TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO JUNTTILA

SUUNNITTELUAUTOMAATION IMPLEMENTOINTI CAD-JÄRJESTELMÄÄN

Diplomityö

Tarkastaja: professori Asko Ellman
Tarkastaja ja aihe hyväksytty
Teknisten tiedeiden
tiedekuntaneuvoston kokouksessa
5. maaliskuuta 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Konetekniikan koulutusohjelma
Junttila, Mikko: Suunnitteluautomaation implementointi CAD-järjestelmään
Diplomityö, 64 sivua, 36 liitesivua
Maaliskuu 2014
Pääaine: Koneiden ja järjestelmien suunnittelu
Tarkastaja: Professori Asko Ellman
Avainsanat: Solid Edge, CAD, Mekaniikkasuunnittelu, Suunnitteluautomaatio, Tuotekonfiguraattori, Myyntikonfiguraattori, Makro, Add-In, Apuohjelma, Liitännäinen

Tämä diplomityö tutkii mekaniikkasuunnitteluohjelman automatisointia. CAD -ohjelmien tuottajat voivat automatisoida ohjelmiaan vain tiettyyn rajaan asti, sillä jokaisen yrityksen tarpeet ja käytännöt, samoin kuin myös suunnittelijoiden taidot, poikkeavat toisistaan radikaalisti. Useissa yrityksissä tietokoneavusteisen suunnittelun pullonkaula kohdistuu yhä arkiseen mallien ja piirustusten muokkaamiseen. Vaikka työn pääpaino keskittyy kohdeohjelmana toimivan Siemens PLM Softwaren Solid Edge CAD -ohjelmistoon, on tässä opinnäytteessä esiteltävä teoria suurelta osin paikkaansa pitävää myös muiden mekaniikkasuunnitteluohjelmien automatisointiin.

Diplomityön kohdistuminen Solid Edgen yhteyteen oli luonteva ratkaisu työskenneltyäni ohjelman parissa CAD-asiantuntijana. Siemensin tarjonta ohjelmaan sekä Teamcenter-integraatioon oli tuttu, mutta Solid Edgen yhteyteen ohjelmoitavat apuohjelmat olivat niin allekirjoittaneelle kuin myös ohjelman maahantuojalle jossain määrin rajalliset. Tässä diplomityössä esiteltävät sovellukset Solid Edgen automatisoinniksi on luotu käyttäen Visual Basic .NET ohjelmointikieltä. Teoria ottaa kantaa myös yleisesti käytettyihin C++ ja C# ohjelmointikieliin.

Osa tämän työn tutkimustuloksista perustuu omiin kokemuksiini asiantuntijatehtävissä, toinen osa kirjalliseen tutkimukseen, joka painottui tehtyihin tutkimuksiin ja olemassa oleviin metodeihin, sekä viimeisenä omatoimiseen ohjelmoinnin kautta tehtävään opiskeluun. Työn myötä opettelin uuden ohjelmointikielen sekä tutustuin ensi kertaa Solid Edgen ohjelmointirajapintaan.

Opinnäytetyön tavoitteet saavutettiin onnistuneesti tutkimuksen myötä. Diplomityö esittelee kirjallisuustutkielman pohjalta suunnitteluautomaation useimmin käytetyt menetelmät, jossa keskitytään pitkälti tuotteen konfigurointiin.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Mechanical Engineering
Junttila, Mikko: Implementation of design automation to a CAD system
Master of Science Thesis, 64 pages, 36 Appendix pages
March 2013
Major: Design of machines and systems
Examiner: Professor Asko Ellman
Keywords: Solid Edge, CAD, Mechanical Desing, Design automation, Product configuration, Sales configurator, Macro, Add-In

This master's thesis studies the automation of a mechanical design program. The producers of CAD programs can automate their programs only to a certain level. This is because every company has their own set of needs and uses for the programs as also the skills of their designers vary. On many companies the bottleneck of the design work is on editing used models and drafts. Although the main focus of this thesis is on Siemens PLM Softwares Solid Edge CAD program, the theory of it, applies also to automation of other CAD programs.

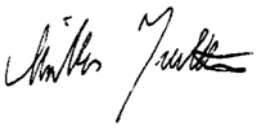
After working as a CAD-specialist, it appeared natural to focus my thesis on Solid Edge. Both I and the local VAR, allready had broad knowledge on both the program and Teamcenter integration. Still we had somewhat limited knowledge on programming on Solid Edge. The applications produced for this thesis, have been produced using Visual Basic .NET programming languge. The theory also takes notes on C++ and C#.

First part of this thesis is based on own experiences working as a CAD-specialist. Second part is a literature study where I read the current studies and known methods on design automation. For the third part I learned how to program using Visual Basic .NET and got to know the Solid Edge application programming interface.

The goals set for this thesis were met. This study presents the most used methods on how to automate a design system and takes special notice on configuration.

ALKUSANAT

Tämä on Tampereen teknillisen yliopiston diplomi-insinöörin tutkintojen opinnäytetyö, eli diplomityö ’koneiden ja järjestelmien suunnittelu’ opintokokonaisuuteen. Työ keskittyy Siemens PLM Softwaren Solid Edge CAD -tuotteen automatisointiin ohjelmoinnin keinoin. Työn tavoitteena on tutkia miten ja missä laajuudessa Solid Edge -ohjelmaa on mahdollista automatisoida ohjelmoinnin avulla, sekä esitellä tulokset esimerkin omaisesti. Työn ideoinnissa auttoi ohjelmiston jälleenmyyjä IDEAL PLM, mutta työ tehtiin omakustanteisesti. Työn ohjauksesta kiitos Tampereen teknillisen yliopiston professori Asko Ellmanille. Erityismainintana haluan kiittää Tanja Heikkistä avusta ja tuesta.

20.3.2014: 

Mikko Junttila

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

ActiveX	Toinen nimi Microsoftin Windows -käyttöjärjestelmässä käytetylle COM-tekniikalle.
Add-In	Apuohjelma tai liitännäinen (tietotekniikka). Isäntäsovellus tarjoaa palveluita, joita liitännäiset voivat käyttää.
Alternate Assemblies	Solid Edgen työkalu tuoteperheen luomiseen.
API	Application Programming Interface - Ohjelmointirajapinta, jonka avulla eri ohjelmat vaihtavat informaatiota.
App.	Application - Ohjelmistosovellus.
ATO	Assembly to Order - Valmistuksen ohjausprosessi, jossa tuote kokoonpannaan tilauskohtaisesti.
Attribuutti	Ominaisuus.
BASIC	Beginner's All-purpose Symbolic Instruction Code - Aloittelijan yleissymbolinen ohjekoodi. Ohjelmointikieli.
CAD	Computer Aided Design – Tietokoneavusteinen suunnittelu (piirtäminen & mallinnus).
CAE	Computer Aided Engineering – Tietokoneavusteinen suunnittelu (suunnittelu & simulointi).
Callout	Solid Edgessä käytettävä kutsu komento, jonka avulla osa- ja kokoonpanotiedostojen tietoja saadaan esitettyä.
CAM	Computer Aided Manufacturing – Tietokoneavusteinen valmistus.
Case Study	Tutkimus, joka kohdistuu yhteen yksikköön, ja jossa käytetään useita tiedonkeruutapoja sekä -lähteitä tilanteen analysoinnissa ja kehittämisessä.
CIM	Computer-Integrated Manufacturing - Tietokoneteknologian käyttö kaikessa tuotannon informaation ja operaatioiden hallinnassa.
Class Library	Microsoft Visual Basic 2008 DLL projektiformaatti.
COM	Component Object Model - Microsoftin binary-interface standardi. COM-komponentit ovat uudelleen käytettäviä ohjelmistokomponentteja.
Console Application	Microsoft Visual Basic 2008 makro projektiformaatti.
cPDM	Collaborative Product Data Management - Työkalu, joka helpottaa kaikkien prosessien, sovellusten ja information hallinnassa, suunnittelun, valmistuksen ja tuen myötä läpi tuotteen eliniän.
C++	Eräs maailman käytetyimmistä ohjelmointikielistä, joka on kehittynyt C -kielen pohjalta.
C#	C Sharp. Mikrosoftin .NET -konseptia varten kehittämä ohjelmointikieli.

Dimensio	Ulottuvuus, mitta, koko, suuruus.
DLL	Dynamically linked library. Ohjelmistokomponentti, jota toinen ohjelma tai komponentti voi suorittaa. Vrt. Makro.
DOS	Disk Operating System - levynkäyttöjärjestelmä.
Draft	2D piirustus tai piirros.
ETO	Engineer to Order - Valmistuksen ohjausprosessi, jossa tuote suunnitellaan tilauskohtaisesti.
Family Of Parts	Solid Edgen aputyökalu osaperheen luontiin ja hallinnointiin.
Feature	CAD kappaleetta, kokoonpanoa tai piirrosta muokkaava ominaisuus.
FEM	Finite Element Method - Elementtimenetelmä.
Form	Visual Studio 2008:ssa formi on suunnittelun aikainen näkymä ”ikkunasta”, joka näytetään käyttäjälle.
GUI	Graphic User Interface - Visuaalinen käyttöliittymä.
GUID	Globally Unique Identifier - Globaalisti uniikki tunniste.
IDE	Integrated Development Environment - Ohjelmointiympäristö on ohjelma tai joukko ohjelmia, jolla ohjelmoija suunnittelee ja toteuttaa ohjelmistoa.
ISO	Kansainvälisiä standardeja tuottava International Organization for Standardization.
Komentosarja	Tietokoneelle annettava tekstimuotoinen joukko käskyjä.
Konfiguraattori	Järjestelmä, jonka avulla tuotetaan konfiguraatioita.
Konfiguraatio	Saman tuotteen eri muunnos, eli tuotevariaatio.
Käyttöliittymä	User Interface (UI) - Rajapinta, jonka kautta käyttäjä operoi laitetta tai ohjelmistoa.
Logiikka	Tietojenkäsittelyä päättelyn, aksioomien ja matemaattisten sääntöjen avulla.
Makro	Tallennettu työvaiheiden ja komentojen sarja, jota suorittaessa ohjelmisto suorittaa kyseiset toiminnot itsenäisesti.
MCAD	Microsoft Certified Application Developer - Microsoftin sertifioima ohjelmistokehittäjä.
Mekaniikkasuunnittelu	Mekaanisten osien ja kokoonpanojen tietokoneavusteista suunnittelua mallinnuksen ja simuloinnin keinoin.
Moduuli	Tuotteen itsenäinen osa, joka yhdistyy rajapinnoistaan toisiin moduuleihin, muodostaen näin erilaisia konfiguraatioita.
MSDN	Microsoft Structured Storage API. Microsoftin standardoima tapa säilöä hierarkista tietoa tiedostossa.
MTO	Make to Order - Valmistuksen ohjausprosessi, jossa tuote valmistetaan tilauskohtaisesti.

MTS	Make to stock - Valmistuksen ohjausprosessi, jossa tuote valmistetaan varastoon.
Myyntikonfiguraattori	Myynnin apuväline, jonka tarkoitus on ohjata asiakas tuotteen konfigurointiprosessin läpi.
Java	Ohjelmointikieli.
Objekti	Olio.
Parametri (CAD)	Mitta- tai relaatiotieto.
Parametri (tietotekniikka)	Ohjelmalle tai komentojonolle käynnistyksen yhteydessä välitettävä, sekä ohjelmoinnisssa funktiolle tai käskylle välitettävä tieto.
PDM	Product Data Management - Tuotetiedonhallinta.
PLM	Product Lifetime Management - Tuotteen elinkaaren hallinta.
Property	Ominaisuus.
Quick Access	Pikavalikko. Sijaitsee useissa sovelluksissa ylhäällä.
Solid Edge	Tietokoneavusteinen suunnittelu, eli CAD, -ohjelmisto (lyhyesti SE).
Spy for Solid Edge	Sovellus, jonka tarjoaa tietoa Solid Edge API:sta.
Standard Parts	Solid Edgen vakio-osakirjasto.
STO	Ship to Order - Valmistuksen ohjausprosessi, jossa tuote toimitetaan tilauskohtaisesti.
Syncronous Technology	Solid Edgessä osan historiatiedosta riippumaton mallinnustekniikka, joka mahdollistaa ”perinteikästä” mallinnusta nopeamman osien uudelleen muokkaamisen.
Teamcenter	Tuotteen elinkaarenhallinta, eli PLM, -järjestelmä (lyhyesti TC).
Tuotekonfiguraattori	Räätälöidyn rakenteen luova ohjelma.
Tyyppikirjasto	Type Library. Sisältää ohjelman toiminnallisen koodin.
Oliomalli	Object Model. Objekti kokoelma, jonka avulla ohjelma voi käsitellä tietoa.
UI	User Interface - Käyttöliittymä.
VAR	Value-Added Reseller - Arvoa lisäävä jälleenmyyjä.
Variables	Solid Edgen taulukkotyökalu dimensioiden hallintaan.
Visual Basic .NET	Microsoftin kehittämä yleiskäyttöinen ohjelmointikieli (lyhyesti VB .NET) .
Windows Forms Application	Microsoft Visual Basic 2008 Windows aplikaatio projektiformaatti.
.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto.
.exe	Executable file. Ohjelmistotiedostomuoto: ohjelma.

