiden paketointi

Parametrien lisäksi on validaatioita luotaessa otettava huomioon, mihin tuoteperheeseen ja osaan validaatio liittyy. ICAREa mukaillen validaatioille voidaan määritellä osaluettelon mukainen rakenne entiteettien eli osien avulla. Vastaavasti CORSUssa validaatiot voidaan jakaa luokkiin. Näille luokille voidaan määritellä rajoitteet, jotka määräävät, tarkistetaanko luokan sisältämät validaatiot.

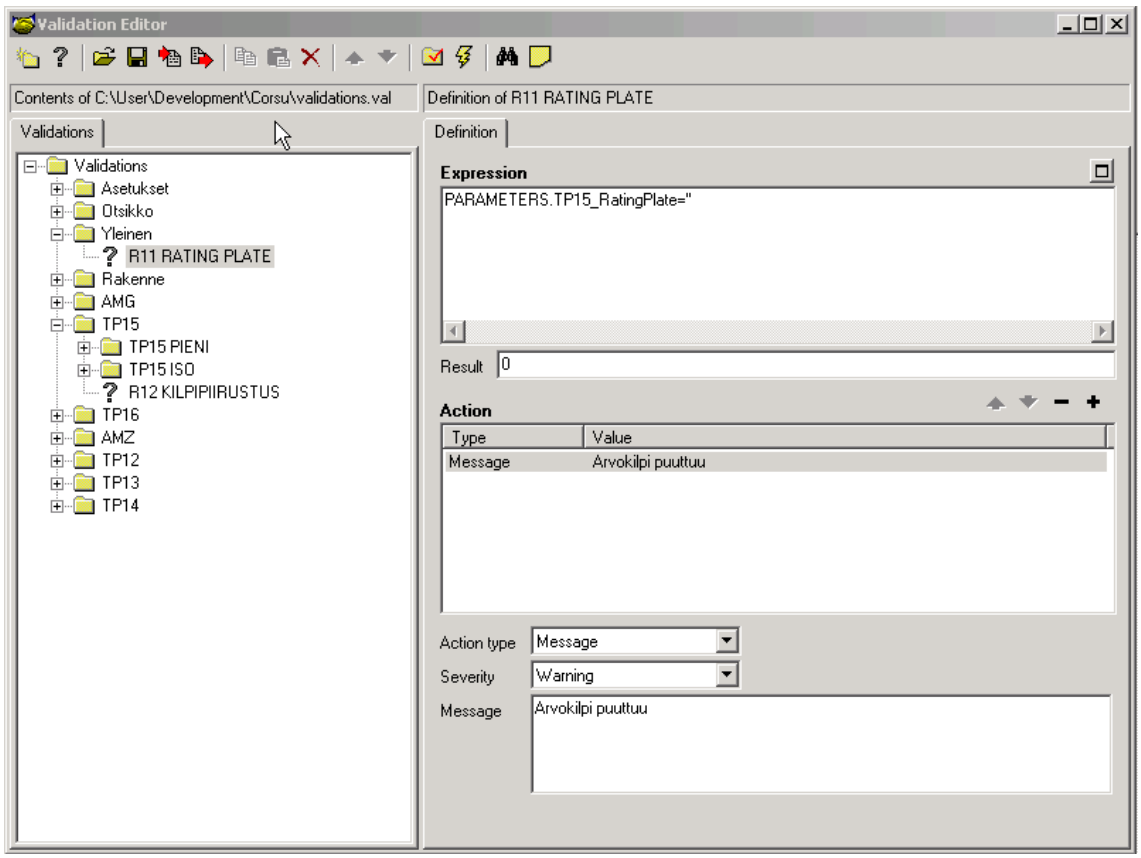
ICARE-lomakkeet määrittelevät mihin kokonaisuuteen osat ja säännöt kuuluvat. Tämä tieto on ilmoitettu lomakkeiden ”Konteksti”-sarakkeissa. Tätä tietoa hyväksi käyttäen validaatiot voidaan muodostaa tuotealusta- ja tuoteperhekohtaisesti. Kuviossa 17 validaatiot on jaettu tuotealustoihin, tuoteperheisiin ja yleiseen kansioon. Yleinen kansio sisältää validaatioita, jotka koskevat kaikkia koneita. Kuviossa lisätään osaan ”ZA002” liittyvä validaatio, joka sisältää säännöt ”R11” ja ”R12”. Liitteen 3 Säännöt-lomakkeesta voidaan huomata, että näistä ensimmäinen on yleinen sääntö ja toinen koskee ”TP15”-tuoteperhettä. Kuvion esimerkkitapauksessa sääntö ”R11” luodaan yleisen kansion alle ja ”R12” kansion ”TP15”. Koska molemmat säännöt ovat osan ”ZA002” alla, voidaan olettaa, että niihin voidaan käyttää samaa parametria. Tässä tapauksessa näin olisikin, jos CORSUn validointia määriteltäessä voitaisiin käyttää jokerimerkkejä eli merkkejä, jotka vastaavat mitä tahansa merkkijonoa. Kuviossa kuvataan kuitenkin tapaus, jossa molemmat säännöt vaativat oman parametrinsa.

Validaatioiden muodostaminen säännöille tapahtuu kuvion esimerkissä samalla tavalla kuin luvussa 4.4. Samalla tavalla muodostetaan myös kansioi-

den ehdot, jotka määräävät, tarkistaako CORSU kansion validaatioita valitulle työnumerolle.

ICAREn Kuvaukset-lomaketta sekä Aktiviteetit-lomaketta voidaan käyttää CORSUn validaatioiden täsmentämiseen. Nämä lomakkeet sisältävät viitteet sääntöihin ja entiteetteihin. Vastaavasti CORSUun voidaan syöttää validaatioita täsmentävää tietoa (ks. luku 4.4). Näin ollen näiden lomakkeiden tieto voidaan liittää täsmentävänä tietona validaatioihin.

ICAREn mukaiset mallit eivät sellaisenaan anna vastausta, mitä parametria käytetään millekin validaatiolle. Mallit antavat vain viitteen mahdolliseen parametriin. Luvun 5.3 kaltaisen sovitettujen MML:n mallien avulla ratkaistaisiin tämä ongelma. Tällaisten mallien rakentaminen pakottaa suunnittelijan pohtimaan, kuinka parametrit muodostetaan. Formaalista mallista on lisäksi helppo nähdä parametrin yhtäläisyys toiseen parametriin.



KUVIO 17 CORSU-validaatioiden paketointi.

6 LOPPUTULOS

Tämän tutkimuksen tarkoituksena oli tutkia, millä tavoin tietämyslähtöistä suunnittelua (KBE) voidaan käyttää hyväksi validointityökalun toteutuksessa. Tietämyslähtöistä suunnittelua pyrittiin käyttämään validointityökalun prototyypin toteutuksessa. Prototyypin kehitys eteni MOKA-menetelmän mukaisesti, jotta prototyypin käyttö tukisi tietämyslähtöisen suunnittelun periaatteita. Tässä luvussa esitellään tulokset siitä, miten tässä onnistuttiin, ja kehitysehdotukset asioille, jotka niitä vaativat.

6.1 Tulosten arviointi

Aiemmista tutkimuksista löydettiin muutamia menetelmiä tietämyslähtöiseen suunnitteluun. Tähän tutkimukseen valittiin MOKA-menetelmä, jonka katsottiin olevan käyttökelpoisin ABB:n tarpeeseen. Lisäksi ABB:ltä löydettiin suunniteltavaa validointityökalua vastaava sovellus, jota muokkaamalla kehitettiin prototyyppi Tahtikoneet-yksikön tarpeeseen. Seuraavaksi esitellään tutkimuksen aikana syntyneet tulokset MOKA-menetelmän soveltuvuudesta ABB:n tarpeeseen ja erityisesti CORSUun. Lisäksi esitellään tulokset prototyypin kelvollisuudesta Tahtikoneet-yksikön tarpeeseen. Aluksi on kuitenkin syytä arvioida, kuinka MOKA tukee tietämyslähtöisen suunnittelun mukaisten järjestelmien kehitystarpeita.

6.1.1 MOKA-menetelmän arviointi

Tietämyslähtöisen suunnittelun kannalta on keskeistä rakentaa tehdastuotteen tietämys kattavaksi malliksi. Tällaisten mallien avulla pyritään koostamaan hajanainen tietämys yhdeksi kokonaisuudeksi. MOKA-menetelmä antaa selkeän tavan, jolla tietämystä voidaan mallintaa ja formalisoida. Formalisoitujen mallien tieto on muutettavissa muotoon, jossa se voidaan helposti lukea eri ohjelmien avulla. Tällainen formalisoidun tiedon siirto jää kuitenkin vain suositusten

varaan. MOKA suosittelee XML-pohjaa tietoalustaksi, josta tieto voidaan siirtää KBE-järjestelmään. Menetelmän käyttöön on lisäksi saatavissa Stokesin (2001) teos ”Managing Engineering Knowledge”. Tämän teoksen selkeiden askelien ja esimerkkien avulla menetelmän käyttö on helposti omaksuttavissa ja käytettävissä.