1. JOHDANTO

Kolmiuloitteiset (3D) parametriset mekaniikkasuunnitteluohjelmat tarjoavat mahdollisuuden mallintaa kappaleista ja kokoonpanoista tarkan visuaalisen mallin, jonka avulla rakenteen toimivuutta voidaan tarkkailla. Myös muu tekninen- ja tietotekninen kehitys tukee tarkan mallin luomista. Esimerkiksi prototyyppien luominen 3D-tulostamalla on yleistynyt kovaa vauhtia, ja rakenteiden kestävyyden tarkastelu voidaan suorittaa mallia hyväksi käyttäen lujuuslaskentaan pohjautuvilla simulointiohjelmilla (Computer Aided Engineering, CAE). Mekaniikkasuunnittelijat käyttävät kuitenkin suuren osan työajastaan rutiininomaiseen mallien ja piirrosten muokkaamiseen. Työskennellessäni CAD-asiantuntijana, asiakkaat tiedustelivat yhä useammin tarjoamamme *tietokoneavusteisen suunnitteluohjelman (Computer Aided Design, CAD)* automatisointimahdollisuuksia, usein tarkoittaen lähinnä *tuotekonfiguraattorien* luomisen mahdollisuutta. Näistä syistä valitsin opinnäytteeni työn aiheeksi suunnitteluautomaation implementointi CAD-järjestelmään. Opinnäytetyön tavoitteena on pitkällä tähtäimellä ylläpitää suomalaisen teollisuuden kilpailukykyä, ja kehittää sitä yhä kovenevassa globaalissa kilpailussa.

Kuva 1.1. Opiskelukäyttöön lisensoitu Solid Edge ST6

Esittäessäni työn kannalta tärkeän termin, nimityksen tai nimen ensimmäistä kertaa tässä työssä, esitän sen kursivoidulla tekstillä, painottaakseni sanan merkitystä. Tällaisessa tapauksessa sana löytyy opinnäytteen alkuosiossa löytyvästä termistöstä. Käytän tekstin ohessa jonkin verran koodi esimerkkejä, jolloin erottaakseni koodin normaalista tekstistä, käytän Times New Roman fontista poikkevaa fonttia.

Opinnäytteen teon aikaan suunnitteluautomaatiosta löytyy paljon ”yleispätevää” kirjallista aineistoa, muttei juurikaan opinnäytteeni isäntäohjelmaa: *Solid Edge*ä (kuva 1.1.) koskevaa tietoa. Vastaavia ohjelmistokehitysprojekteja löytyy kyllä Solid Edgen kilpaileville tuotteille. Tutkimuksen alussa tutustuttiin esimerkiksi ammattikorkeakoulujen lopputöitä kokoavaan theseus.fi -palveluun [24], jonka kautta löytyi samankaltaisia lopputöitä. Näihin tutustuttiin perusteellisesti sekä niiden yhtäläisyyksien, vahvuuksien, että myös muiden kirjallisuuslähteiden löytämiseksi. Työn ohessa otetaan kantaa suunnittelun automatisoinnin ja *tuotekonfiguraattoreiden* lisäksi *myyntikonfiguraattoreihin*, sekä teorioihin, joita tilauskohtainen toiminta mahdollistaa. Myyntikonfiguraattori on suhteellisen uusi tapa markkinoida tuotetta. Sovelluksen avulla eri tuotevariaatiot esitetään visualisointia käyttäen. Opinnäytteen esimerkkeihin ei sisällytetty myyntikonfiguraattoria, sillä sen toteutus ei ole Solid Edgestä riippuvainen, mutta asia käsitellään teoreettisesti laajalti opinnäytteen aikana.

Kuva 1.2. Siemens PLM Solid Edge Community [26].

Aihe tuntui alkuun haastavalta, sillä opinnäytteen tekijän kokemus ohjelmoinnista on lähes olematon. Tutkittuani aihetta kiinnostuin tästä koko ajan enemmän ja toivoin, että olisin voinut tutkia asiaa täysipäiväisesti. Näin ei kuitenkaan ollut, vaan työ tehtiin alusta saakka omalla ajalla töiden päätyttyä, ja näin ollen jo tehtyjä asioita jouduttiin usein kertaamaan muistin virkistämiseksi. Täysin tyhjästä ohjelmoinnin opettelussa ei täytynyt lähteä, sillä Solid Edge käyttäjät, jotka ovat automatisoineet omaa suunnitteluaan, ovat osin jakaneet koodinsa kaikkien käytettäväksi. Tällainen sivusto on esimerkiksi solidedge.codeplex.com [25]. Lisäksi Solid Edgen kehittäjä *Siemens PLM Software* kannustaa käyttäjiään kehittämään ohjelmistoa ja tarjoaa ohjelmointioppaan Solid Edge *ohjelmointirajapinnasta (Aplication Programming Interface, API)* [21], sekä kuvan 1.2 Siemensin oman Solid Edge -yhteisö sivuston, jossa keskustellaan muun muassa suunnittelun automatisoinnista. Tämän lisäksi Solid Edgen *Help*-osio kattaa myös ohjelmointitiedoston, josta löytyy *Visual Basic .NET* -ohjelmointikielelle käytettävät objektit, muuttujat ja komennot. Ajankäytön rajallisuuden vuoksi aihepiirin ulkopuolelle rajattiin Siemens PLM Softwaren tuotteen elinkaarenhallintajärjestelmä Teamcenter.

2. TAUSTATIETOA

Tässä opinnäytetyössä tutkitaan Siemens PLM Softwaren kehittämän Solid Edge -mekaniikkasuunnitteluohjelmiston automatisointia. Solid Edge on yksi kahdesta Siemensin kehittämästä tietokoneavusteisesta suunnitteluohjelmistoista. Toinen CAD-ohjelmisto on nimeltään NX, mutta sen ollessa täysin erillinen tuote Solid Edgen rinnalla, ei tämä opinnäytetyö ota kantaa sen automatisointiin.

Nykyaikaiset CAD-ohjelmistot voidaan jakaa kolmeen sekmenttiin niiden tarjoamien ominaisuuksien mukaisesti: ilmaisohjelmat, ”mainstream” ja ”high-end”. Ilmaisohjelmat ovat työkaluiltaan usein puutteelliset, ja näin ollen niiden tuoma etu moderniin ja tehokkaaseen tietokoneavusteiseen suunnitteluun on rajallinen. Esimerkkinä tästä, Siemens tarjoaa Solid Edge -ohjelmistosta ilmaisen 2D-version, joka sisältää tarvittavat työkalut perinteiseen *tietokoneavusteiseen piirtämiseen (Computer Aided Drafting, CAD)*. Solid Edge, normaalisti lisensoitavassa muodossaan, sisältää kaikki 2D- ja 3D-suunnitteluun vaadittavat peruselementit. Näin ollen se kilpailee mainstream-kategoriassa SolidWorksin, Vertexin ja Inventorin kaltaisten ohjelmistojen kanssa. Tämän segmentin asiakaskunta on suurin, mutta yksittäisten käyttäjien määrä ei kuitenkaan yllä high-end järjestelmien käyttäjämäärien tasolle. Mainstream-ohjelmat ovat usein high-end ohjelmia helppokäyttöisempiä, ja näin ollen asiakaskunta koostuu pääsääntöisesti pienistä ja keskisuurista yrityksistä. High-end ohjelmistoja käyttävät usein suuret yritykset, joille järjestelmän tehokkuus on tärkeää. Siemensin omistama NX edustaa CAD-ohjelmistojen huippua, ja näin ollen kilpailee high-end luokassa. Tässä luokassa samassa käyttöliittymässä toimii usein suunnitteluominaisuuksien lisäksi myös muita toimintoja, kuten tietokoneavusteinen valmistus (Computer Aided Manufacturing, CAM). NX:n kanssa suunnittelutyökalujen laajuudessa tai laadussa pystyy kilpailemaan ainostaan Dassault Systemesin tarjoama Catia.

IDEAL PLM on Siemens PLM Softwaren yhteistyökumppani, jälleenmyyjä ja maahantuoja tämän kehittämille tuotteille. Teknisen suunnittelun kolmiuloitteisuus on ollut nykyaikaa IDEALin asiakkaille pitkään, ja nykyisenä trendinä on huomattu asiakkaiden kasvava kiinnostus suunnittelun automatisointiin. Näin suunnittelijat vapautetaan toistoa vaativilta rutiineilta arvoa tuottavaan suunnittelutyöhön. Aiemmat suomalaiset Solid Edge -suunnitteluautomaatioprojektit on hoidettu IDEAL PLM:n yhteistyökumppaneiden toimesta, mutta yritys haluaa panostaa tulevaisuudessa sisäiseen osaamiseen, ja näin ollen alkaa tuottaa itse projektit kokonaisuudessaan, tarkoituksenaan parantaa asiakkailleen tarjoamaa palvelua. Seuraavissa luvuissa kerron tarkemmin Siemens PLM Softwaresta, Solid Edgestä sekä IDEAL PLM:stä.

2.1 Siemens PLM Software

Siemens PLM Software on markkinoiden johtava tuotteen elinkaaren hallintajärjestelmien (Product Lifecycle Management, PLM) tuottaja. Siemens on globaali teollisuusorganisaatio, ja kuvasta 2.1 voidaan havaita, että Siemens tuottaa palveluitaan ja tuotteitaan hyvin laaja-alaisesti sekä teollisuuteen että yksityisille kuluttajille. Siemensin organisaation segmentit voidaan lukea kuvan vasemmasta laidasta, ja kunkin rivin kuviin on kirjattu divisioonien nimet. Teollisuusautomaatio divisioonan sisäisiin yksiköihin kuuluvat Siemens Product Lifecycle Management Softwaren lisäksi Automation Systems, Operator Control and Monitoring Systems, Identification Systems, Industrial Communications, Industrial Controls, Manufacturing Execution System, PC-based Automation, Process Control System, Sensor Systems ja Power Supplies -yksiköt. [1]

Kuva 2.1. Siemensin divisioonat.

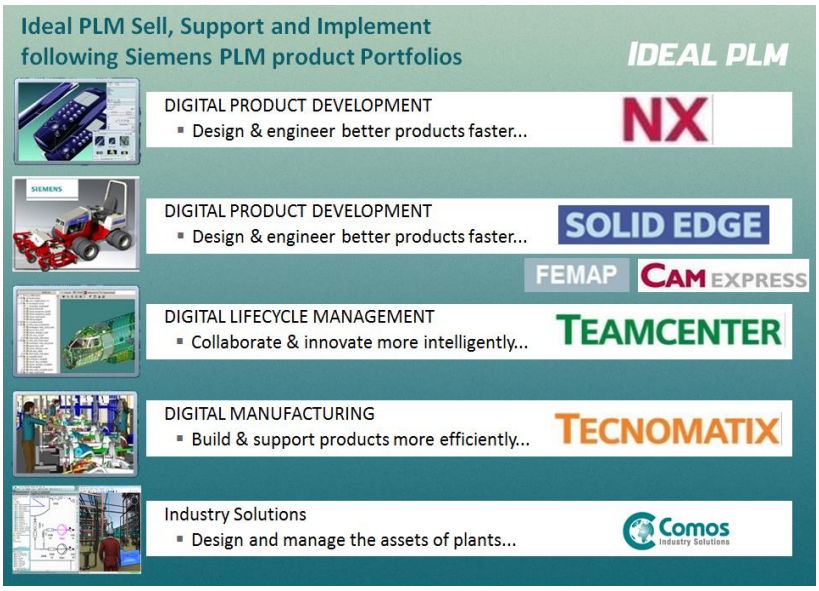
Product Lifecycle Management Software -liikeyksikkö kehittää teollisuuden tarpeisiin tuotetiedon hallintatyökaluja (Product Data Management, PDM), tietokoneavusteisia suunnittelu-, valmistus- ja simulointiohjelmistoja (CAD, CAM, CAE). Tuoteportfolioon kuuluu kattava valikoima ohjelmistoja, joiden tarkoitus on mahdollistaa tuoteideoiden kehittäminen toimiviksi tuotteiksi, sekä hallinnoida tuotannon kaikissa vaiheissa syntynyttä dataa. Ohjelmistojen globaalista johtoasemasta esimerkkinä tieto, että maailman kahdestakymmenestä suurimmasta autovalmistajasta kaikki käyttävät Siemens PLM Softwaren -sovelluksia. Yhteensä lisensoituja käyttäjiä ohjelmistoilla on yli seitsemän miljoonaa ympäri maailmaa (2013). [2]

2.2 IDEAL PLM

Opinnäytetyön aiheen ideoinnissa auttoi Siemens PLM Softwaren yhteistyökumppani, aputoiminimellään tunnettu, IDEAL PLM (virallinen toiminimi: Ideal Product Data Oy). Yritys on tarjonnut vuodesta 1992 lähtien Siemensin PLM -ohjelmistoja, näihin liittyvää tukipalvelua ja koulutusta. Yritys panostaa jatkuvasti palveluiden parantamiseen ja sisäiseen osaamiseen. Mahdollisuus tämän opinnäytteen tekemiseen syntyi, kun asiakkailta saadun palautteen myötä suunnittelunautomaatio-projekteja haluttiin kehittää asiakaslähtöisemmäksi. Allekirjoittanut toimi IDEAL PLM:n palveluksessa vuodesta 2012 lähtien, aluksi CAD-asiantuntijana ja vuoden 2013 loppuun myyntipäällikkönä.

Kuva 2.2.1. Siemens PLM Solution Partner [30].