Verrattuna tässä tutkimuksessa ilmi tulleisiin, aiempiin tietämyslähtöisen suunnittelun määritelmiin, MOKA-menetelmä antaa vain pohjan tietämyslähtöisen suunnittelun mukaisille järjestelmille. Menetelmä antaa vain tavan, millä kehityskohteen rakenteeseen liittyvä tietämys voidaan siirtää dataksi KBE-järjestelmiin. Rakenteen mallintamisella on suuri merkitys, mutta tietämyslähtöisien järjestelmien käytöllä on kuitenkin monia eri tarkoituksia. Tätä havainnollistettiin tutkimuksen kohdassa 2.4.2, missä kuvattiin, mitä eri tehtäviä KBE-järjestelmät toimittavat sekä millä tavoin KBE-järjestelmiä voidaan ryhmitellä eri kategorioihin. KBE-järjestelmien kategorisointi tapahtui tässä erityisesti niiden suorittamien tehtävien mukaan. MOKA ei esimerkiksi ota kantaa siihen, mitä suunniteltavalla järjestelmällä halutaan tehdä. MOKAa voidaan oletettavasti sovittaa monenlaisiin eri KBE-järjestelmiin, mutta se ei esimerkiksi kerro miten loogisen rakenteen validointi ja verifiointi tapahtuu tai miten ongelman ratkaisuun voitaisiin käyttää tuotesuunnittelun eri tapojen vertailua. Lisäksi MOKA-menetelmän käyttöä vaikeuttaa sen laajuus. Jos rakenteellinen tietämys muuttuu usein, MOKAn käytöstä tulee vaivalloista. Tämä johtuu siitä, että MOKA edellyttää tarkkaa tietämyksen määritystä ja tietämystä kerättäessä on aina otettava huomioon koko tuotteen rakenne.

Koska MOKA perustuu tapaan, jolla mallinnetaan kehityskohteen tietämystä kokonaisvaltaisen tuoterakenteen avulla, sen käyttö haluttuun kehityskohteeseen saattaa olla turhan kankeaa. Esimerkiksi suurten yritysten valmistamat tehdastuotteet saattavat muodostaa monimutkaisia rakenteita, joiden sisältämä tietämys on hajaantunut moniin eri tietolähteisiin.. Tällaisten rakenteiden kattava mallintaminen on hyvin haastavaa ja aikaa vievää. Tällaisissa tapauksissa MOKA-menetelmän käyttö vaatii sen sovittamista käyttötarkoitukseen sopivaksi.

6.1.2 MOKA-menetelmän soveltuvuus kehityskohteeseen

ABB:n Tahtikoneet-yksikkö valmistaa tuotteita kymmenien eri tuoteperheiden pohjalta. Tuoteperheet koostuvat sadoista eri osista, nimikkeistä sekä piirustuksista ja muodostavat näiden avulla monia eri tuotevariantteja. Jokainen tuotevariantti sisältää käytänteitä ja sääntöjä, jotka on dokumentoitu satoihin dokumentteihin. Lisäksi tehdastuotteiden suunnittelijat omaavat tuotteisiin liittyvää tietämystä, jota ei välttämättä ole dokumentoitu lainkaan. Validointityökalun tarkoituksena on tehdä yksinkertaisia tarkistuksia osaluetteloon ja ylläpitää tarkistussääntöjen avulla rakenteisiin liittyvää tietämystä. Validointityökalun prototyypin kehityksessä ei ole tarkoitus käydä kokonaisvaltaisesti läpi tuoteperheen osaluetteloa ja sen suunnitteluun liittyviä sääntöjä, aktiviteetteja ja rajoit-

teita. Tällaiseen ohjelmatarpeeseen MOKAn käyttö saattaa sellaisenaan olla liian raskasta. Tämä johtuu siitä, että MOKA edellyttää kattavaa tuoteperheen rakenteen ja sen suunnittelun määrittämistä ennen tietämystä käyttävän järjestelmän käyttöönottoa.

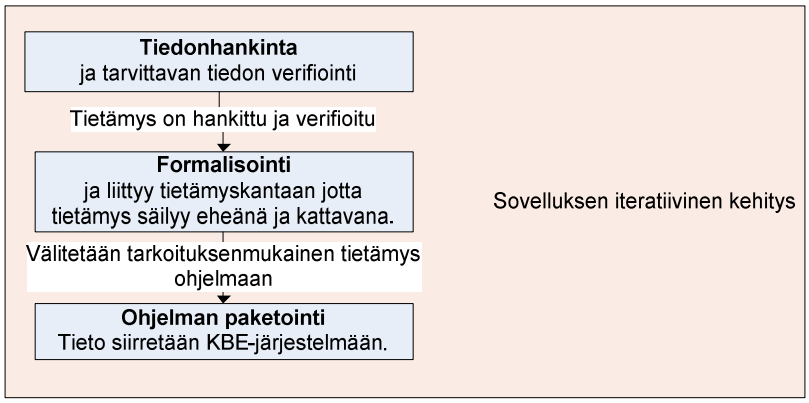
MOKA-menetelmän mukainen tietämyksen keräys pakottaa kuitenkin ohjelman suunnittelijan sekä tehdastuotteiden suunnittelijat pohtimaan, kuinka validaatioiden luominen on käytännössä mahdollista. Esimerkiksi ICAREn mukainen tietämyksen keräys yhdistää validaatiot osaluettelon osiin. Lomakkeessa myös osat viittaavat toisiinsa muodostaen osaluettelon rakenteen ICARE-lomakkeelle. Rakenteen avulla selviää ongelmia liittyen validoitavan tiedon saatavuuteen. ICARE voi epätäydellisenäkin olla avuksi validaatioiden toteutuksessa. Lisäksi rakenne mahdollistaa validaatioihin liittyvän tietämyksen varastoinnin tavalla, jonka avulla esimerkiksi CORSU-parametrien luonti on selkeämpää ja niiden ylläpito on yksinkertaisempaa. Tätä havainnollistettiin luvuissa 4 ja 5.

ABB:n tapauksesta on opittu, että validaatiot tarkentuvat sitä mukaa, kuin ymmärrys tietokannoista sekä osaluettelon muodostumisesta kasvaa. Tehdastuotteen suunnittelijalla ei ole kattavaa tietoa esimerkiksi tietokantoihin liittyvistä rajoittavista tekijöistä. Ohjelman suunnittelijalla ei ole tarvittavaa tietämystä koneen suunnittelusta ja siihen käytettävästä tietämyksestä. Tällaisessa tapauksessa ohjelman suunnittelu etenee samalla, kun validaatiot tarkentuvat ja tehdastuotteiden suunnittelijoiden ymmärrys kasvaa esimerkiksi tietokantayhteyksien rajoittavista tekijöistä. Tällaisessa suunnittelijoiden yhteistyössä MOKA-menetelmän toimii hyvänä kommunikaatiovälineenä. MOKAn mallien avulla kumpikin osapuoli saa täsmällisempää tietoa siitä, mitä ollaan tekemässä tai millä tavalla toteutus on mahdollista tehdä.

ABB:n tapaukseen voidaan soveltaa MOKA-menetelmää siten, että validointityökalun suunnittelu kohdistuu iteratiivisesti kolmeen MOKAn elinkaaren vaiheeseen. Kuviossa 18 havainnollistetaan validointityökalun suunnitteluprosessista MOKA-menetelmän Tiedonhankinta-, Formalisointi- sekä Ohjelman paketointi -vaiheita. Prototyypin toteutuksen aikana huomattiin, että väärinkäsitykset ja muut virheet, jotka liittyivät validointityökalun toteutukseen, syntyivät näiden vaiheiden aikana. Tästä syystä sovelluksen toteutusta olisi järkevää toteuttaa iteratiivisesti näiden kolmen vaiheen avulla. Yksi iteraatio voi olla esimerkiksi yksi tuoteperhe tai jopa pienempi kokonaisuus. Jokaisen iteraation aikana kehitetään prototyyppi ja varmistetaan, että validointityökalu toteuttaa tarkistukset halutulla tavalla. Lisäksi katselmoidaan, tarvitseeko toteuttaa uusia toiminnallisuuksia. Näin ollen kehitys tukee muun muassa RADin mukaista sovelluskehitystä, missä kehitetään prototyyppiä iteratiivisesti.

MOKA-menetelmän formalisointi ja paketointi eivät sellaisenaan sovi iteratiiviseen suorittamiseen, koska askeleet kattavat koko rakenteen mallinnuksen. Luvussa 5.3 kuvataan kuitenkin, miten MOKAn formaaleja malleja sovittamalla voidaan mallintaa pelkästään parametrien, osien ja validaatioiden välistä suhdetta. Tällä tavoin kuvion 18 formalisointiaskeleen ei tarvitse yhdellä kertaa sisältää koko rakenteen mallinnusta. Iteraatioiden formaalit mallit voitaisiin

paketoida luvun 5.4 mukaisella tavalla. Iteratiivisen lähestymistavan avulla voitaisiin ehkäistä järjestelmässä käytettävän tietämyksen virheellisyyttä. Muun muassa kohdassa 2.4.2 tuotiin esille, että siirrettävän tietämyksen validointi koetaan pullonkaulaksi KBE-järjestelmien kehityksessä.



KUVIO 18 ABBn tapaukseen suositeltu kehitysiteraatio MOKA-menetelmää mukaillen.

Prototyypin toteutuksen aikana huomattiin myös, että CORSU-parametrien tuntemus yksilöityy vain CORSUn ylläpitäjälle. Parametreista ei ole olemassa dokumentointia, joka selventäisi niiden käyttötarkoitusta. Jos parametrien tuntemusta voitaisiin levittää tehokkaammin, niin uusien ylläpitäjien koulutus olisi vaivattomampaa. Lisäksi tehdastuotteiden suunnittelijat voisivat osallistua validaatioiden suunnitteluun tehokkaammin, jos heillä olisi tarkempi tuntemus validaatioista ja näihin liittyvistä parametreista. Tällaiseen tarkoitukseen sopii MOKA-menetelmästä muunnettava formaali malli. Tällaista mallia kuvattiin luvussa 5.3. Mallin avulla voidaan jakaa tietämystä siitä, mitkä parametrit liittyvät mihinkin osaan ja validaatioon. Näin ollen parametrien käyttö helpottuisi esimerkiksi edellä mainituissa tilanteissa.

MOKA-menetelmä ei siis suoraan sovellu Tahtikoneet-yksikön käyttötarkoitukseen. ABB:n ympäristö, siinä esimerkiksi osaluettelon rakenne ja sen käyttö, on kuitenkin hyvin ainutlaatuinen. Tämän perusteella voidaan olettaa, ettei mikään muukaan menetelmä soveltuisi suoraan samaan käyttötarkoitukseen. Tämän tutkimuksen perusteella MOKA-menetelmää on kuitenkin mahdollisuus sovittaa tähän käyttötarkoitukseen. Sillä tavoin voidaan tulevaisuudessa saavuttaa suurempiakin hyötyjä, kuin tavoiteltiin validointityökalun tapauksessa.

6.1.3 Toteutetun prototyypin esittely

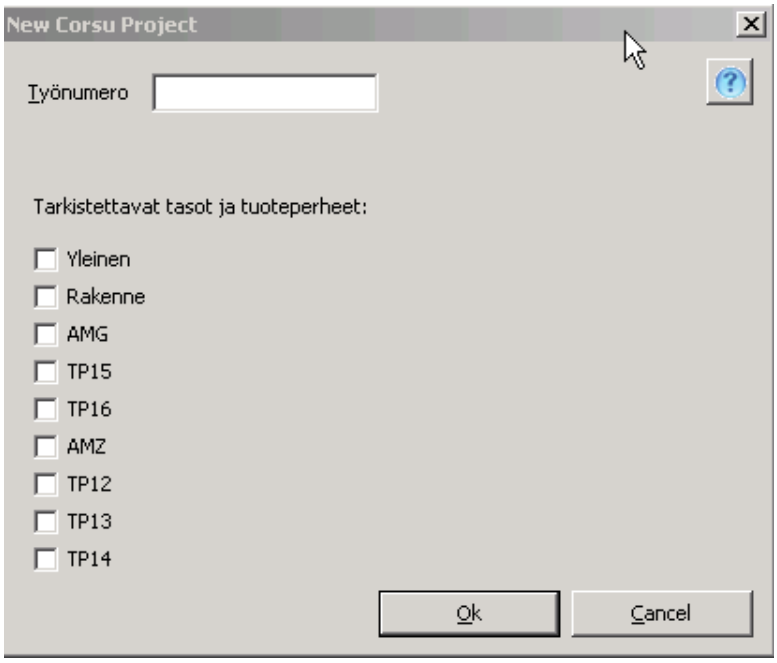
Prototyypille kerättiin tarkistettavia validaatioita yhdelle tuoteperheelle. Taulukossa 4 on lueteltu prototyypille kerättyjen validaatioiden kokonaismäärä sekä toteutetut ja toteuttamatta jääneet validaatiot. Lisäksi taulukosta ilmenee luotujen parametrien määrä, joita käytettiin validaatioiden toteuttamiseen. Taulukosta selviää, että parametreja luotiin enemmän kuin toteutettuja validaatioita.

Jokaista validaatioita kohden luotiin keskimäärin 2 parametria. Tämä johtuu muun muassa siitä, että parametreja on käytetty edelleen hakukriteerinä, jotta haluttu validaatio on voitu toteuttaa. Taulukossa on lisäksi mainittu staattiset validaatiot, jotka liittyvät prototyypin hakemistorakenteeseen. Näiden perusteella prototyyppi käy läpi vain kyseiseen tuotteeseen liittyvät validaatiot sekä valitut validaatiot.

TAULUKKO 4 Toteutetut validaatiot sekä parametrit

Validaatioita yhteensä	50
Validaatioita toteutettu	30
Validaatioita toteuttamatta	20
Parametrien määrä	100
Staattiset validaatiot	15

Toteutetussa prototyypissä on mahdollista valita, mitä validaatioita tarkistetaan. Validaatiot on jaettu kohdan 5.3.2 mukaan yleisiin, tuoteperhettä koskeviin sekä kokoon viittaaviin ryhmiin. Ryhmien mukaisia validaatioita voidaan valita tarkistettavaksi erikseen. Kuviossa 19 on kuvattu lomake, josta osaluettelon tarkistaja voi valita tiettyjä tarkistuksia osaluettelosta. Tarkistaja voi halutessaan tarkistaa osaluettelon osissa ja jatkaa keskeneräistä tarkistustyötä lataamalla tallennetun tarkistuksen. Näin ollen tarkistajan työtä voidaan tukea suunnittelemalla validaatioiden hakemistorakenne vastaamaan tahtikoneen suunnittelun työnkulkua. Tahtikoneen suunnittelija voi esimerkiksi tehdä tietyn työvaiheen mukaiset tarkistukset ja korjata tarkistuksessa ilmenneet virheet ja tämän jälkeen ladata tallennetun tarkistusvaiheen uudelleen tarkistettavaksi. Näin työvaihe säilyy muistissa myös prototyypissä.

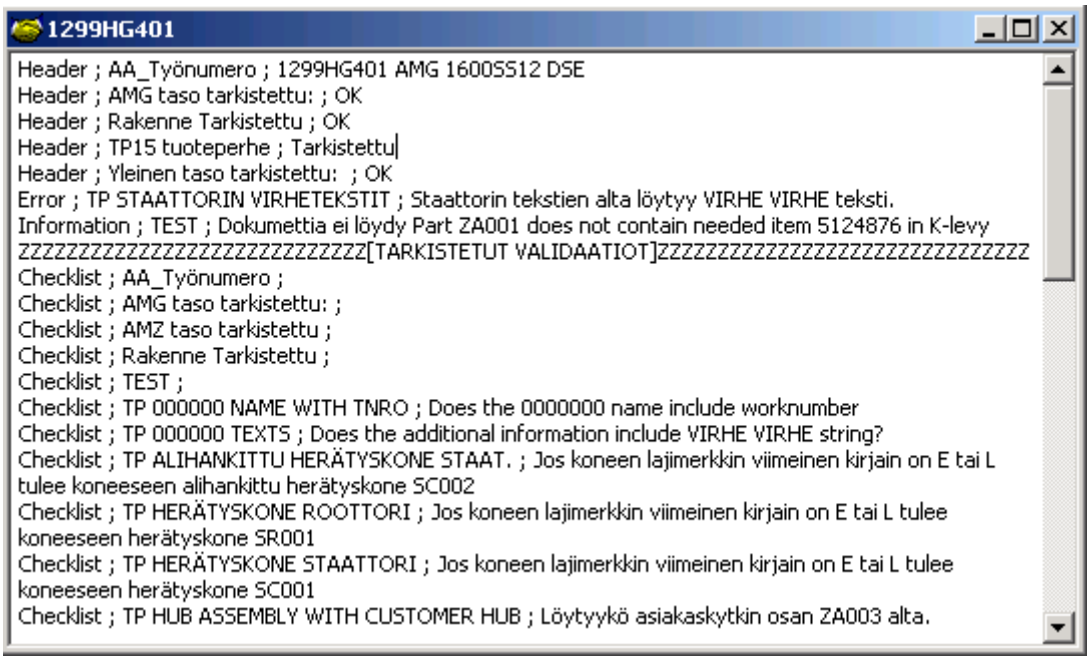


KUVIO 19 Tarkistusvalinnat.

Kuvion 19 valinnat luodaan dynaamisesti validaatioiden hakemistorakenteen mukaisesti. Validaatioiden luontilomakkeella on mahdollista valita, onko hakemisto erikseen valittavissa tarkistettavaksi aloitusvaiheessa. Näin valittavien tarkistusvalintojen muokkaaminen on helppoa hakemistorakenteen muuttuessa.

Prototyypin validaatioiden hakemistorakenne ei tällä hetkellä vastaa työn kulkua. Prototyypin valinnat ovat hyvin yleisiä ja näin ollen toimivat vain esimerkkeinä valintamahdollisuuksista. Hakemistorakennetta ja valintoja on kehitettävä, jotta ne vastaavat suunnittelun kulkua.