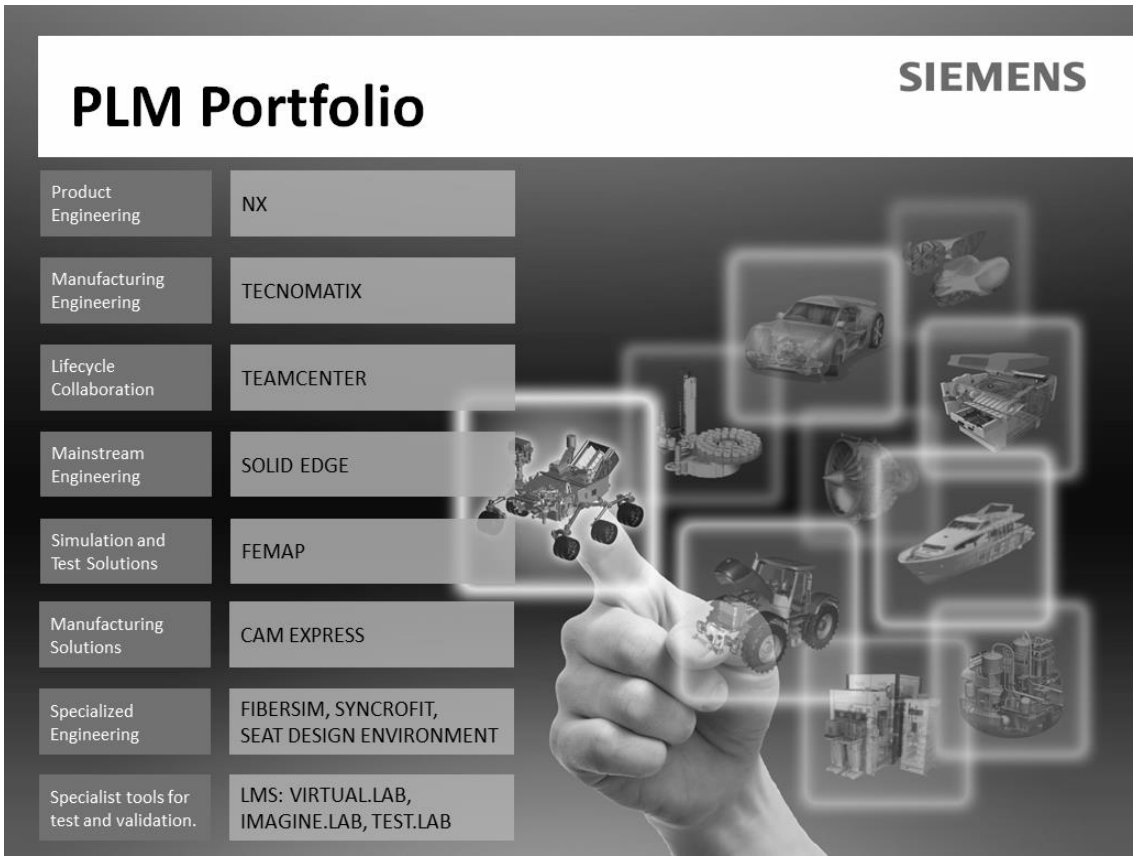
Onnistuneen yhteistyökumppanuuden seurauksena, IDEAL PLM on Siemensin pyynnöstä vuodesta 2012 lähtien toimittanut palveluitaan myös Venäjälle. Venäjän markkinoilla IDEAL toimii yhtenä Siemensin tuotteiden arvoa kasvattavana jälleenmyyjänä (Value-Added Reseller, VAR) useiden toimijoiden joukossa, kun Suomen markkinoilla yrityksellä on yksinoikeus Siemensin PLM Software -tuotteiden jälleenmyyntiin. Kuten kuvasta 2.2.2 voi havaita, poiketen Siemens PLM Softwaren ohjelmistoista, edustaa IDEAL PLM myös Siemensin prosessi- ja automaatiosuunnitteluohjelmisto Comosta. Kyseinen lisäys IDEAL PLM:n ohjelmistotarjontaan tuli Siemensin pyynnöstä vuonna 2013.



Kuva 2.2.2. IDEAL PLM tuoteportfolio [3].

Siemensin usein palkitsema dynaaminen yritys työllistää diplomityön kirjoitushetkellä noin 85 PLM- ja CAD-asiantuntijaa, PLM-arkkitehtiä ja muuta henkilöstöä. Täyden palvelun tuotteen elinkaaren hallintajärjestelmiä tarjoavalla talolla on laaja tietämys eri toimialoilta, ja henkilöstöllä laaja PLM-, prosessi-, ohjelmointi-, PLM-arkkitehtuuri sekä projektinjohtokokemus. [3]

Kuten kuvasta 2.2.3 voidaan havaita, Siemens tarjoaa myös tuotteita, kuten Seat Design Environment, jotka eivät ole IDEAL PLM:n edustamia. Näitä ohjelmistoja vaativa hyvin erikoistunut toimiala sisältää hyvin vähän tai ei lainkaan toimijoita Suomessa. IDEAL PLM:n edustamien tuotteiden pääpaino on sekä suunnittelu-, valmistus- että tuotetiedon hallinta ohjelmistoissa.



Kuva 2.2.3. Siemens PLM Software tuoteportfolio [30].

Tämän opinnäytetyön kannalta merkitsevät ohjelmat ovat CAD-ohjelmisto Solid Edge, sekä jatkokehitystä silmällä pitäen tuotteen elinkaaren hallinta järjestelmä Teamcenter. Seuraavassa luvussa kerron lisää Solid Edge -ohjelmistosta. Teamcenteristä voit lukea lisää opinnäytteen jatkotoimista kertovassa luvusta.

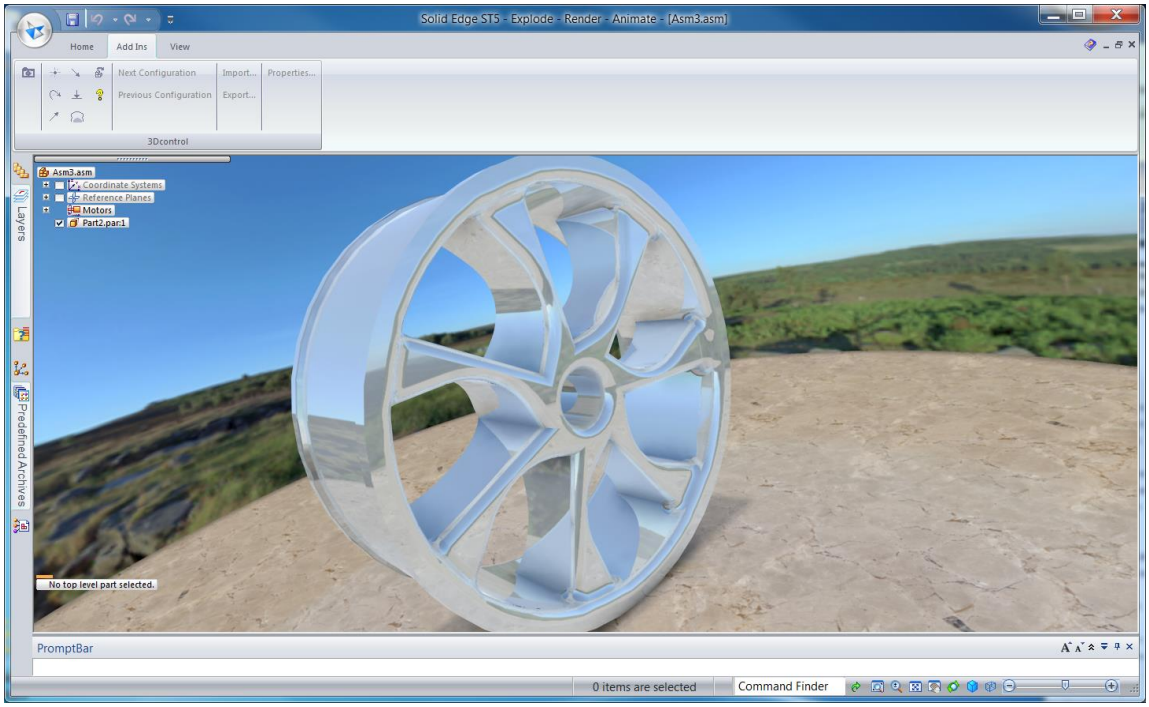
2.3 Solid Edge

Solid Edge on yksi maailman tunnetuimmista mainstream CAD-ohjelmista. Ohjelmistolla on tällä hetkellä (2013) noin 450 000 käyttäjää. Intergraph Corporation julkaisi ACIS mallinnusytimen päälle rakennetun ensimmäisen version ohjelmasta

vuonna 1995. Vuoteen 1998 mennessä, Solid Edgen ollessa UGS Corp. omistuksessa, sen pohjalla toimiva ydin vaihdettiin saman yrityksen omistamaan Parasolid mallinnusytimeen. Siemens PLM Softwaren omistukseen Solid Edge siirtyi vuonna 2007, kun Siemens AG osti UGS Corp.:in osaksi itseään, näin saaden omistuksen myös sen omistamiin ohjelmistoihin, kuten Solid Edge ja NX, sekä sen ohjelmistokomponentteihin, kuten Parasolid mallinnusytimeen ja D-Cubed relaatioiden hallintakirjastoihin.

Vaikka Siemens PLM Software on varmistanut kilpailuedun aggressiivisten teknologia- ja yritysostojen kautta, tarjoaa se alalle oleellisia komponentteja myös kilpailijoilleen. Tästä esimerkkinä Parasolid, joka on tänä päivänä maailman käytetyin geometrinen mallinnusydin. Siemens säilyttää itsellään kilpailuedun pidättäytymällä myymästä komponenteista viimeisintä versiota. Näin ollen esimerkiksi Solid Edgen ja NX:n sisältämä, usein palkittu suunnitteluteknologian suunnannäyttäjä, suoramallinnustekniikka Synchronous Technology, ei sisälly kauppaan.

Solid Edge kuuluu Siemens PLM Softwaren Velocity -sarjaan. Sarjan muita tuotteita ovat: lujuuslaskentaohjelmisto FEMAP, tuotteen elikaaren hallintajärjestelmä Teamcenter Express, tietokoneavusteisen valmistuksen ohjausohjelmisto: CAM Express sekä CAD-formaatista riippumaton 3D-mallin muokkaus ohjelma: 3Dsync. Solid Edgen mukana on saatavissa myös tuotetiedon hallintaohjelma Solid Edge SP. Velocity sarjan muut tuotteet on kehitetty Solid Edge CAD-ohjelmiston jatkeeksi, eikä tämä opinnäyte ota niiden automatisointiin kantaa.



Kuva 2.3. Solid Edge ST5 UI.

Kuva 2.3 on kuvakaappaus Solid Edgen *käyttöliittymästä (User Interface, UI)*. Käyttöliittymästä on huomattavissa, että sen yläpalkki on rakennettu samaan tapaan

Microsoft Office -tuotteiden kanssa. Valintanauhan (Ribbon) käyttö Solid Edgen käyttöliittymässä on seurausta Siemensin ja Microsoftin tiiviistä yhteistyöstä. Yhteistyön voi havaita konkreettisesti myös Solid Edgen työkaluissa, sekä Teamcenterin ja Office tuotteiden yhteiskäyttöominaisuuksissa. Kuvasta voidaan havaita myös, että kuvakaappauksen otoshetkellä ja diplomityön aloitushetkellä, ei *Add-Ins* välilehti sisällä 3D-hiiren (kuva 2.4.1) ohjainten lisäksi muita apuohjelmia.

2.3.1 Lisensointi

Solid Edgen suunnittelutyökalujen määrä on riippuvainen tuotteen lisenssistä. Suomessa linsensoitavalle ohjelmistolle on lisenssitasoja tarjolla kolme: Foundation, Classic ja Premium. Foundation lisenssi sisältää kaikki moderniin 2D-piirtämiseen ja 3D-mallintamiseen vaadittavat työkalut. Käytettäessä alemman tason lisenssejä, on lisätyökaluja mahdollista saada käyttöönsä myös yksittäisinä lisämoduuleina. Premium-linsenssi tarjoaa suunnittelijan käyttöön kaikki Solid Edgen mallinnustyökalut, jotka Siemens on siihen luonut. Tämän opinnäytetyön tutkimuksissa käytetään osittain tätä korkeinta lisenssiastetta sekä osittain opiskelijoille ilmaista opiskelijalisenssiä. Opiskelijalisenssin puutteet ovat pieniä, kuten esimerkiksi kappaleiden tai niiden pintojen väriä ei voi vaihtaa. Suurimpana rajoitteena opiskelijalisenssissä on, ettei sillä lisensoidulla ohjelmistolla tuotettuja tiedostoja saa auki kaupallisilla ohjelmistoversioilla. Tämän lisäksi hyväksyttävä sopimus kieltää opiskelijaversiolla tuotettujen mallien kaupallisen käytön. Opinnäytetyön tutkimuksissa käytettävän ohjelmistoversion nimi on Solid Edge ST6, eikä tutkimustuloksia voi suoraan yleistää muihin ohjelmistoversioihin. Ohjelmistoversio on kuudes lanseeraus, joka sisältää Synchronous Tehcnologyn, tästä lyhenne ST6. Solid Edge ohjelmistosta lanseerataan vuosittain uusi versio. Synchronous -suoramallinnustekniikkaa edeltäneet versiot on nimetty: ”vXX”, jossa v on lyhenne sanasta version ja XX on järjestysnumero.

Solid Edgen yhteyteen on saatavilla Siemens PLM Softwaren yhteistyökumppaneiden kehittämiä applikaatioita. Nämä ulkoisten ohjelmistokehittäjien tarjoamat lisätyökalut käyttävät hyväksi Solid Edgen suunnitteluominaisuuksia, sekä ohjelmointirajapintaa, ja näin ollen laajentavat ohjelmiston käyttömahdollisuuksia. Tarjonnasta lisää luvussa 3.4 Ulkoiset ohjelmistokehittäjät.

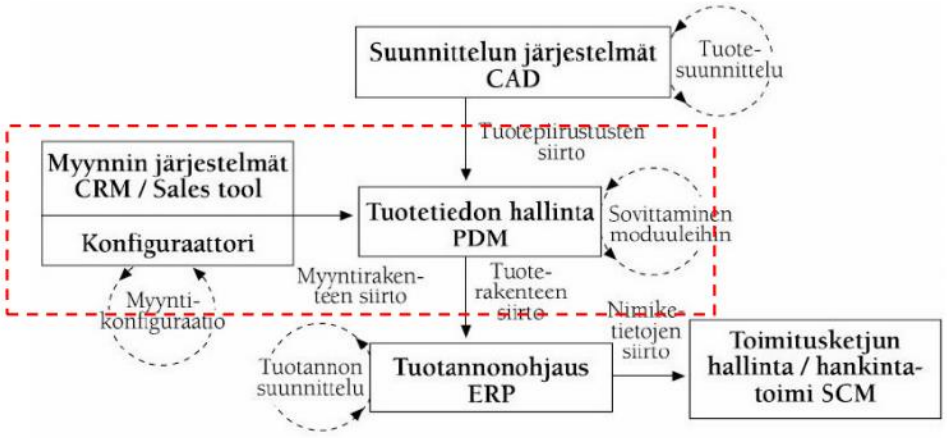
2.4 Suunnittelujärjestelmä

Alkupyrkimyksenä tietokoneiden käyttöönotossa suunnittelutyöhön, oli nopeuttaa suunnittelutyötä ja parantaa samanaikaisesti suunnittelun laatua. Kaksiuloitteisten (2D) ohjelmistojen tavoitteena on tuottaa piirustuksia nopeammin kuin tuohon astisilla keinoilla, eli lähinnä piirustuslaudalla työskennellen. Kolmiuloitteisilla (3D) ohjelmilla tarkoituksena on mallintaa kappaleesta todenmukainen malli, josta otetuilla projektioilla saadaan luotua yksinkertaisesti 2D-piirustukset lomakepohjalle. Lisäämällä piirroksiin mitoitus, saadaan kappaleista valmistuskuvat tuotantoon sekä kokoonpanokuvat. [4]

Kuva 2.4.1. 3Dconnexion 3D-hiiret [5].

Suunnittelijan työkaluina nykyaikaisessa suunnitteluympäristössä toimivat joko tehokas kannettava- tai pöytätietokone, joka on kytketty kookkaaseen näyttöön. Tietokone pyörittää tavallisimmin parametristä 3D-suunnitteluohjelmistoa, sillä 2D-ohjelmien käyttö vähenee koko ajan. Suunnittelijan komentoja ohjelmistolle välittävät näppäimistö, osoitinlaite (hiiri) ja suunnittelijoiden suosima 3D-hiiri, joka mahdollistaa näkymien helpon liikuttamisen kappaleen tai kokoonpanon ympärillä. Kuvassa 2.4.1 on esitetty 3Dconnection nimisen yrityksen 3D-hiiret.

On tavallista tulostaa halutut työkuvat paperille ja toimittaa nämä tuotantoon, mutta teknologian kehitys on mahdollistanut myös suoran tiedonvälityksen suunnitteluohjelmistoista tietokoneavusteisiin valmistusohjelmiin, ja tästä aina työstökoneille sekä kokoonpanoasemille asti (Computer-Integrated Manufacturing, CIM). Tämä tietokoneavusteinen suunnittelu- ja valmistusketju, joka on hahmotettu kuvassa 2.4.2, on pitkälle viedyssä järjestelmässä yhdistetty sekä toiminnanohjausjärjestelmään (Enterprice Resource Planning, ERP) että PLM-järjestelmään, jotka välittävät dataa hankintaketjussa molempiin suuntiin, sekä yrityksen sisäisesti esimerkiksi markkinoinnin ja myynnin tarpeisiin. Kaikkea tätä pyörittää parhaillaan serverifarmi, johon käyttäjillä ympäri maailman on toisistaan erillään tai ryhmittäin hallinnoidut käyttöoikeudet.



Kuva 2.4.2. Tietojärjestelmäkokonaisuus massaräätälöitävässä yrityksessä [6].

Tämä opinnäytetyö ei tutki tuotantoketjua edellä kuvatulla laajuudella, vaan keskittyy ainoastaan suunnittelijan työn helpottamiseen automatisoimalla häneltä toistoa vaativia tehtäviä. Lopputuloksia ja jatkotoimia käsittelevässä luvussa otetaan kantaa tuotantoketjun huomioon ottamiseen automatisaation implementoinnissa.

2.4.1 Tietokoneavusteinen suunnittelu

Tietokoneet ovat olleet jo pitkään arkipäivää teknisen suunnittelun apuvälineinä. Niiden laskentatehon kasvu ja komponenttien kehitys ovat mahdollistaneet sen, että kolmiuloitteisuus on niin tietokoneavusteisessa suunnittelussa kuin pelimaailmassa lähes ehdoton vaatimus. Lyhenne CAD tarkoitti ennen nykyistä merkitystään, tietokoneavusteinen suunnittelu (Computer Aided Design), tietokoneavusteista piirtämistä (Computer Aided Drafting). 1970 -luvulta alkaen ensimmäisissä tietokoneohjelmissa piirrettiin yksinomaan kaksiuloitteisesti. Näin toimiessa suunnittelijan tehtävänä on piirtää mallista kaikki tarvittavat projektiot, jotta se voidaan valmistaa. 2D-piirtämistä toteutetaan vielä, mutta vähenevässä mittakaavassa, lähinnä 1982 perustetun Autodeskin tuotteiden saaman laajan suosion ansiosta. Viimeiset ainoastaan kaksiulotteisesti suunnittelevat katoavat työmaailmasta eläkelöitymisen myötä, sillä kolmiuloitteisuuden tuomat edut ovat suuret, ja nykyään koulutus annetaan lähes ainoastaan kolmiuloitteisesti. 2D-suunnittelusta voidaan käyttää myös termiä CADD (Computer Aided Design and Drafting) [7]. Tässä opinnäytteessä lyhennettä CAD, käytetään tästä eteenpäin yksinomaan tarkoittamaan tietokoneavusteista suunnittelua, jossa osien mallinnus, sekä kokoonpanojen teko, tuotetaan kolmiuloitteisesti, jonka jälkeen valmiista rakenteesta luodaan haluttaessa kaksiuloitteiset osa-, valmistus- ja kokoonpanokuvat.

Tietokoneavusteisia suunnitteluohjelmia voidaan jakaa myös sen käyttökohteiden mukaan. Tässä opinnäytteessä tietokoneavusteisella suunnitteluohjelmalla tarkoitetaan nykyaikaista mekaniikkasuunnitteluun tarkoitettua ohjelmaa. Näiden ohjelmien yhteisinä piirteinä voidaan luokitella parametrisuus, *assosiatiivisuus ja piirrepohjaisuus*. Näiden sanojen merkitystä avataan seuraavassa alaluvussa 2.4.1.1 Parametrinen mallinnus. Seuraavassa on listattu mekaniikkasuunnitteluohjelmien käyttöönotolla saavutettuja hyötyjä: [8]

- **Lisääntynyt suunnittelun tuottavuus**. CAD-ohjelmiston käyttö helpottaa tuotteiden ja sen osien hahmottamista, mikä puolestaan vähentää suunnittelijoiden tarvitsemaa aikaa analysoida ja dokumentoida suunnitelma.
- **Geometristen muotojen hallittavuuden lisäys**. CAD-ohjelmistot sisältävät laajat mahdollisuudet luoda erilaista hallittua geometriaa, kuten matemaattisesti määritettyjä muotoja tai muotoon sulautuvia reunoja, joiden toteutus käsin olisi haastavaa.
- **Suunnittelun laadun parannus**. CAD-ohjelmistoilla kyetään hallinnoimaan mallia paljon tarkemmin kuin käsin, ja suunnittelun tarkoituksenmukaisuus

on helpompi todeta mittatarkasta 3D-mallista. Lisäksi ohjelmistojen käyttö mahdollistaa erilaisten analyysien teon suunnittelun yhteydessä.

- **Suunnitteludokumenttien parempi hallittavuus**. Tietokoneella olevia tiedostoja on helpompi hallinnoida kuin fyysisiä papereita, ja työpiirustuksista on yksinkertaisempaa luoda standardien mukaiset, ja sisällyyttää niihin vähemmän virheitä kuin käsin, sillä kuvat ovat aina mittakaavassa.
- **Tuotantotietokannan yhdenaikainen luominen**. Samalla kun tuotteesta luodaan tietokonemalli, sisällytetään tiedostoon muodon lisäksi mittoja, materiaaleja, materiaalivaatimuksia ja osataulukkoja, joita voidaan käyttää hyväksi tuotannon myöhemmissä vaiheissa.
- **Suunnittelun standardisointi**. CAD-ohjelmiin pystyy sisällyttämään suunnittelusääntöjä, joiden avulla voidaan varmistaa, että suunniteltu tuote pystytään myös valmistamaan. Esimerkiksi sallimalla vain tiettyjen reikäkokojen käytön, jota yrityksen konekanta pystyy toteuttamaan.

Mekaniikkasuunnitteluohjelmistojen lisäksi tietokoneavusteista suunnittelua varten on myös sähkösuunnitteluohjelmistoja, joista käytetään usein lyhennettä eCAD (Electrical Computer Aided Design), sekä lähinnä viihdekäyttöön, lelutuotantoon tai animointiin olevia 3D-muotoiluohjelmistoja, joiden avulla voidaan luoda ja muovata muotoja ilman tarkempia mitta-arvoja tai relaatioita. Sähkösuunnitteluohjelmistoista kerrotaan esimerkkinä Elecdes ja 3D-muotoiluohjelmista Blender.

2.4.1.1 Parametrinen mallinnus

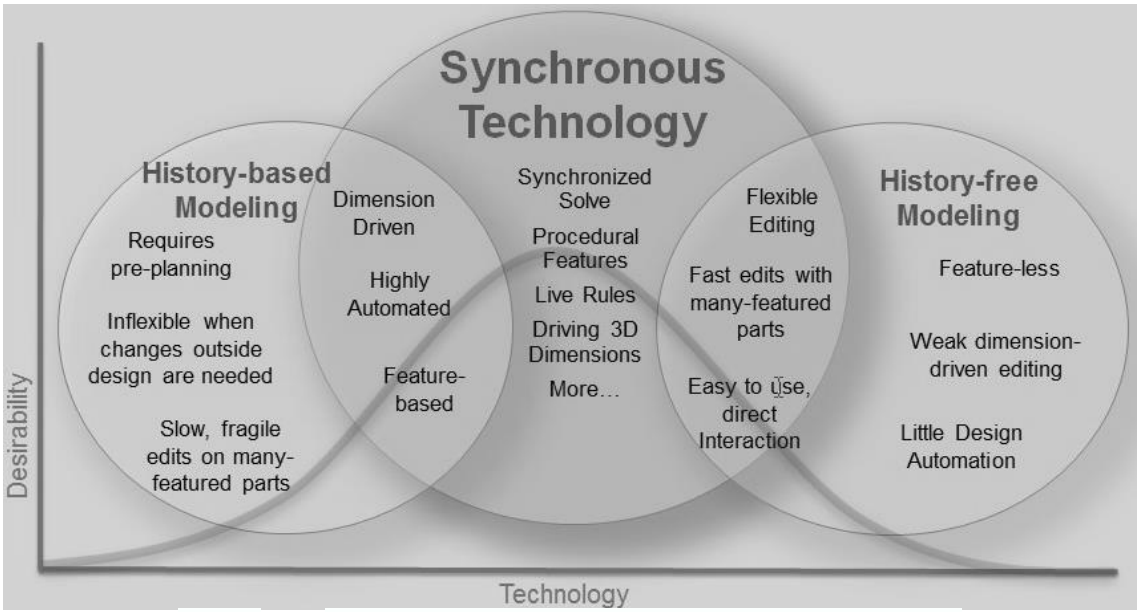
Parametric Technology julkaisi ensimmäisen parametrisen 3D-mallinnusohjelman jo vuonna 1988. Parametrisyydestä on kehittynyt kolmiuloitteisten CAD-ohjelmistojen perusvaatimus. Jukka Korpela kuvaa sanan parametri Pienehkössä sivistyssanakirjassaan seuraavasti: ”Parametri matemaattisen kaavan tai mallin taikka tietokoneohjelman olennainen muuttuja, jonka arvoja muuttelemalla saadaan aikaan erilaisia versioita”[9]. Parametrisessä mallintamisessa kappaleet ja rakenteet mallinnetaan käyttäen tarkkaa geometristä kuvausta. Parametrisyyden mallintamiseen tuo käytettyjen mittojen ja relaatioiden käyttäminen, joiden muutokset periytyvät kaikkiin kohteisiin missä kyseistä mittaa, osaa tai kokoonpanoa on käytetty. Näin ollen esimerkiksi kappaleeseen tehdyt muutokset periytyvät sekä kokoonpanotasolle että kappaleen sisältäviin työpiirustuksiin. Tämä assosiatiivisuus mahdollistaa mittalukujen käytön laskennallisissa kaavoissa ja poistaa turhan työn kun muutosta ei tarvitse tehdä jokaiseen käyttökohteeseen erikseen. Relaatioiden käyttö mallinnuksessa on luontevaa, kun esimerkiksi osatasolla luonnoksen viivat voidaan osoittaa samansuuntaisiksi, kun taas kokoonpanotasolla relaatioilla osoitetaan kappaleiden väliset liitokset toisilleen, esimerkiksi sylinterien samankeskisyytenä. Geometristen mallien käyttö mahdollistaa myös koneistuksen työstöratojen tai simuloinnin toteutuksen. Parametrisyys

mahdollistaa myös näissä tapauksissa tiedon siirron toiseen ohjelmaan, eikä mallia tarvitse luoda uudelleen toisella ohjelmalla. [10]

Parametrisen suunnittelun heikkoutena on voitu pitää sen piirrepohjaisuutta. Perinteisesti mekaniikkasuunnitteluohjelmat ovat olleet niin sanotusti historiapohjaisia. Tällä tarkoittaen, että kappaleen 3D-malli on luotu rakentamalla yksi piirre toisen päälle. Mallin rakenne on aiheuttanut muutoksia tehdessä piirteeseen, joka on ”historiapuun” alkupäässä, sen että tietokone joutuu laskemaan uudelleen kaikki tämän jälkeen luodut piirteet. Tämä on vaatinut tietokoneelta turhaa laskenta-aikaa ja -tehoa, sekä usein aiheuttanut jokaiselle suunnittelijalle tutun mallin ”räjähtämisen”, kun jokin historiapuun jälkipään piirteistä ei pysty rakentumaan puuttuvan tai muuttuneen referenssin vuoksi.