Prototyypin tuloksesta ilmenee, mitä validaatioryhmiä on tarkistettu. Kuviossa 20 on kuvattu prototyypin tarkistuksen tuloksia. Tuloksen ensimmäisellä rivillä ilmoitetaan tarkistettavan koneen tiedot. Seuraavaksi luetellaan kaikki valinnat, jotka ovat mahdollisia valita tarkistettaviksi. Valintojen perässä on maininta, jos kyseiseen valintaan liittyvät validaatiot on tarkistettu. Tämän jälkeen tuloksessa luetellaan prototyypin havaitsemat virheet. Lopuksi tuloksessa on listattu kaikki tarkistetut validaatiot ja niiden selitykset. Tulos vastaa suurelta osin alkuperäistä CORSUa. Sen ulkoasua on muutettu selkeämmäksi rajaamalla tarkistettujen validaatioiden lista selkeästi muusta informaatiosta. Alkuperäisestä poiketen tuloksessa näkyvät validaatioryhmät ja niiden status.



KUVIO 20 Prototyypin validointiraportti.

6.1.4 Prototyypin soveltuvuus Tahtikoneet-ympäristöön

CORSUn avulla voidaan toteuttaa perustarkistukset osaluettelolle. Alkuperäinen CORSU ei kuitenkaan kyennyt hakemaan tietoa esimerkiksi SAPista. Lisäksi alkuperäisellä ohjelmalla ei voinut suorittaa tarkistuksia tilanteissa, joissa tie-

don tarkistus koostui monimutkaisista hauista. Lisäksi rakenteen tarkistusta ei voitu toteuttaa halutulla tavalla. Tarkistukseen liittyviä toimintoja kehitettiin prototyyppiin, jotta halutut tarkistukset voitaisiin suorittaa. Taulukossa 5 kuvataan toteutettuja toimintoja sekä testauksessa todettuja tuloksia. Taulukon sisältämät toiminnallisuudet ja tavoitteet ovat samoja, joita kuvattiin tarkemmin luvussa 5.1.

Taulukon 5 tuloksista selviää, että prototyyppiin toteutetuissa toiminnallisuuksissa on puutteita. Osaluettelon rakenne tarkistetaan tällä hetkellä tavalla, joka ei tunne koko rakennetta kerrallaan. Rakenteen tarkistus suoritetaan vertaamalla osaluetteloa annettuun referenssirakenteeseen. Vertailu tapahtuu yksi osa kerrallaan. Jos referenssirakenne ei ole täydellinen, prototyyppi ei voi suorittaa täydellistä tarkistusta. Esimerkiksi, jos referenssirakenteella on mainittu tason 4 osa ja tämän alla tason 6 osa, niin tarkistus ei tunne näiden väliin jäävää osaa ja näin ollen ei voi suorittaa tasolla 6 olevan osan äitiosan tarkistusta. Lisäksi osien alta löytyviä piirustuksia ei voida vertailla muihin tietokantoihin riittävän tarkasti. Tämä johtuu siitä, että Teamcenteriä varten kehitetty SQL-hakufunktio ei tunne tarpeellisen tarkkaa tietoa. Hakufunktioon tarvittaisiin tieto esimerkiksi piirustuksen revisiosta sekä sen nimestä. Tämä koskee myös erikseen mainittua ”piirustusten tarkistus” –toiminnallisuutta.

TAULUKKO 5 Prototyypin soveltuvuus Tahtikoneet-ympäristöön.

Kehitetty toiminnallisuus	**Tavoite**	**Testauksen tulos**
Osaluettelon rakenteen tarkistus	Ohjelma tarkistaa rakenteen ja vertailee eri tietokantojen tietoa rakenteesta.	Rakennetarkistus ei ole riittävän täsmällinen. Teamcenter-tietokannasta haettavan tiedon on oltava tarkempaa.
Tietokantayhteyksien toteutus	Ohjelma voi hakea tietoa Tahtikoneet-yksikköön liittyvistä tietokannoista.	Tarvittavat yhteydet saatiin toteutetuksi.
Tallennus, lataus ja suunnittelun seuranta	Ohjelma voi tallentaa ja ladata raportin. Raportin tulisi tukea suunnittelua.	Tallennus ja lataus onnistuivat. Tarkistuksia voidaan kohdistaa tiettyihin alikokoonpanoihin, mutta alikokoonpanot vaativat tarkempaa määrittelyä.
Piirustusten tarkistus	Ohjelma tarkistaa piirustukset eri tietokannoista osan tunnuksen mukaan.	Haku onnistuu. Haku tarvitsee tarkennuksen liittyen piirustuksen nimeen ja revisioon.

Tietokantayhteydet saatiin toteutetuksi halutulla tavalla. Muun muassa käyttöoikeudet voitiin määritellä niin, että suunnittelijat kykenivät käyttämään prototyyppiä halutulla tavalla. SAPiin liittyvät haut onnistuivat prototyyppiin vaadittujen validaatioiden osalta. SAP-yhteyttä voidaan kuitenkin vielä kehittää, jos tarvitaan monimutkaisempia hakuja SAPista. Yhteys käyttää tällä hetkellä

vain yhtä RFC-funtiota, mutta SAP-komponentit mahdollistavat myös monien muiden RFC-funktioiden käytön. Esimerkiksi hankinta-aloitteiden haku SAPista voisi edesauttaa suunnittelun verifiointia.

Prototyyppiin onnistuttiin myös tekemään raportin tallennus- ja latastoiminto. Latauksen tarkoitus oli, että tarkistukset voidaan suorittaa osittain. Tarkistusten osittaminen tehdastuotteen suunnittelun mukaiseksi vaatii kuitenkin vielä tarkempia määrityksiä ja kattavampaa testausta. Tällä tavoin toiminto voidaan säätää tukemaan tehdastuotteen suunnittelua. Valittavien osarakenteiden on siis vastattava tehdastuotteen suunnittelun kulkua.

Prototyyppi otetaan käyttöön ABB:n Tahtikoneet-yksikössä. Jatkossa validaatioita lisätään iteratiivisesti muille tuoteperheille. Lisäksi prototyyppiä kehitetään tutkimuksen kehitysehdotusten mukaisesti.

6.2 Kehitysehdotuksia

Prototyypin validaatioiden luonnin yhteydessä törmättiin useasti tilanteeseen, jossa olisi ollut tarpeellista tuntea validoitavan osaluettelon rakenne. Ilman rakenteen tuntemusta monet validaatiot eivät ole mahdollisia. Tästä syystä keskeisin kehitysehdotus koskee tuoteperhekohtaisen rakenteen sisällytystä CORSUn toimintaan. Tämän lisäksi validaatioiden muodostamiseen liittyviä muita toiminnallisuuksia esitellään seuraavaksi.

6.2.1 Osaluettelorakenteen sisällyttäminen CORSUun

Jos CORSU tuntee tuoteperhekohtaisen osaluettelon rakenteen, se voi muun muassa tarkistaa, ovatko osat oikeilla paikoillaan. Prototyyppiin kehitetty rakenteen tarkistus ei tällä hetkellä tarkista muuta, kuin onko jokin osa tietyn osan alla. Se ei siis esimerkiksi kerro, onko osa laitettu väärän osan alle, vaan ilmoittaa vain sen puuttumisesta. Tämä johtuu prototyypin toteutuksesta, missä jokainen osa ja sen alaosa vaatii yksittäisen haun tietokannasta. CORSU ei siis tunne koko rakennetta kerrallaan. Tämä johtuu siitä, etteivät parametrit voi ylläpitää tietoa taulukkomuodossa.

Edellä mainittu esimerkki voitaisiin toteuttaa luomalla oma tarkistus, jossa tarkistetaan löytyykö puuttuva osa rakenteen minkään osan alta. Nykyisellään tämä tarkoittaa erillisen haun tekemistä tietokantaan. CORSUn käytölle olisi edullisempaa lisätä tietotyyppi, joka varastoi osaluettelon rakennetiedon taulukkomuodossa. Tällöin PDM-tietokannan osaluettelorakenne voidaan hakea yhdellä rekursiivisella SQL-lauseella ja käyttää tätä yleisesti hyväksi validaatioiden luonnissa. Luomalla vain yksi parametri, joka ylläpitää tietoa rakenteesta, helpotetaan parametrien käyttö merkittävästi nykyiseen verrattuna.

Jos rakenteen tarkistus kohdistuu osien alla oleviin nimikkeisiin tai piirustuksiin, niin se kohdistuu myös SAP- ja DG-tietokantoihin sekä K-levylle. Tällöin myös tiedon hakumassa nousee useisiin satoihin hakuihin jokaista tuote-

perhettä ja tietokantaa kohden. Tällä hetkellä haku SAP-tietokannasta on hidasta. Näin ollen tällaisen tietomassan tarkistus kestää kohtuuttoman kauan. Tämän ongelman ratkaisemiseksi täytyy SAPin RFC-funktioita käyttää tehokkaammin. Yhtenä ratkaisuna voisi olla SAPin käyttöliittymän kautta tehtävien tietokantakyselyiden käyttö RFC-funktioiden välityksellä. Tällöin CORSUun voitaisiin luoda SAP-haku, joka käyttää SAPiin tehtyjä tietokantakyselyitä, jotka taas hakevat tiedon SAPin tietokannasta. Koska kyselyt tietokantaan luotaisiin SAPin käyttöliittymässä, CORSUun ei tarvitsisi määritellä kuin käytettävä kysely. Näin ollen tarkempi suunnittelu tiedon hausta tapahtuisi SAPissa. Tällainen ratkaisu voisi sopia esimerkiksi hankinta-aloitteiden hakuun SAPista.