Synchronous Technology

Siemens on ratkaissut parametrisen mallintamisen ongelman kehittämällään suoramallinnusmenetelmällä nimeltä Synchronous Technology. Ajatus tekniikan kehityksen taustalla oli kuvan 2.4.3 mukaisesti yhdistää historiapohjaisen parametrisen mallinnuksen, sekä 3D-muotoiluohjelmien historiattoman mallinnuksen vahvuudet. Näin ollen piirteet eivät ole täysin riippuvaisia toisistaan, ja mallin jälkimuokkaaminen on jopa sata kertaa nopeampaa perinteiseen historiapohjaiseen mallinnukseen verrattuna.



Kuva 2.4.3. Ajatus Synchronous Tehcnologyn taustalla [30].

Suoramallinnuksen nopeus kohdistuu sekä suunnittelijan toiminnan nopeuttamiseen että myös mallin rakenteeseen, kun suunnittelijan ei tarvitse tietää miten malli on rakennettu muuttaakseen tätä. Myöskään tietokoneen ei tarvitse laskea koko kappaleen muodostumista piirre kerrallaan muokkausten tapahtuessa, vaan se tekee muutokset vain haluttuun piirteeseen. Myös muut valmistajat ovat yrittäneet kehittää Siemensin esimerkin mukaisesti omia suoramallinnustekniikoita vaihtelevalla menestyksellä,

huomattuaan sen tuoman suuren hyödyn mallien jälkimuokkaamiseen. Tekniikan tuomien etujen merkitys korostuu moniportaisessa tuotantoketjussa, jossa käytettävää mallia ei ole aina luotu jälkimuokkauksen mukaisella suunnitteluohjelmistolla. Tämä tilanne koskee lähes poikkeuksetta pieniä ja keskisuuria yrityksiä, kun kaikkea ei voida valmistaa itse. Tällöin yrityksien välisessä tiedonsiirrossa turvaudutaan usein lähes standardeihin tiedonsiirtoformaatteihin, kuten step tai iges. Kyseisten formaattien hyvä puoli on se, että lähes jokainen suunnitteluohjelma tukee niitä, mutta ongelmana on, etteivät ne siirrä historiapuuta mukanaan. Näin ollen perinteiset keinot muokata mallia eivät ole mahdollisia.

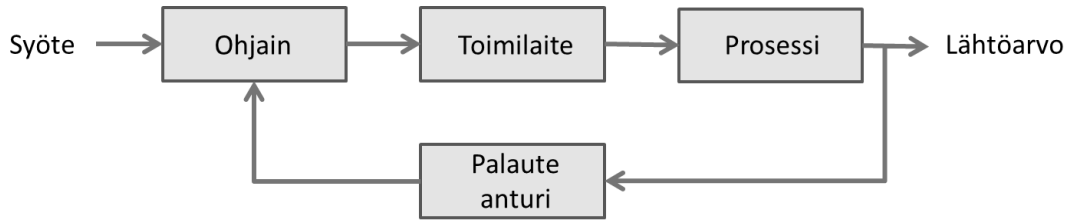
Synchronous Technology ei itsessään kuulu osaksi diplomityötä, sillä hyvin harva suomalainen yritys on omaksunut sen käytön, koska heillä ei ole ollut resursseja tai mielenkiintoa oppia uutta. Henkilökohtaisesti näen tämän olevan virheellinen päätös, sillä Synchronous tekniikan opittuaan työskentely nopeutuu ja se mahdollistaa paljon, mikä ei ole mahdollista *ordered*-puolella (Solid Edgen historiapohjainen mallinnus).

Tähän päättyy diplomityön Taustatietoa -luku. Seuraavassa luvussa käsittelen kirjallisuustutkimuksen tuloksia yleisiksi muodostuneista käsitteistä mitä tulee itse suunnitteluprosessiin, suunnitteluautomaatioon ja tuotteisiin, joihin suunnittelun automatisoinnin implementointia voidaan soveltaa.

3. SUUNNITTELUN AUTOMATISOINTI

Jukka Korpela määrittelee automatisoinnin ihmistyön korvaamisena automaattien suorittamalla työllä [9]. Vaikka sana automaatio yleistetään usein teknillisissä piireissä koskemaan teollisuusautomaatiojärjestelmiä, tämän opinnäytetyön yhteydessä sanalla tarkoitetaan suunnittelujärjestelmän automatisointia. Toisin kuin suunnitteluautomaatiosta, on teollisuusautomaatiosta saatavilla laajasti kirjallisuutta. Aiheeseen tutustuessani huomasin, että ajatustasolla perusperiaatteet ovat molemmissa samat, täten myös teollisuusautomaatiosta kertovaan kirjallisuuteen tutustuttiin näiden yhtäläisyyksien löytämiseksi.

Kuten teollisuusautomaation avulla saavutettavia etuja: toistettavuus, laadunhallinta, kasvanut tuotanto ja pienentynyt työvoiman tarve, myös suunnittelua automatisoimalla pyritään saavuttamaan samoja hyötyjä. Suunnitteluautomaation avulla suunnittelija vapautetaan toistoa vaativalta yksinkertaiselta työltä, arvoa tuottavaan suunnittelutyöhön, samalla vähentäen inhimmillisten virheiden määrää ja niiden toteutumisen mahdollisuutta. Myös moderniin tuotantoa ohjaavaan automaatiojärjestelmään verrattuna perustyökalut ovat lähes samat. Tuotantoa ohjataan tietokoneiden, ohjelmoitavien logiikoiden, sensoreiden ja toimilaitteiden avulla, kun taas suunnittelua automatisoidessa perustuu kaikki tietokoneella toimiviin verrannollisesti vastaaviin perustyökaluihin [11]. Aliohjelma sisältää ohjelmoitavan *logiikan*, eli *komentosarjan*, joka syötetään toimilaitteena toimivalle pääohjelmalle, eli opinnäytteen tapauksessa Solid Edgelle. Sensoreina voidaan nähdä sekä aliohjelman sisäisen syötteen tarkastuksen että Solid Edgen suunnittelutyökalut, joiden avulla suunnitteluparametreja voidaan rajoittaa.



Kaavio 1. Suljettu automaatiojärjestelmä.

Automaatiojärjestelmät voidaan jakaa kahteen osaan: suljettuihin ja avonaisiin. Suljetussa järjestelmässä syöteparametrejä verrataan saatuihin lähtöarvoihin palauteanturin avulla. Tällaisesta systeemistä esimerkkinä kodin lämmityspatteri, jonka termostaatti toimii palauteanturina ja näin ollen vertailee lämpötila-arvoa esiasetettuun arvoon nähden, ja säätää lämmityslaitetta tämän mukaisesti joko tuottamaan lisää

lämpöä tai vähentämään sen tuotantoa. Suljetun automaatiojärjestelmän periaate kaaviokuva 1. [8]

Kaavio 2. Avoin automaatiojärjestelmä.

Kaaviosta 2 voidaan havaita, että avoin automaatiojärjestelmä poikkeaa suljetusta järjestelmästä syöte- ja lähtöarvojen vertailun puutteena. Avoin järjestelmä ei näin ollen sisällä palautetta antavaa anturia, jonka signaalin mukaan järjestelmä säätää ohjainlaitetta. Avoimen järjestelmän hyötyjä ovat, että ne ovat yleensä yksinkertaisempia sekä edullisempia suljettuun järjestelmään verrattuna. Mikell P. Grooverin mukaan avointa automaatiojärjestelmää voidaan harkita tilanteissa, joissa seuraavat ehdot täyttyvät: 1. Ohjaimen suorittamat säädöt ovat yksinkertaisia, 2. Toimilaitteen toiminta on erittäin luotettavaa ja 3. toimilaitteeseen kohdistuvat voimat ovat niin pieniä, etteivät ne vaikuta sen toimintaan [8]. Vaikka suunnitteluautomaattia varten on mahdollista kehittää palauteanturina toimiva aliohjelma, joka tarkastaa Solid Edgen luoman rakenteen, nähdään tämä useaan automaatioprojektiin liian työläänä. On järkevämpää varmistaa järjestelmän toimivuus syötteitä rajoittamalla, laajoilla testeillä ennen käyttöönottoa, ja lähtöarvojen, eli Solid Edgen tuottamien rakenteiden ja kuvien, tarkastamisella ennen lopullista hyväksyntää. Näin Grooverin kaksi ensimmäistä sääntöä toteutuvat ja kolmas voidaan jättää huomioimatta, sen koskiessa mekaanisia toimilaitteita, ja järjestelmä voidaan toteuttaa avoimena. Toisaalta suunnitteluautomaation lopputuotteen tarkoituksenmukaisuuden tarkastaa suunnittelija, joka tekee myös mahdolliset muutokset, ja joka semantiikan myötä voidaan nähdä palauteanturina.

3.1 Tuotevaatimukset

Ennen kuin voidaan kuvata automatisoinnin rajoitteita ja mahdollisuuksia, täytyy listata vaatimukset tuotteelle, jonka yhteyteen automatisointia voidaan harkita. Tuoterakenteen tulee olla selkeä, hyvin ymmärretty ja mielellään modulaarinen. Ei-modulaarisen rakenteen suunnittelua voidaan kyllä automatisoida, mutta se on huomattavasti haastavampaa sekä tieto/taidon, kustannusten että lopputuloksen ylläpidon kannalta. Suunnittelun automatisointia harkitessa vaihtoehtojen ei tule olla kaikki tai ei lainkaan, vaan otetaan huomioon, että automatisoidaan se osa joka voidaan!

”Vaikeina taloudellisina aikoina yritykset, jotka eivät automatisoi,
tekevät sen omalla uhallaan.”
-EDA Inc. Solid Edge kotisivut [12].

Suunnittelun automatisointiprojektia helpottaa tuoterakenteen selkeys, sen hyvä ymmärrys ja modulaarinen rakenne. Rakenteen yksinkertaisuus tuo yritykselle myös

muita etuja. Tämä opinnäytetyö tarkastelee suunnittelutyön automatisoinnin lisäksi myös myyntikonfiguraattorien tuomia etuja. Suunnittelua automatisoidaan sekä suunnitteluautomaattien että tuotekonfiguraattorien avulla. Myyntikonfiguraattorit ovat yleistyvä trendi, joiden avulla tuotteen tilaaja joko itse tai asiantuntijamyyjän avustuksella yksilöi tuotteen tilauksen yhteydessä. Konfiguraattorista kokonaisvaltaisesti enemmän luvussa 3.3 Konfiguraattorit.

Tutustuessani konfiguraattoreiden tuomiin etuihin, kohtasin myös useita muita suunnittelua ja tuotantoa kehittäviä toimia, jotka tukevat myynti- ja tuotekonfiguraattorien käyttöönottoa. Mikäli yritys pystyy käyttöönottamaan myyntikonfiguraattorin, voidaan otaksua, että sen liiketoiminta on lähinnä tilauskohtaista. Tilauskohtainen toiminta puolestaan mahdollistaa vakiintuneiden valmistuksen ohjausprosessien käyttöönoton: [13]

- **MTO (Make to order)**, tuote valmistetaan tilauskohtaisesti
- **ATO (Assembly to order)**, tuote kokoonpannaan tilauskohtaisesti
- **ETO (Engineer to order)**, tuote suunnitellaan tilauskohtaisesti
- **STO (Ship to order)**, tuote toimitetaan tilauskohtaisesti
- **MTS (Make to stock)**, tuote valmistetaan varastoon.