Prototyypin käyttö on lisäksi osoittanut osaluettelon rakenteen olevan tärkeässä asemassa validaatioita luotaessa. Lisäksi se on osoittanut, että rakenteen määrittely ja siihen kohdistuvat tarkistukset ovat monimutkaisia. Sovittamalla MOKA-menetelmän mukaista, rakenteellista tietämyksen keräystä saadaan selkeämpi ja kattavampi kuva rakenteesta. Suunnittelemalla validaatiot osaluettelon mukaisen mallin yhteyteen, määritellään samalla rakenne sekä se, mitä rakenteesta voidaan tarkistaa. Tällainen suunnittelu on oleellisessa asemassa etenkin, jos validaatioita varten täytyy toteuttaa uusia toiminnallisuuksia.

CORSUn validaatiot voidaan nykyisellään jäsentää eri tasoihin kansioiden avulla. Kansioita voidaan soveltaa myös tarkempaan osaluettelon mukaisen rakennetietämyksen tallentamiseen. Jokaiselle osalle voidaan luoda oma kansio ja kansiot voidaan jäsentää sisäkkäin rakenteen mukaisesti. Tällä tavoin validaatiot jäsentyvät selkeäksi kokonaisuudeksi, josta on helppo löytää kuhunkin osaan kohdistuvat validaatiot, ja osien väliset tarkistukset pysyvät ymmärrettävinä. Tällaista mallia tukee myös MOKA-menetelmä, missä jokainen validaatio liitetään osaan. MOKAn mukaiset mallit soveltuvat hyvin tämän tapaiseen validaatioiden jäsentämiseen. Lisäksi mallit kannustavat hankkimaan kattavampaa tietämystä validaatioista ja tätä kautta prosesseista ja koko osaluettelosta.

6.2.2 Kehitysehdotuksia muihin toiminnallisuuksiin

Toisin kuin parametreissa, validaatioissa ei voida käyttää jokerimerkkejä – eli merkkejä, jotka vastaavat mitä tahansa merkkiä tai merkkijonoa. Validaatioiden kannalta olisi edullista, jos validaatioiden yhteydessä voitaisiin käyttää jokerimerkkejä. Näin samoja parametreja voitaisiin hyödyntää laajemmin validaatioissa. Nykyisellään validaatiot sisältävät useita parametreja, joiden tehtävänä on hakea tietoa merkkijonosta. Usein tällaisissa tapauksissa merkkijonosta tiedetään varmasti vain, että se sisältää esimerkiksi mitan. Muut merkkijonon merkit ovat tuntemattomia. Jokerimerkkien avulla mitta voitaisiin etsiä validaatioiden ehtolauseiden avulla. Tällöin samaa merkkijonoa voitaisiin käyttää myös muissa validaatioissa, koska merkkijono saattaa sisältää myös muuta tarvittavaa tietoa.

Parametrien toteuttamiin Teamcenter-tietokantahakuihin tarvitaan mahdollisuus hakea tarkempaa tietoa piirustuksista. Tämä tarkoittaa, että tietokantaan liittyvä hakufunktio täytyisi päivittää vastaamaan Tahtikoneet-yksikön

tarpeita. Hakufunktioon tarvitaan lisätietona piirustuksen nimi ja revisionumero. Hakufunktion päivittäminen vaatii tarkempaa Teamcenter-tietokannan viittausten tuntemusta.

6.2.3 Kehitysehdotuksia MOKAn sovittamiseen

MOKA-menetelmä vaatii sovittamista, jotta sen käyttö olisi mahdollista Tahtikoneet-yksikössä. Silloin sen käytöstä voidaan saada suurtakin hyötyä, ajatellen prototyypin jatkokehitystä sekä sen ylläpitoa. Menetelmän eri lomaketyyppien avulla voidaan ylläpitää tehdastuotteiden suunnittelun tietämystä kattavammalla ja selkeämmällä tavalla. Lisäksi sovittamalla formaaleja malleja voidaan ylläpitää tietoa käytetyistä parametreista ja mallintaa näiden käyttöä.

Tämän tutkimuksen perusteella MOKAn määrittelemää ICAREa voidaan käyttää jo sellaisenaan, kohdan 4.6 kuvailemalla tavalla. Formaalit mallit vaativat kuitenkin luokkien ja niiden käytön sovittamista. Formaaleja malleja koskevaa sovittamista kuvattiin kohdissa 5.2 ja 5.3. Sovittaminen tarkoittaa luokkien ja niiden käytön määrittämistä etenkin parametrien osalta. Kohdassa 5.3 anettaan esimerkki, millä tavoin luokkien määrittäminen tapahtuu ja mihin se perustuu. Tällaista tietämyksen varastointia voidaan pitää implisiittisenä suhteessa koko tehdastuotteen suunnittelutietämyksen rakenteeseen. Tällainen tietämyslähtöisten suunnittelujärjestelmien lähestymistapa tuotiin esille kohdassa 3.3. Tiedon implisiittisessä varastoinnissa on kuitenkin tarkasteltava pois jätettävän tiedon tarpeellisuutta.

Jos parametreja ei haluta mallintaa formaalien mallien avulla, niille voidaan myös suunnitella oma paikkansa ICAREn mukaisesta mallista. Tällöin ICARE voi ylläpitää tarvittavaa tietoa parametreista. Tällainen lähestyminen ei kuitenkaan tue yhtä vahvasti järjestelmään siirrettävän tietämyksen validointia kuin formaali mallinnus. Sovittamalla MOKA-menetelmän mukaista formaalia mallintamista, mahdollistetaan muitakin kehitysmahdollisuuksia tulevaisuudessa. Formaalit mallit ovat välttämättömiä, jos tulevaisuudessa katsotaan tarpeelliseksi esimerkiksi toteuttaa toiminnallisuus, jossa tietämys voidaan siirtää automaattisesti malleista validointityökaluun.

Kohdassa 6.2 tuotiin esille MOKAn olevan liian raskas esimerkiksi Tahtikoneet-yksikön tarpeeseen sellaisenaan. Lisäksi luvussa mainittiin, kuinka suurten yritysten on vaikea käyttää MOKAn kaltaisia menetelmiä, koska niiden tuotetiedon rakenne ja siihen liittyvä tietämys on hyvin monimutkaista ja yksilöllistä. Sovittamalla MOKA-menetelmää voidaan rakentaa asteittain kattavaa tietämysvarastoa. Parhaassa tapauksessa rakenne voidaan tulevaisuudessa koostaa kokonaisvaltaiseksi tietämysvarastoksi, johon MOKAkin pyrkii. Tällöin esimerkiksi CORSU-prototyypin kaltainen ohjelma voisi toimia tietämyksen ylläpitäjänä, joka ylläpitää aiempaa eksplisiittisempää tietoa rakenteesta. Lisäksi prototyyppi voi ylläpitää tietämystä rakenteen suunnittelusta ja siihen liittyvistä ohjeista. Tällöin prototyyppi toimisi tietämyksen kokoamispaikkana.

Suunnittelun tietämystä voidaan ylläpitää esimerkiksi viittauksilla suunnitteluohjeisiin. Jos tällainen tietämysvarasto saavutetaan, niin sen hyödyt voi-

vat olla hyvinkin suuria. Se voi esimerkiksi toimia apuna tarkemmassa tuotealustojen ja tuoteperheiden suunnittelussa. Esimerkiksi kohdassa 2.2 tuotiin esille, että suunnittelijoiden on kyettävä toimimaan systemaattisella tavalla, jotta yritys kykenee vastaamaan vaihteleviin asiakasvaatimuksiin. Tähän liittyen tuotealustojen ja tuoteperheiden suunnittelulla voidaan saavuttaa suuria hyötyjä (ks. kohta 2.3). Lisäksi kattavan tietämysvaraston avulla tuotesuunnittelun avuksi voidaan kehittää hienostuneempia automaatiotyökaluja.

7 YHTEENVETO

ABB Oy:n Tahtikoneet-yksikön tavoitteena on luoda työkalu, jonka avulla yksikön suunnittelijat voivat tarkistaa osaluettelon kelvollisuuden eli validoida sen. ABB:n osaluettelo on ainutlaatuinen rakenteellinen luettelo suunniteltavan koneen osista. Osaluettelon suunnittelu sisältää monia muuttujia ja sääntöjä, liittyen suunnitteluprosessiin ja suunniteltavaan koneeseen. Lisäksi osaluettelon sisältämä tietämys on hajanaista ja sitä ylläpidetään monissa eri tietämyslähteissä. Tahtikoneiden suunnittelussa on käytetty hyväksi tuotealustalähtöistä lähestymistapaa, missä kukin tahtikone kuuluu johonkin tuoteperheeseen ja kukin tuoteperhe edelleen johonkin tuotealustaan.

Validointityökalu on ohjelma, jonka avulla voidaan tarkistaa osaluettelon kelvollisuus. Tarkistus tapahtuu ohjelmaan syötettyjen tarkistuskriteerien eli validaatioiden avulla. Ohjelmaan syötetään validaatiot, jotka ohjelma tarkistaa muista tietokannoista haetun tiedon avulla. Ohjelma suorittaa tiedonhaun jokaisen suoritetun validoinnin yhteydessä. Lisäksi ohjelma vertailee muiden tietokantojen sisältämää tietoa tutkiakseen sisältävätkö eri tietokannat saman tiedon osiin liittyen.

Tässä tutkimuksessa kehitettiin validointityökalun prototyyppi tahtikoneiden valmistuksen yhteyteen. Prototyypin suunnittelussa otettiin huomioon erityisesti käyttötarkoitus ja ympäristö. Prototyypin ympäristöön liittyen huomioitiin muun muassa eri tietokannat, joista tietoa tarkistetaan. Käyttötarkoitukseen liittyen huomioitiin sen toiminnallisuus sekä kuinka helposti ohjelma on ylläpidettävissä. Tärkeimpänä toiminnallisuutena huomioitiin validointisuorituksen luotettavuus.

Validaatioiden luotettavuus koetaan pullonkaulaksi myös tietämyslähtöistä suunnittelua käsittelevissä tutkimuksissa. Tietämyslähtöisen suunnittelun käsitettä voidaan yleisesti kuvailla suunnittelun menetelmänä, jossa tietämys tehdastuotteesta varastoidaan tehdastuotteelle ominaisella tavalla sekä käytetään hyväksi tehdastuotteen suunnittelussa, analysoinnissa ja valmistuksessa. Tietämyslähtöinen suunnittelu tutkii muun muassa, kuinka tietämys voidaan koostaa rakenteelliseksi malliksi, ja siihen on kehitetty erityisiä menetelmiä. Koska osaluetteloon liittyvä tietämys muodostuu sen rakenteen ympärille, sopii

tietämyslähtöisen suunnittelun mukainen lähestymistapa myös validointityökalun suunnittelun lähtökohdaksi.

Tietämyslähtöisestä suunnittelusta on hyvin vaikea luoda yksiselitteistä määritelmää. Tästä johtuen tietämyslähtöinen suunnittelu edustaa monessa yrityksessä niin sanottua hämärää tiedettä. Kirjallisuudesta löytyy kuitenkin määritelmiä, joita tässä tutkimuksessa käytettiin hyväksi. Määritelmien avulla saatiin kuva esimerkiksi siitä, mitä tietämyslähtöisten järjestelmien on tarkoitettu tekevän sekä millaiseen ympäristöön ne liittyvät. Tämän lisäksi kirjallisuudesta löytyi menetelmiä, jotka oli kehitetty tietämyslähtöiseen suunnitteluun.

Tutkimuksessa esiteltiin ja vertailtiin muutamia tietämyslähtöisen suunnittelun menetelmiä. Vertailun perusteella menetelmän valinta kohdistui MOKA-menetelmään, koska sen katsottiin olevan kelvollisin vaihtoehto Tahtikoneet-yksikön tarpeeseen. MOKAn avulla tietämyslähtöisen suunnittelun mukaisia järjestelmiä voidaan kuvata tasoina käyttämällä avuksi ehtoja, prosesseja, mallintamismenetelmiä ja määrityksiä. Se tarjoaa viitekehyksen tiedon esittämiseen ja varastointiin. Tämä viitekehys voidaan jakaa vapaamuotoiseen ja formaaliin tasoon, joille molemmille MOKA tarjoaa omat mallinnusmenetelmänsä. Vapaamuotoista tietämystä voidaan kerätä ja mallintaa ICARE-lomakkeiden avulla. Formaali mallinnus onnistuu MOKAn tarjoaman oliokeskeisen kielen avulla.

Tässä tutkimuksessa löydettiin kolme varteenotettavaa tapaa toteuttaa itse validointityökalu. Toteutus voitaisiin tehdä alusta loppuun omaksi itsenäiseksi sovelluksekseen tai vaihtoehtoisesti voitaisiin käyttää olemassa olevaa CORSU-ohjelmaa ja muokata tätä ohjelmaa Tahtikoneet-yksikön käyttötarpeita vastaavaksi. Kolmantena vaihtoehtona oli rakentaa validointityökalu käytössä olevan PDM-järjestelmän yhteyteen. PDM-järjestelmänä ABB:lla on Teamcenter, joka voisi toimia lähtökohtana toteutettavalle validointiyökalulle. Tässä tutkimuksessa päädyttiin käyttämään CORSUa, koska tämän toiminnallisuus on hyvin samankaltainen tutkimuksen tavoitteen kanssa. Lisäksi huomattiin, että CORSUn toiminnallisuuteen kyettiin yhdistämään MOKA-menetelmän mukaisia käytänteitä.

CORSUsta rakennettiin prototyyppi vastaamaan Tahtikoneet-yksikön tarpeita. CORSU ei sellaisenaan täysin soveltunut Tahtikoneet-yksikön tarpeisiin. Prototyyppiin jouduttiin tekemään muutoksia, liittyen sen ympäristöön sekä käyttötapaan. Muun muassa prototyypin tiedonhakuvaatimukset poikkesivat alkuperäisestä. Tahtikoneet-yksikössä haluttiin validointityökalun hakevan tietoa myös SAP-järjestelmästä. SAPin ollessa hyvin suljettu järjestelmä, haut eivät onnistuneet alkuperäisillä toiminnallisuuksilla. Prototyyppiin kehitettiin toiminnallisuus, joka toteuttaa haut SAPin omien komponenttien avulla. CORSUn käyttötapaan liittyen tarvittiin uusia toiminnallisuuksia, jotta validaatioiden suorittaminen voitaisiin sovittaa tahtikoneiden suunnitteluprosessiin. CORSUn tulisi siis suorittaa validaatioita suunnittelun eri vaiheiden aikana. Tähän liittyen prototyyppi kehitettiin toimimaan siten, että validaatioita suoritettaessa voidaan valita tarkistettavat osaluettelon kokonaisuudet, tallentaa tämä tarkistus sekä jatkaa tarkistusta myöhemmin muiden kokonaisuuksien osalta.

Tutkimuksessa selvitettiin, että MOKAn mukaisen menetelmän avulla voidaan muun muassa yhtenäistää tuotteiden suunnittelua ja mahdollistaa suunnittelun läpivieminen systemaattisella tavalla. Tuotteiden suunnittelun samankaltaisuus ja systemaattisuus olivat samalla edellytyksiä toimivalle tuotealusta- ja tuoteperhekohtaiselle suunnittelulle. Prototyypin suunnittelua ja käyttöä verrattiin MOKAn mukaiseen tietämyksen keräykseen. Tämän perusteella suunniteltiin, kuinka MOKAa voidaan käyttää CORSUn yhteydessä niin, että saavutetaisiin tietämyslähtöisen lähestymistavan tarjoamia hyötyjä. Prototyypin käyttöön ja ylläpitoon liittyen, yksi merkittävä MOKAn tarjoama hyöty oli tietämyksen rakenteellinen varastointi. Rakenteellisuuden huomattiin olevan edellytys validointityökalun ylläpidolle sekä kehitykselle. Rakenteellisten mallien avulla pystyttiin paremmin ymmärtämään ja paikallistamaan validaatiot.

Lopuksi tutkimuksessa arvioitiin miten MOKA ja CORSU soveltuvat tahtikoneiden suunnittelun yhteyteen. Tutkimuksessa selvisi, ettei MOKA-menetelmä täysin sellaisenaan sovellu käytettäväksi validointityökalun kehityksessä. Tutkimuksessa annettiin kuitenkin esimerkki siitä, millä tavoin MOKAa voidaan sovittaa tähän tarkoitukseen. Lisäksi annettiin esimerkki, miten jatkossa validaatioiden suunnittelu ja toteutus voidaan suorittaa. Tutkimus antoi myös ehdotuksen, kuinka sovitetun menetelmän yhteyteen voitiin sitoa koko validointityökalun tuleva kehitys. Tässä tietämyksen hankinta ja validointityökalun toiminnallisuuksien kehitys kulkivat rinnakkain. Tutkimuksen tuloksissa arvioitiin, kuinka hyvin onnistuttiin vaatimusmäärittelyn mukaisten toiminnallisuuksien toteutuksessa sekä millaisiin rajoittaviin tekijöihin törmättiin prototyypin toteutuksen aikana. Esimerkiksi tietokantojen käyttö ja tietämyksen siirtäminen järjestelmään tuottivat ongelmia. Tutkimuksessa annettiin kehitysehdotukset näiden toiminnallisuuksien jatkokehittämisestä.

Kokonaisuudessaan tämä tutkimus toi esille uuden näkökulman validointityökalun kehitykseen. Tutkimuksessa kävi ilmi, että validaatioiden muodostaman tietämyksen hankinta vaatii selkeän menetelmän. Tietämyslähtöisen menetelmän avulla voidaan varmistaa, että toteutettavat validaatiot suorittavat tarvittavat oikeat tarkistukset. Uutena näkökulmana tutkimus toi kehityssuunnan, jonka avulla voidaan samalla kehittää koko tuotekehityksen tietämyksen varastointia. MOKA-menetelmän avulla voitiin luoda tietämysrakenne, johon tutkittu tietämys kootaan. Samaan rakenteeseen voitiin yhdistää myös muu tuotesuunnitteluun liittyvä tietämys. Muu tietämys voitaisiin koota myös validointityökaluun, esimerkiksi viittauksina eri tietämyskantoihin. Tällä tavoin prototyypin ja tietämyslähtöisen menetelmän avulla voidaan yhtenäistää ABB:n tuotesuunnittelun tietämys kokonaisvaltaiseksi rakenteeksi, missä validointityökalu voi toimia tietämysrakenteen ylläpitäjänä. Näin syntyisi järjestelmä, jonka viittausten kautta löytyy kaikkiin rakenteisiin ja niiden komponentteihin liittyvä tietämys.