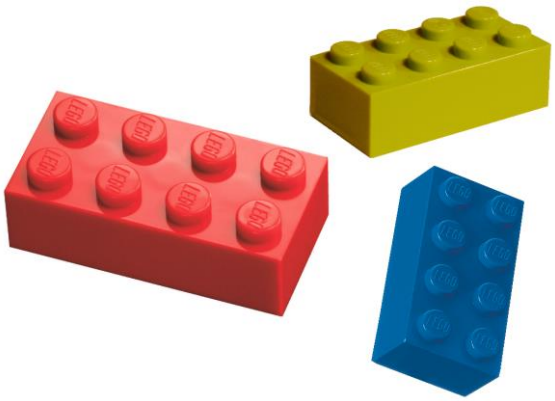
Tuote joka soveltuu hyvin konfiguroitavaksi, voidaan nähdä kuuluvan listan kahteen ensimmäiseen ryhmään, eli tuote valmistetaan tai kokoonpannaan tilauskohtaisesti. Myös tilauskohtaisesti suunniteltava tuote, voidaan hyväksyä suunnitteluautomatisointi projektiin, jos valmistusprosessi ei muutu paljon. Esimerkkinä tällaisesta tapauksesta jäähdytyskaappeja valmistava yritys, joiden komponentit (kaappi, jäähdytyskennot, kompressorit yms.) ovat aina samat, mutta kunkin komponentin määrä vaihtelee tilauksen mukaan. Näin saadaan tilauskohtaisen tuoman toiminnan etu sekä tuotannon että suunnittelun läpimenoaikojen alenemisella, sekä laadun ja sen ennustettavuuden parantuvuudella. Nämä kolme valmistuksen ohjausprosessia soveltuvat parhaiten tuotteille, jotka ovat tuotantomääriltään ja yksilöinnin asteeltaan kohtalaiset. Jotta automatisointiprojekti olisi kannattava, tulee menekki olla riittävän suuri. [13]

Tilauskohtaisen toiminnan haasteina nähdään tuotevariaatioiden ja niiden kustannusten hallinta. Tutkimusten mukaan toiset yritykset ovat asettaneet tiukan säännön, joka estää tuotteisiin tehtävät lisävaatimukset, joita ei ole etukäteen suunniteltu osaksi tuoterakennetta. Näin tuotteen hinnoittelu on yksinkertaisempaa kun toiveen toteutuksesta aiheutuvalle lisätyölle ei tarvitse arvioida hintaa. Etuna lisävaatimuksista kieltäytymisellä nähdään myös täysin tai osittain valmiiden tuotteiden varastoon valmistuksen mahdollisuus, jonka avulla kiireisen ajanjakson toimitusaikojen lyhyenä pitäminen voidaan varmistaa. Siinä tapauksessa, että yritys kieltäytyy varioimasta tuotteitaan toiveiden mukaan, tulee miettiä tarkkaan mitä tarjotaan. Vaarana on tarjota variaatioita mitä asiakkaiden kuvitellaan tarvitsevan, ja näin tuottavan varastoon suuria määriä uniikkeja lopputuotevariaatioita, joille ei ole kysyntää. Näin käy helposti jos pyritään erottautumaan kilpailijoista valmistamalla enemmän variaatioita kuin

kilpailijat. Yrityksissä, jotka hyväksyvät tuotteisiin tehtävät erilliset toiveet, on eduksi, jos toiveet pystytään toteuttamaan tuotteeseen mahdollisimman myöhäisessä vaiheessa. Näin tuotannon vaiheet pysyvät mahdollisimman pitkälle samana, eikä toive näin lisää useisiin vaiheisiin aikaa vaativia toimia tai työkaluja. [13]

Modulaarinen tuoterakenne

Modulaarisen tuoterakenteen koetaan olevan lukijoille selkeä. Tässä opinnäytteessä ei oteta kantaa mikä on sopiva tuote moduloitavaksi ja todettakoon vain, ettevät kaikki rakenteet sovellu moduloitavaksi. Tuoterakenteen modulaarisuus tuo selkeitä etuja suunnittelun automatisointia ajatellen, täten nähdään tarpeellisiksi listata joitakin moduloinnin mukanaan tuomia etuja. Kuvassa 3.3 nähdään yksi kaikkien aikojen tunnistettavin modulaarinen rakenneosa: lego rakennuspalikat.



Kuva 3.3. Lego rakennusmodulit.

Selkeä etu suunnittelua automatisoidessa saadaan kun moduuli- ja optiosuunnittelussa moduulit luokitellaan perus- ja valinnaisiin moduuleihin. Perusmoduulit muodostavat niin sanotun tuotealustan (product platform), ja ne voidaan nähdä myös tuotteiden jaotteluna tuoteperheisiin. Tuotteen on tarkoitus toimia itsenäisesti jo pelkästään perusmodulien avulla, mutta lisäksioptiona valittavat moduulit ovat lisäosia, joita asiakas voi halutessaan valita tuotteeseen. Esimerkkinä valinnaisista moduuleista toimii auton ilmastointi. Perusmoduuli, eli auto, toimii ilman tätä lisäoptiota. Kirjassa: Massaräätälöinnillä kilpailukykyä, esitetään loistava syy, miksi valinnaisten moduulien määrää on syytä rajata: ”Nykypäivänä täysin uutta tuotealustaan sopimatonta, vain tietylle asiakkaalle kehitettävää moduulia tai ominaisuutta ei lähdetä kustannuskysymysten takia tekemään, koska tällöin joudutaan huomioimaan elinkaaren mittainen tuki sekä uudelle että vanhalle järjestelmälle [13].” Eli ei riitä, että moduuli valmistetaan kerran, vaan sille on varmistettava varaosat ja tuki koko sen käyttöiälle.

Modulaarisesta tuoterakenteesta saadaan etu suunnitteluun kun huomioidaan, että moduulien eliniät voivat poiketa toisistaan. Näin yksittäisten moduulien kehitystä voidaan suorittaa moduuli kerrallaan, ja suunnitteluautomaatiojärjestelmään voidaan päivittää yksittäiset moduulit niiden valmistuttua, näin varmistaen koko rakenteen

jatkuva toiminta ennen ja jälkeen muutoksen. Tällä keinolla tuotteen elinkaarta voidaan jatkaa ilman, että koko tuoterakenne joudutaan suunnittelemaan aina täysin uudelleen.

Tuotannollisista lähtökohdista tarkastellen, modulaarinen tuoterakenne mahdollistaa usein tuotteelle lyhyemmän toimitusajan. Modulaarisuus mahdollistaa nopean reagoinnin muuttuviin asiakasvaatimuksiin, kun muutostyö sekä suunnittelussa että valmistuksessa kohdistetaan vain tiettyyn tuotteen osaan. Jopa tuotantoprosessia on mahdollista moduloida; kun tuotemuutokset kohdistuvat vain tiettyyn moduuliin, voidaan myös tuotannossa muutokset kohdistaa vain tiettyyn osaan prosessia. Näin variaatioiden vaikutus tuotantoprosessiin saadaan rajattua tiettyyn tuotantovaiheesseen, joka on vain pieni osa koko prosessia, eikä tarjottujen variaatioiden määrä vaikuta tuotannon tehokkuuteen ratkaisevalla tavalla. [13]

3.2 Suunnitteluautomaatio

Nykyiset mekaniikkasuunnitteluohjelmat mahdollistavat osien ja kokoonpanojen tarkan mallintamisen. Kolmiulotteisen parametrisen visuaalisen mallin avulla mahdolliset suunnitteluvirheet ja ongelmakohdat on helppo havaita jo ennen kalliiden prototyyppien valmistusta. Kolmiuloitteisen mallin luomista on pitkään kannustanut myös sen avulla helposti tuotettavien tarkkojen kaksiuloitteisten työ- ja kokoonpanopiirustuksien luonti. Teknologian kehitys on alkanut tukemaan tarkan CAD-mallin luomista myös muilla keinoin. Esimerkiksi nykyään 3D-mallista on helppo tuottaa mittatarkka kappale 3D-tulostamalla. Edellä mainituista syistä, tarkan 3D-mallin luomista ei kannata välttää. Ohjelmistojen, tekniikoiden ja tietokoneiden kehityksestä huolimatta, suunnitteluaikojen pituus on edelleen usein toimitusprosessin pullonkaula, ja yritykset osoittavat kasvavaa kiinnostusta sen nopeuttamiseksi.

Suunnittelijat kuluttavat ison osan työajasta piirustusten ja 3D-mallien rutiininomaiseen muokkaamiseen. Suunnittelua automatisoimalla voidaan turhat toistoa vaativat tehtävät automatisoida, näin parantaa työn miellekkyyttä, tehostaa suunnittelua ja vapauttaa suunnittelija lisäarvoa tuottaviin suunnittelutehtäviin. Samanaikaisesti pyrkimyksenä on vähentää inhimmillisiä virheitä, joita suunnittelijalla saattaa toiston ohessa muodostua, sillä liian myöhään huomattuna virheet voivat aiheuttaa merkittäviä kustannuksia. Hyvin suunniteltuna, tarkalla lähtöarvojen määrittämisellä ja testaamisella, suunnitteluautomaatti vähentää merkittävästi ihmisen tekemien virheiden määrää. Toimivan suunnitteluautomaatin avulla kannattavuus ja kilpailukyky parantuvat, kun vapautuneet suunnitteluresurssit voidaan kohdistaa tuotekehitykseen ja suunnittelun laatu on parantunut. Suunnittelun automatisointi palvelee yrityksen koko toimintaketjua: markkinoinnista loppudokumentaatioon. [13]

Tässä opinnäytetyössä suunnitteluautomaatilla tarkoitetaan aliohjelmia, joiden avulla voidaan joko täysin automatisoida jokin tapahtuma, tai suunnittelijan pienen vuorovaikutuksen avulla, suunnittelutehtäviä nopeuttavien työkalujen käyttö. Luvussa 5.2 esimerkkinä tällaisesta tapauksesta toimii Solid Edge käyttöliittymässä toimiva *Add-In*, joka luo halutun ohutlevyrakenteen, sekä luvussa 5.3 *makro*pohjainen

kokoonpanopiirustuksen toteuttava automaatti. Suunnittelun automatisointi ei ole ainoa keino tehostaa suunnittelua. Ensimmäisenä yrityksessä kannattaa noudattaa erästä suunnittelun perussääntöä: mitään ei kannata tehdä kahdesti! Eli osien uudelleen käyttöä kannattaa tehostaa. Vakioidut osat tai piirteet siirtävät edun myös tuotannon puolelle, kun osia voidaan tilata tai valmistaa varastoon, ja tietyn piirteen valmistustapaan voidaan panostaa.

3.3 Konfiguraattorit

Konfigurointi voidaan nähdä synonyyminä valinta vaihtoehtojen väliselle valintojen tekemiselle. Yksittäisen konfiguraattorin avulla voidaan edellä mainitun mukaisesti tehdä valintoja, jotka perustuvat ennalta määrättyihin valinta vaihtoehtoihin. Konfiguraattoreiden edeltäjänä voidaan pitää perinteistä ”rasti ruutuun”-tilauslomaketta, jossa tilauksen yhteyteen valitaan haluttavat optiot. Tässä opinnäytetyössä käsiteltävät konfiguraattorit jaetaan kahteen osioon. Ensimmäisenä kappaleessa 3.3.1 kerrotaan myyntikonfiguraattoreista, jonka avulla asiakas voi suorittaa tilauksen ilman muuta kanssakäymistä myyjänä toimivan yrityksen kanssa. Myyntikonfiguraattoreita pyritään käsittelemään laajasti teoreettisesta näkökulmasta, muttei niiden luomiseen tai ylläpitoon oteta kantaa. Toisena konfiguraattori-tyyppinä opinnäytteessä esitellään tuotekonfiguraattori. Tuotekonfiguraattori voi olla valintaperusteiltaan samankaltainen myyntikonfiguraattorin kanssa, jolloin ohjelman käyttäjänä toimii tuottajana toimivan yrityksen suunnittelija tai asiantuntija, mutta tuotekonfiguraattoria ohjaavat komennot voidaan myös periyttää esimerkiksi myyntikonfiguraattorilta. Tuotekonfiguraattori muodostaa valintojen pohjalta tarkan tuoterakenteen, jonka pohjalta suunnittelu tehdään. Edellä mainitun mukaisesti konfiguraattorijärjestelmä on useissa tapauksissa kaksiportainen: myyntikonfiguraattorilla tehty tilaus lähtee suoraan tuotekonfiguraattorille, joka muodostaa rakenteen ja tuottaa tarvittavat työpiirustukset. Luku 3.3.2 kertoo lisää tuotekonfiguraattoreista ja tämän opinnäytteen myötä luotu tuotekonfiguraattori esitellään luvussa 5.1.

Ennen kuin käsitellään erilaisia tuote- ja myyntikonfiguraattoreita eroineen, esitellään ensin yleisesti hyväksytyt konfiguraattori-luokat. Luokkajaottelu tehdään konfiguraattorin sisältämän rakenteen tarkastelun mukaan helpoimmasta vaikeimpaan. Ryhmiä on kolme: *primääriset*, *interaktiiviset* ja *automaattiset* konfiguraattorit, ja seuraavissa luvuissa kerrotaan kustakin.

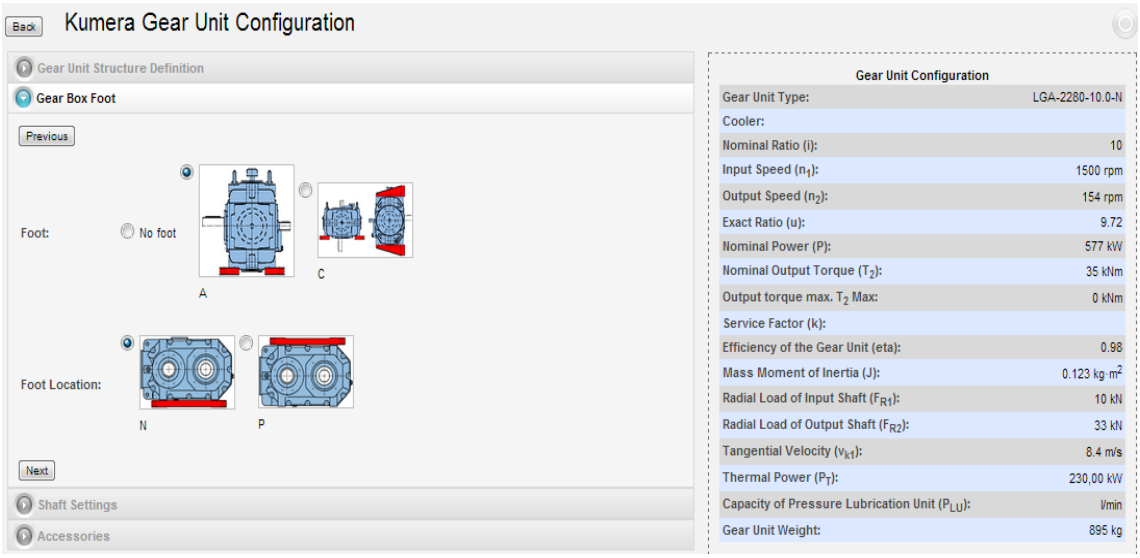
Primääriset konfiguraattorit

Tämä konfiguraattoriluokka on sekä toteutuksen vaikeusasteelta että toiminnaltaan yksinkertaisin. Primäärisissä konfiguraattoreissa tuotteen moduulit ja/tai komponentit valitaan luettelosta. Konfiguraattorin toiminta on niin yksinkertainen, ettei se sisällä lainkaan moduulien/osien yhteensopivuuden tarkastelua. Näin ollen esimerkiksi kappaleet voivat paikoittua toistensa päälle. Mallin toimivuuden tarkistus jää käyttäjän osaamisen varaan. Primäärisen konfiguraattoreiden sisältämä logiikka on hyvin vähäistä

[13]. IDEAL PLM:n Solid Edge asiakkaan Riikku Rakenteet Oy:n suunnitteluautomaatti on luokaltaan primäärinen, sillä sen syötteiden tarkastus on lähes olematon ja lopputuloksen toimivuus on käyttäjän harteilla. Tästä huolimatta Riikkun tuotekonfiguraattorin käyttöönotto on lyhentänyt Riikku Rakenteiden tarjouslaskentaan käytetyn työajan kahdesta päivästä kahteen tuntiin. Liittenä 1, löytyy Riikku Rakenteet Oy:stä tehty Case Study (englanninkielinen).

Interaktiiviset konfiguraattorit

Interaktiiviset konfiguraattorit sisältävät primäärisiä enemmän logiikkaa. Ne ohjaavat käyttäjää valinnoissa tarkistamalla valittujen komponenttien yhteensopivuuden, ja esimerkiksi pois-sulkemalla jatkosta moduuli- tai komponentti vaihtoehtoja jo tehtyjen valintojen perusteella [13]. Interaktiiviset konfiguraattorit vaativat toimiakseen jonkinlaisen sääntökannan muodostamisen, jossa eri moduulien/komponenttien yhteensopivuudet on määritelty, ja näin ollen interaktiivisen konfiguraattorin ohjelmointityö on haasteellisempi.

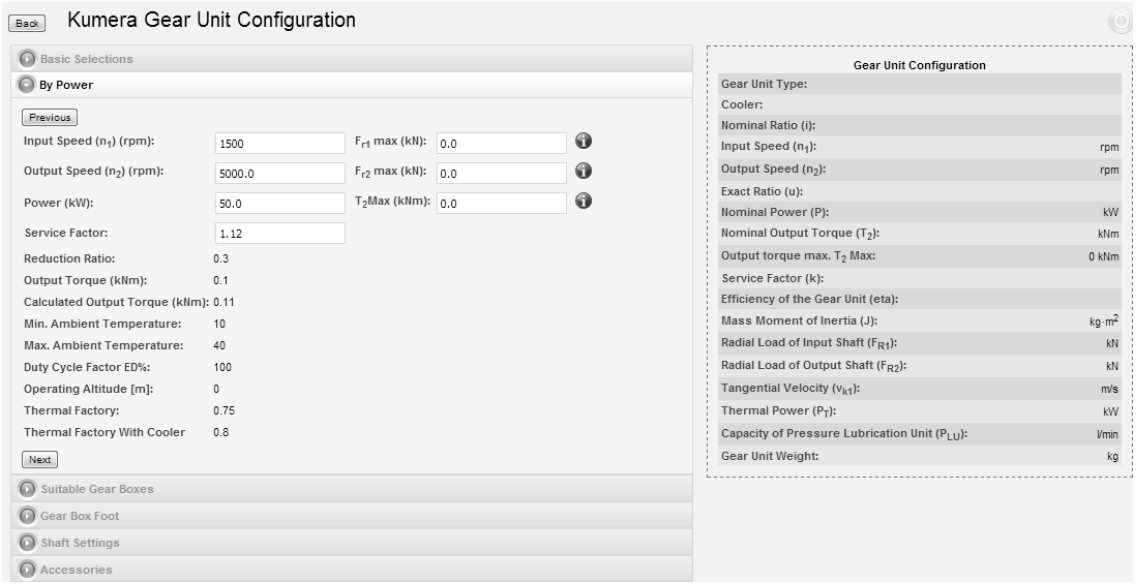


Kuva 3.4. Kumera Gear Unit Configuration, Express Selection [14].

Opinnäytteen aikana luotu yksinkertainen konfiguraattori, joka esitellään luvussa 5.1, lukeutuu interaktiivisiin konfiguraattoreihin. Kumera Oy:n voimansiirtoyksiköitä Solid Edgellä luova ”Express Selection” -konfiguraattori on myös luokitukseltaan interaktiivinen. Käyttäjä valitsee esiasetetuista arvoista aina itselleen sopivimman, ja jokainen käyttäjän tekemä valinta vaikuttaa seuraaviin optioihin. Esimerkki voidaan havaita kuvasta 3.4, jos jalan (”Foot”) kohtaan olisi valittu: ei jalkaa (”No foot”) -optio, ei jalan sijaintia (”Foot Location”) olisi ollut lainkaan valittavana. Konfiguraattori luo internet sivuilla luotujen valintavaihtoehtojen pohjalta komennon, jonka se lähettää Solid Edgelle, joka generoi halutun mukaisen 3D-mallin ja sen pohjalta 2D-kuvat, jotka lähetetään automaattisesti konfiguraattorin käyttäjän osoittamaan sähköpostiosoitteeseen.

Automaattiset konfiguraattorit

”Automaattiset konfiguraattorit ovat haastellisimpia ja pisimmälle vietyjä ratkaisuja, eivätkä perustu enää tuotteen moduulien tai komponenttien suoraan valitsemiseen asiakasvuorovaikutuksessa. Ne perustuvat attribuuttitiedon keräämiseen. Sitä kerätään joko tuotteen ominaisuuksista (esimerkiksi teho tai käsittelyvolyymi), asiakkaan tarpeista tai toimintaympäristöstä (esimerkiksi tasainen kuorma stabiileissa olosuhteissa tai kuormapiikit vaihtelevissa olosuhteissa). Tuotteelta vaadittavat ominaisuudet ovat johdettavissa kerätystä attribuuttitiedosta. Esimerkkinä voidaan mainita työkoneelta vaadittavan käyttötarkoituksen asettamat vaatimukset sen moottorin tehoon, jolloin automaattinen konfiguraattori voi sulkea pois liian pienet tuotemallit.” [13]



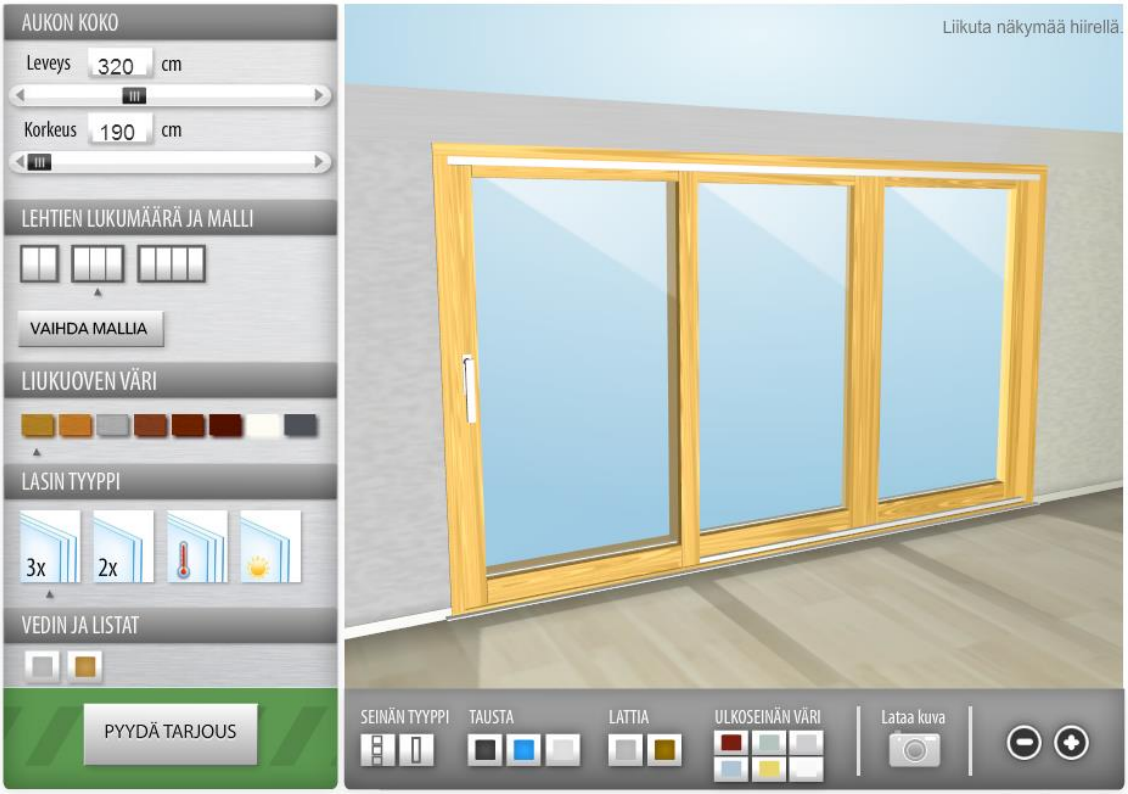
Kuva 3.5. Kumera Gear Unit Configuration, Normal Selection [14].

Kumera Oy:n ”Normal Selection” -konfiguraattori toimii lähes edellä kuvatun mukaisesti. Kyseistä konfiguraattoria ei voi kutsua täysin automaattiseksi, sillä se vaatii käyttäjältä jonkin verran vuorovaikutusta, mutta se on allekirjoittaneelle tunnetuista Solid Edgeä hyväksi käyttävistä konfiguraattoreista lähimpänä automaattista. Käyttäjän syötteiden pohjalta suoritetaan vaihteiston ominaisuuksia koskevaa laskentaa, jonka mukaisesti konfiguraattori ehdottaa seuraavassa vaiheessa laskentatuloksiin sopivimpia vaihteistovaihtoehtoja (”Suitable Gear Boxes”). Vaihe, joka seuraa kun kaikki valinnat on tehty, ei näy käyttäjälle, mutta konfiguraattori tarkastaa myös valintavaihtoehtojen mukaisen vaihdelaatikon 3D-mallin olemassa olon. Jos kyseisestä konfiguraatiosta on luotu jo 3D-malli ja työpiirustukset, käytetään ne uudelleen hyväksi, mutta jos niitä ei ole, ne luodaan. Kuten mainittua, ei konfiguraattori toimi täysin automaattisesti, eikä tutkimuksissa löydetty täysin automaattista konfiguraattoria, mutta tämä voidaan nähdä esimerkkinä automaattisten konfiguraattorien luomisen haasteellisuudesta. Parhaimmassa tapauksessa konfiguraattori jopa loisi kyseisen konfiguraation osat itse, kun edellä mainitussa tapauksessa käytetään vain suunnittelijan aiemmin luomia osia.

3.3.1 Myyntikonfiguraattori

Tieto- ja kommunikaatiotekniikan kehitys on mahdollistanut entistä vuorovaikutteisemman sähköisen kaupankäynnin. Internetin välityksellä asiakkaat voivat aktiivisesti seurata tuotteen saatavuutta, ja tehdä tilauksia ilman muunlaista kommunikaatiota myyjänä toimivan yrityksen kanssa. Myyntikonfiguraattorit ovat suhteellisen uusi ilmiö sähköisen kaupankäynnin saralla. Kyseisen tietokoneohjelman avulla asiakas tekee tilauksen joko itsenäisesti tai myyjän avustuksella. Myyntikonfiguraattorit on usein luotu niin, että ohjelma antaa visuaalisen palautteen jokaisesta valitusta vaihtoehdosta. Kuten aiemmin mainittu, valmistettavien tuotteiden modulaarinen rakenne on suunnittelunäkökulmasta perustavanlaatuinen edellytys myös myyntikonfiguraattorille, ja tästä on etua myös tuotekuvauksen visualisoitavuudessa ja selkeydessä tilausta tehdessä. ”Modulaarinen tuoterakenne mahdollistaa tuotteiden konfiguroinnin eli asiakaskohtaisen tuotteen ominaisuuksien määrittämisen vaihtoehtoisten ja erilaisiin asiakastarpeisiin vastaavien tuoteoptioiden perusteella” [13]. Myyntikonfiguraattori voidaan sijoittaa esimerkiksi yrityksen Internet-sivuille. Myös muita sijainteja, kuten yrityksen toimipaikassa sijaitseva fyysinen toimilaite, voidaan käyttää hyväksi, mikäli asiantuntijan apu myynnissä koetaan tarpeelliseksi.