LÄHTEET

ABB Oy (2010). *ABB Inside*. Haettu 29.12.2010 osoitteesta: http://inside.abb.com

Akerkar, A., Sajja, P. (2010). *Knowledge-based systems*. Sudbury, MA: Jones and Bartlett.

Ammar-Khodja, S., Perry, N., Bernard, A. (2008). Processing knowledge to support knowledge-based engineering systems specification. *Concurrent Engineering*, 16(1), 89-101 Haettu 23.11.2010 osoitteesta http://cer.sagepub.com/content/16/1/89.

Badiru, B.A. (2006). *Handbook of Industrial and System Engineering*. Boca Raton, FL: Taylor and Francis Group.

Brown, E.L. (2006). *SQL Server Distilled.* Boston, MA: Addison-Wesley.

Brimble, R., Sellini, F. (2000). The MOKA Modeling Language. Teoksessa R. Dieng, O. Crosby (toim.), *EKAW 2000* (49-56). Berliini: Springer.

Chapman, C.B, Pinfold, M. (1999). Design engineering – a need to rethink the solution using knowledge based engineering. *Knowledge-Based Systems*, 12(5-6), 257-267.

Du, X., Jiao, J., Tseng, M. (2001). Architecture of product family: fundamentals and methodology. *Concurrent Engineering*, 9(4), 309-325. Haettu 26.11.2010 osoitteesta http://cer.sagepub.com/content/9/4/309

Fan, I., Bernell-Garcia P. (2008). International standard development for knowledge based engineering services for product lifecycle management. *Concurrent Engineering*, 16(4), 271-277. Haettu 17.11.2010 osoitteesta http://cer.sagepub.com/content/16/4/271.

Genworks (2011). *Genworks International.* Haettu 26.4.2011 osoitteesta: http://www.genworks.com.

Hallman, M.I.J., Hofer, P. A., van Vuuren, W. (2006). Platform-driven development of product families. Teoksessa T.W, Simpson, Z. Siddique, J. Jiao, *Product Platform and Product Family Design: Methods and Applications* (27-47). New York: Springer.

Hou, S., Liu, Y., He, L., Zhao, W., Wang, W. (2010). Research on knowledge-based engineering system for rapid response design of machine tool. *Chinese Control and Decision Conference 2010, Xuzhou, toukokuu 26-28* (4310-4314). Singapore: IEEE Industrial Electronics Chapter.

Hunter, R., Rios, J., Perez, J.M., Vizan, A. (2005). A Functional approach for the formalization of the fixture design process. *International Journal of Machine Tools and Manufacture*, 46, 683-697.

Ishikawa, Y. (2003). Activity model for product development/production. Teoksessa J. Cha (toim.), *The Vision for the Future Generation in Research Application*, 2003, Lisse: Sweet & Zeitlinger.

Karjalainen, J. (2010). Teamcenter-ohje. Helsinki: ABB Oy.

Klein, R. (2000). Knowledge modeling in design: the MOKA framework. Teoksessa: J.S. Gero (toim.). *Proceeding of the International AI Design.* Worcester, MA: Kluwer.

Kulon, J., Broomhead, P., Mynors, D. J. (2005). Applying knowledge-based engineering to traditional manufacturing design. *The International Journal of Advanced Manufacturing Technology*, 30(9-10), 945-951.

Lovett, P. J., Ingman, A., Bancroft, C. N. (2000). Knowledge-based engineering for SMEs – a methodology. *Journal of Materials Processing Technology*, 107(1-3), 384-389.

Meyer, M.H., Lehnerd, A.P. (1997). *The Power of Product Platforms: Building Value and Cost Leadership*. New York, NY: Free Press.

Mulvenna, M.D., Huges, J.G. (1999). Expert agents in knowledge-based systems. Teoksessa *Proceeding of Expert System 1993,* Cambridge: Information Press.

Nayak, R.U., Chen, W., Simpson, T.W. (2002). A Variation-based methodology for product family design. *Engineering Optimization*, 34(1), 65-81.

Oldham, K., Kneebone, S., Callot, M., Murton, A., Brimble, R. (1998) MOKA – A Methodology and tools oriented to knowledge-based engineering applications. Teoksessa: S. Björgvinsson, *Changing the ways we work: shaping the ICT-solutions for the next century.* Amsterdam: OIS Press.

Pahl, P., Beitz, W., Feldhusen, J., Grote, K. H. (2007). *Engineering Desing: A Systematic Approach* (3. painos). London: Springer.

Penoyer, J. A., Burnett, G., Fawcett, D. J. (1999). Knowledge based product life cycle systems: principles of Integration of KBE and C3P. *Computer-Aided Design*, 32(5), 311-320.

Rasovska, I., Chebel-Morello, B., (2008). A mix method of knowledge capitalization in maintenance. *Journal of Inteligent Manufacturing*, 19(3), 347-359.

Ritchie, S. G., Harris, R. A. (1987). Expert systems in transportation. Teoksessa *Expert System for Civil Engineering*. New York: American Society of Civil Engineers.

Robertson, D., Urlich, K. (1998). Planning for Product Platforms, *Sloan Management review*, 39(4), 19-31.

Romanowski, C., Rakes, N. (2005). On comparing bill of materials: a similarity/ distance measure for unordered trees, *IEEE Transactions on Systems, Man, and Cybernetics*, 35(2), 249-260.

Rumbaugh, J., Jacobson, I., Booch, I. (1999). *The Unified Modeling Language Reference Manual*. Boston, MA : Addison-Wesley.

Sainter, P., Oldham, K., Larkin, A., Murton, A., Brimble, R. (2000). Product knowledge management within knowledge-based engineering systems. Teoksessa *Proceedings of DETC'00 ASME 2000 Design Engineering Technical Conference and Computers and Information in Engineering Conference.*

Sajja, P. S. (2000). *Knowledge-based systems for socio-economic rural development*. Väitöskirja, Sardar Patel University.

Sandberg, M. (2003). *Knowledge based engineering in product development*. (Raportti 2005:05). Lulea university of technology, Department of Applied Physics and Mechanical Engineering.

SAP (2011). *Programming with Business Objects.* Haettu 24.4.2011 osoitteesta: http://help.sap.com/

Shrobe, E. H. (1988). Exploring artificial intelligence. Teoksessa *The National Conference on Artifical Intelligence* (297-346), San Mateo, CA: Morgan Kaufman.

Simpson, T.W., Siddique Z., Jiao J. (2006). Platform-based product family development. Teoksessa T.W Simpson, Z. Siddique, J. Jiao, *Product Platform and Product Family Design: Methods and Applications* (1-15). New York: Springer.

Stokes, M. (2001). *Managing Engineering Knowledge*. Suffolk: Professional Engineering Publishing Limited.

UGS Corp. *Teamcenter Engineering Process Management: Integration Toolkit Programmer'sGguide*. Banbury: UGS

Ulengin, F. Topcu, Y. I. (1997). Cognitive map-KPDSS integration in transportation planning. *Journal of the Operational Research Society*, 48, 1065-1075.

Van Vliet, H. (2000). *Software Engineering: Principles and Practice* (2. painos). Chichester: Wiley.

Wiegers, K.E. (2003). *Software requirements* (2. painos). Washington: Microsoft press.

LIITE 1 OSALUETTELO

Level	Part ID	Revision	Part name	Stat/time diff./subcontr.	Included to	Prepared	Approver				Weight(kg)
2 120	3AFP200079047	A	Markings and tags ZA002	//	3AFP200079046	Riikonen, Heikki			Rev. date 2011-02-09		2.91
Part no. DRAWINGS	ID	R Description	Functional/stock code	Dimensions	Material	Mat.std	Dim.std	PCS	Net(kg)	Gross	Unit
	3AFP5155881-D	A PRINCIPLE OF EMERGENCY COOLING	AH								
	3AFP9874871-D	C RATING PLATE FOR SYNCHRONOUS GENERATORS	ZT								
	3AFP9957088-D	A PLATES AND STICKERS	AH								
PARTS/MATERIALS/MODULES	3AFP9871813										
1		A NAME PLATE ABB (LARGE)	YKL ABB2		PC	PC-GR/RT		1	0.200	1.000	KPL
3	3AFP102111166	A MARKING PLATE FOR DIR. OF ROT. --> UNIDIRECTIONAL	YKLP 3036					1	0.012	1.000	KPL
4	3AFP5116757	B OIL FILLING SHIELD	ZA		Al 99.5 H14			2	0.032	2.000	KPL
5	3AFP9872115	A TYPE DESIGNATION PLATE	YKL AMG 1600		AlSiMg T6	SFS 2593	SFS 2582	1	0.080	1.000	KPL
6	3AFP9874871	B RATING PLATE	YKL ABB148X80 AMG					1	0.020	1.000	KPL
7	3AFP5116770	B WARNING PLACARD	ZA					2	0.028	2.000	KPL
8	3AFP5116792	C MAIN TERMINAL BOX PLACARD	ZA					1	0.007	1.000	KPL
9	3AFP5116789	C TERMINAL BOX PLACARD	ZA	1X30X80				1	0.007	1.000	KPL
9	3AFP5116790	C EXTRA TERMINAL BOX PLACARD	ZA	1X30X80				1	0.007	1.000	KPL
10	3AFP5125738	B HEATER PLACARD	ZA	1x30x80				3	0.021	3.000	KPL
11	3AFP5130131	B EARTH PLACARD	ZA					4	0.004	4.000	KPL
12	3AFP5138134	C GENERATORS BADGE	ZA					1	1.040	1.000	KPL
13	3AFP5128098	C EMERGENCY COOLING SIGNBOARD	ZA					10	1.000	10.000	KPL

Identifier	Revision	Description	Functional/stock code	Machine type	Drive code	Printed	Status	Page
3AFP200079017	A	1916HG103 AMG 1600QM14 LSE 0000000				2011-04-08	Approved	8/157
1916HG103								

LIITE 2 ENTITEETIT-LOMAKE

iCARE form: entiteetti				Viittaukset entiteetteihin				Muut liittyvät rajoitteet, säännöt, kuvaukset		
Nimi	**Referenssi**	**Tyyppi**	**Käytäntyminen**	**Toiminnot**	**Äiti**	**Lapsi**	**Konteksti**	**Kuvaus**		**Kaavakkeen hallinnat[Tekijä, pvm, versio, status]**
Pääkokoonpano	00000000	Rakenteellinen	K1		TNRO	ZA001	Yleinen	Jokaisen koneen rakenteella on päätaso, jonka alle tulevat muut osat.		Hyvönen, 3.3.2011, 0.1, keskeneräinen
Pakollinen osa	K1	Käyttäytyminen					Yleinen	Pakollinen osa löytyy kaikista konteksiin koneista		Hyvönen, 3.3.2011, 0.1, keskeneräinen
Valinnainen osa	K2	Käyttäytyminen					Yleinen	Valinnainen osa asetetaan rakenteelle tilauskohtaisesti.		Hyvönen, 3.3.2011, 0.1, keskeneräinen
Työnumero	TNRO	Rakenteellinen				0000000	Yleinen	Rakenne		Hyvönen, 3.3.2011, 0.1, keskeneräinen
Pakkaus ja maalaus	ZA001	Rakenteellinen			00000000	AA001, ZA002, ZA003, ZA040-049, AS001, ZP100	Yleinen	Osa sisältää pakkaukseen ja maalaukseen liittyvät muut osat ja nimikkeet		Hyvönen, 3.3.2011, 0.1, keskeneräinen
Mechanical drawings	AA020	Rakenteellinen			AA001		TP15	KU1	Osa sisältää mekaanisen suunnittelun piirustukset.	Hyvönen, 3.3.2011, 0.1, keskeneräinen
Electrical drawings	AA030	Rakenteellinen			AA001		Yleinen	KU2	Osa sisältää sähkösuunnittelun piirustukset.	Hyvönen, 3.3.2011, 0.1, keskeneräinen
Markings and tags	ZA002	Rakenteellinen			ZA001		Yleinen		Sisältää kilvet ja tarrat	Hyvönen, 3.3.2011, 0.1, keskeneräinen
Hub assembly	ZA003	Rakenteellinen			ZA001	ZA004	Yleinen		Sisältää kytkimen asennuksen	Hyvönen, 3.3.2011, 0.1, keskeneräinen

LIITE 3 SÄÄNNÖT-LOMAKE

ICARE form: säännöt Nimi	Referenssi	Tavoite	Konteksti, informaatio/warning/error	Kuvaus
Pääkoonpanon teksti tarkistus	R1	Tarkistetaan löytyykö pääkokoonpanosta "VIRHE VIRHE" tekstiä	Yleinen, Error	DG järjestelmä kirjoittaa pääkokoonpanon tekstiihin "VIRHE VIRHE", jos pääkokoonpanolla ei ole rakennetta.
Pääkokoonpanon nimen tarkistus	R2	Tarkistetaan löytyykö pääkokoonpanon nimestä työnumero	Yleinen, Error	Pääkokoonpanon nimi sisältää aina myös työnumeron.
Työnumeron tarkistus	R3	Tarkistetaan löytyykö työnumeron alta 0000000 taso	Yleinen, Error	Kaikilla työnumeroilla täytyy löytyä pääkokoonpano.
Rakenteen tarkistus	R4	Tarkistetaan onko valmistettavien osien rakenne vakiodun osaluettelorakenteen m	AMG, Error	Konelle on määritelty tietty vakiorakenne. Tässä tarkistetaan löytyykö osia jotka ei löydy vakiorakenteesta.
Rakenteen ulkopuolisten osien tarkistus	R5	Tarkistetaan löytyykö vakio-osaluettelon ulkopuolisia osan tunnuksia	Yleinen, Error	Konelle on määritelty tietty vakiorakenne. Tässä tarkistetaan löytyykö vakiorakenteen mukaiset valmistettavat osat oikeilla tasoillaan.
Peitteen tarkistus	R6	Tarkistetaan löytyykö osan alta nimiketä nimellä ""TARPAULIN""	Yleinen, Error	Jokaiselle ZA001 osalla on oltava nimikkeenä peite. Tässä tarkistetaan nimikkeen nimen perusteella löytyykö nimketä rakenteelta.
Kilaterästen tarkistus	R7	Tarkistetaan löytyykö osan alta kilaterästä lajimerkillä ""BDER""	AMG, Error	Jokaiselle ZA001 osalla on oltava nimikkeenä kilateräs. Tässä tarkistetaan nimikkeen lajimerkin perusteella löytyykö nimiketä rakenteelta.
Dokumenttien tarkistus	R8	Tarkistetaan löytyykö osan alta tietyt mekaniikkasuunnitelun piirustukset.	TP15, Informaatio	Osa AA020 sisältää yleiset mekaniikkasuunnitelun piirustukset.
Dokumenttien tarkistus	R9	Tarkistetaan onko dokumentteja julkaistu.	Yleinen, Informaatio	??? ks. R8
Dokumenttien tarkistus	R10	Tarkistetaan löytyykö osan alta tietyt sähkösuunnitelun piirustukset.	TP15, Informaatio	Osa AA030 sisältää yleiset sähkösuunnitelun piirustukset.
Löytyykö ZA002 rakenteella	R11	Löytyykö ZA002 rakenteella	Yleinen, Error	Rakenteella on aina ZA002
Kilpipiirustuksen tarkistus	R12	Tarkistetaan löytyykö nimikkeen nimen perusteella kilpipiirustus osan ZA002 alta	TP15, Error	TP15 koneissa tulee olla kilpipiirustus osan ZA002 alla. Nimikkeen nimi voi olla ""RATING PLATE"" tai ""KILVET JA TARRAT""
Kytkimen tarkistus	R13	Tarkistetaan onko osaa ZA003 olemassa	Yleinen	TP15 koneissa tulee olla kytkin
	R14			

Muut liittyvät aktiviteetit	Liitetyt säännöt	Liitetyt rajoitteet	Muut liittyvät entiteetit	Muut liittyvät kuvaukset	Tiedon alkuperä	Kasvakkeen hallinta[Tekijä, pvm, versio, status]	TC	Glove	Nestori	THW	K-levy	SAP
			00000000			Domnina, 3.3.2011, 0.1, kesken.	x					
			00000000			Hyvönen, 3.3.2011, 0.1, kesken.	x					
			TNRO			Domnina, 3.3.2011, 0.1, kesken.	x					
R5			00000000			Hyvönen, 3.3.2011, 0.1, kesken.	x					
			00000000			Domnina, 3.3.2011, 0.1, kesken.	x					
			ZA001			Hyvönen, 3.3.2011, 0.1, kesken.	x					
			ZA001			Domnina, 3.3.2011, 0.1, kesken.	x					
R9			AA020	KU1		Hyvönen, 3.3.2011, 0.1, kesken.	x				x	
			AA020	KU1		Domnina, 3.3.2011, 0.1, kesken.	x	x				
			AA030	KU2		Hyvönen, 3.3.2011, 0.1, kesken.	x					
			ZA002			Domnina, 3.3.2011, 0.1, kesken.	x					
			ZA002			Hyvönen, 3.3.2011, 0.1, kesken.	x					

LIITE 4 KUVAUKSET-LOMAKE

ICARE form: kuvaus Osa	Nimi	Referenssi	Konteksti	Kuvaus	Muut liittyvät säännöt	Muut liittyvät aktiviteetit	Muut liittyvät entiteetit	Muut liittyvät kuvaukset	Tiedon alkuperä	Kaavakkeen hallinta[Tekijä, pvm, versio, status]
	Mekaniikkasuunnittelun piirustukset	KU1	Yleinen	Mekaniikkasuunnittelun yleiset piirustukset ovat: Main Dimensions , Rotor Drawing, Cross Sectional Drawing, Rotor Removal, Principle of Emergency Cooling, Transport and	R8, R9		AA020			
	Sähkösuunnittelun piirustukset	KU2	Yleinen	Sähkösuunnittelun yleiset piirustukset ovat: Exciter Rotor with Diode Bridge, Layout of Connection, Main Terminal Box B1, Neutral Terminal Box B2, Auxiliary Terminal Box B3A, , Main Connection Diagram	R10		AA030			