Asiakkaan itsensä suunnittelemana, tuote tuottaa ainutlaatuista lisäarvoa, ja myyntikonfiguraattorin avulla se voidaan tuottaa kustannustehokkaasti. Vaikka konfiguraattorin avulla tuote vain räätälöidään esiasetettujen arvojen mukaisesti, saa asiakas haluamansa kaltaisen tuotteen, ja tunteen kuin olisi luonut jotain uutta. Mika Aho kertoo konferenssipaperissaan, että: ”Tarpeet massaräätälöintiä tukeville tietojärjestelmille ovat kasvaneet huomattavasti. Gartner ennustaa, että 80% todennäköisyydellä vuoteen 2015 mennessä ihmiset kustomoivat 90% informaatiostaan, työkaluistaan, koulutuksestaan sekä teknologisista resursseistaan, joita he käyttävät töissä, kotona tai vapaa-ajalla” [6]. Tämä kertoo yksilöllisyydestä, mutta myös tarjonnan paljoudesta. Täten yrityksien tulee pystyä erottautumaan kilpailijoistaan tarjoamillaan tuotteilla, sekä niiden markkinointikeinoilla. Yksi keino erota kilpailijoistaan on tarjota muita laajemman tuotevalikoiman. Tämä keino voi olla riskialtis, sillä varastoarvo kasvaa helposti, ja tuotteen tuki läpi tämän elinkaaren tulee ottaa huomioon. Näin ollen strategian kannattavuus laskee jyrkästi. Vuonna 2007 valmistuneen tutkimuksen mukaan Suomessa tutkituista yrityksistä vain 43 % käytti konfiguraattoreita myynnin tukena, ja ainoastaan 14 %:ssa tutkituista yrityksistä, kaikki tilaukset tulivat konfiguraattorin kautta. [15]



Kuva 3.6. Profin Oy. Myyntikonfiguraattori [16].

Suomalainen teollisuus on perinteisesti menestynyt korkean teknologisen osaamisen ja asiakaskohtaisesti räätälöityjen ratkaisujen avulla. Näiden perinteiden tukemiseksi myyntikonfiguraattorit tarjoavat asiakkaalle loistavan keinon vaikuttaa tuotteen ominaisuuksiin tai ulkonäköön, tarkoitusperistä riippuen. Myyntikonfiguraattorin avulla asiakas tekee valintoja, jotka useissa ratkaisuissa ovat kuvaavan nimen lisäksi myös visuaalisesti esitetyssä muodossa havainnollistamassa lopputuotetta. Näin ollen asiakas ei ole enää vain tuotteen ostaja, vaan myös suunnittelija, kommentoija, kehittäjä tai jopa verkostomarkkinoija [13]. Erikoistuneita ikkuna- ja ovijärjestelmiä valmistavan Profin Oy:n liukuovia tuottava myyntikonfiguraattori on nähtävissä kuvassa 3.6. Kyseisen konfiguraattorin avulla asiakas voi ennen tilauksen muodostamista tutkia erinäköisiä ja -kokoisia ratkaisuja kolmiulotteisen visualisoinnin avulla. Muokattuaan halutun kaltaisen konfiguraation, voi asiakas pyytää tarjouksen luodusta mallista napin painalluksella.

Myyntikonfiguraattoreiden toimintaan tai rakenteeseen ei oteta tässä opinnäytetyössä tätä lukua enempää kantaa. Myyntikonfiguraattori ei useissa tapauksissa ota kantaa tarkkaan rakenteeseen, vaan tarjoaa vain havainnollistavan visualisoinnin tuotteesta, joka auttaa tilausta tehdessä. Kuten edellä mainittu, konfiguraattorijärjestelmä on usein kaksiportainen: myyntikonfiguraattorilla tehty tilaus lähtee joko suoraan tuotekonfiguraattorille tai asiantuntijalle, joka syöttää tilauksen järjestelmään. Suunnittelija voi joutua myös mallintamaan osan tai koko tuotteen myyntikonfiguraattorin luoman tilauksen tai tarjouspyynnön pohjalta. Massaräätälöinnillä kilpailukykyä -kirjaa varten tehdyn selvityksen mukaisesti: mikäli 70–80 % tilauksista on helposti johdettavissa samoista perusratkaisuista,

konfiguraattorien avulla voidaan saavuttaa huomattavia toimitusaika- ja kustannusetuja [13]. Seuraava luku: Suunnittelun automatisoinnin haasteet, esittää kysymyksiä, jotka suunnittelun automatisointia suunnittelevan yrityksen tulee miettiä. Kyseinen luku pätee sekä myynti- että tuotekonfiguraattoreihin.

3.3.1.1 Suunnittelun automatisoinnin haasteet

Edellä olevissa luvuissa on käsitelty sekä itse tuotteeseen että tuotantoon liittyviä haasteita. Tässä luvussa keskitytään konfiguraattorin luomisen, käytön ja ylläpidon aikaisiin asioihin, jotka kannattaa huomioida jo ennen kehitysprojektin alkua.

Hinta

Mika Ahon mukaan nykytrendinä on, että ihmiset ovat valmiita maksamaan nykyistä enemmän henkilökohtaisesti räätälöidyistä ja personoiduista tuotteista [6]. Maksuhalukkuuteen voi vaikuttaa myös se, onko tuotteen avulla tarkoitus erottua yksilöitymällä, vai onko erilaisuudelle vaatimuksena tekninen ominaisuus. Esimerkiksi sähkömoottorin ulkonäöllä ei varmasti ole merkitystä, mikäli se ei pysty tuottamaan tarpeeksi tehoa, tai sen kiinnitys on epäkelpo. Suunnittelun automatisointiprojektilla on myös oma hintalappunsa. Ennen projektin aloitusta voidaan miettiä sekä toimittajana että asiakkaana, paljonko yksilöllisyydestä ollaan valmiita maksamaan.

Kuten edellä mainittua, on oleellisen tärkeää tuntea tarkat tuoterakenteet, jotta konfiguraattorista saadaan täysi etu. Konfiguraattori-projektin yhteydessä voidaan myös panostaa tuotekohtaisen laskennan tehostamiseen. Useissa yrityksissä on haasteita esimerkiksi kohdistaa kustannuksia oikeille tuotteille, ja tämä voi johtua virheellisistä tuoterakenteista. Rakenteissa voi olla nimikkeitä joita tuotteissa ei ole, tai tuotteen kokoonpanosta voi puuttua käytettäviä nimikkeitä, kuten ruuveja ja muttereita. Lisäksi, jos materiaalien hintatiedot eivät ole ajan tasalla, seuraa tuotteille väärät materiaalikustannukset. Myös välittömän ja välillisen työn kohdistamisperusteet voivat olla puutteelliset, tai esimerkiksi puutteelliset tai vanhentuneet konetuntihintatiedot aiheuttavat vääristyneitä hinnoitteluperusteita tuotteelle. Vääristyneitä tuotekohtaisia kustannuksia aiheuttaa myös jos ei tunneta koko tuotteen elinkaarta tai sen vaikutuksia, kuten varaosa- huolto- ja jälkimarkkinavaiheen tuottojen ja kustannusten yhteysvaikutuksia. [13]

Asiakastoiveet

Asiakastoiveet tuotekonfiguraatioihin ovat ”kaksiteräinen miekka”, jossa hinta on myös oleellinen tekijä. Suostumalla asiakastoiveisiin saadaan lyhyellä aikajänteellä lisättyä menekkiä ja asiakkaiden lojaalius tuottajaa kohtaan kasvaa. Toisaalta: ”Yrityksen täytyy analysoida, minkälainen tuotteen varioituminen tarjoaa sellaista lisäarvoa, jotta asiakkaat valitsevat räätälöidyn tuotteen standardituotteen sijasta, hyväksyvät pidemmän toimitusajan ja ovat valmiita maksamaan siitä hieman korkeampaa hintaa” [13]. Korkeamman hinnan ja pidemmän toimitusajan lisäksi myyjänä toimivan yrityksen tulee huomioida myös miten toiveen toteutus pystytään toteuttamaan nykyisillä

tuotantoresursseilla ja miten se vaikuttaa vakiotuotteiden tuotantoon. Pitkällä aikajänteellä menekki voi alkuperäisen arvion sijasta jopa pienentyä, kun kokonaisvaikutus liikevaihtoon paljastuu. Massaräätälöinnillä kilpailukykyä -kirjan mukaan osa tutkituista yrityksistä on ottanut systemaattisen kieltäytymisen asiakastoiveita kohtaan [13]. Näin yritys ei jää ”palveluloukkuun”, kun tuotteille pitää pystyä tarjoamaan myös varaosat koko tuotteen elinkaarelle. Asiakaskohtaisen tuotannon kohdalla tämä johtaa puolivalmistevarastojen arvon kasvuun ja turhaan sitoutuneeseen pääomaan, kun varastossa on varaosa jota ei välttämättä koskaan tarvita.

Asiakas ei ole kiinnostunut tuotevariaatioiden lukumäärästä tai kuinka monipuolisesti tuote on varioitavissa, vaan haluaa tarpeeseen sopivan tuotteen mahdollisimman vaivattomasti. ”Suunnittelun tehokkuuden optimi on saavutettu silloin, kun se täyttää sekä asiakkaan, että yrityksen tarpeet ja minimoi molempien osapuolten kokonaiskustannukset” [17].

Parhaat ideat eivät aina tule tuottajana toimivan yrityksen sisältä. Tuotetrendit sekä -vaatimukset muuttuvat ajan kuluessa, ja asiakastoiveiden käsittely voi tuoda tämän esille nopeammin kuin myyjä itse huomioi uuden tai muuttuneen tarpeen. Loppupäätelmänä asiakastoiveista: Myyntikonfiguraattoriin on hyvä lisätä kohta, johon voi syöttää lisätoiveita tuotteelle ja käsitellä ne tapauskohtaisesti. Joissakin tapauksissa voidaan huomata, että sama toive esiintyy usein ja tämä voi johtaa konfiguraation lisäämiseen tuotevalikoimaan, mikäli se päätetään ottaa osaksi hyväksyttäviä tuotevariaatioita.

Integraatiot

Myyntikonfiguraattorin kehityksen alkuvaiheessa tulee ottaa huomioon sen käyttöympäristö, sillä tämä sekä rajaa sovelluksen toteutuskeinoja että vaikuttaa kehitysprojektissa huomioitaviin asioihin. Esimerkkinä, Internet on markkinoinnin kannalta yrityksille loistava asia, ja sen tuomia etuja kannattaa käyttää tehokkaasti hyödykseen. Nykyiset nopeat verkkoyhteydet ja tietokoneiden tehokkaat laskentatehot mahdollistavat myyntikonfiguraattorin sijoittamisen asiakkaiden käytettäväksi Internetin välityksellä, ja tämä onkin nykytrendin mukaista. Näin asiakkaat näkevät suoraan tilauksen visualisoinnin ja saavat arvion tilauksen kestosta sekä hinnasta. Massaräätälöinnillä kilpailukykyä -kirjaa varten tehdyn tutkimuksen mukaan näiden asioiden havainnointi lisää asiakasuskollisuutta. Tämä kuitenkin lisää huomattavan tietoturva-aukon tilanteessa, jossa Internetissä toimiva myyntikonfiguraattori on suoraan yhteydessä tuotekonfiguraattoriin, tai mikäli konfiguraattorien toiminta edellyttää yrityksen PDM-järjestelmän yhdistämistä verkkopalveluun [13]. Tähän seikkaan kannattaa paneutua, sillä tietoturva on ehdottoman tärkeää säilyttää, mutta samalla verkon yli jaettavan sovelluksen yhteydessä kaikilla käyttäjillä on aina käytössään uusin ja yhtäläinen versio, kun ohjelma päivitetään samanaikaisesti. Verkkosovellusta suunniteltaessa kannattaa ottaa huomioon myös se, ettei kaikkia yksityiskohtia ja rakenteellisia komponentteja kannata esittää sekä laskentatehovaatimusten kasvun että myös kopioinnin vaaran vuoksi. Mainittakoon vielä, että vaikka konfiguraattoriin

sisällytettävän tiedon määrä on lähes rajaton, tulee sen määrää harkita. Liian paljon valintoja ja monimutkaista tietoa sisältävä myyntikonfiguraattori vain hämmentää asiakasta ja pahimmillaan siirtää asiakkaan asioinnin kilpailijalle.

Nykyisin tyypillisten metalli- ja konepajateollisuuden yritysten tietojärjestelmäkokonaisuus voi olla täysin perustellusti edellä kuvassa 2.4.2, kuvatun mukainen. Usean eri järjestelmän yhteen toimivuutta ei auta, että ohjelmistotoimittajia tiedon- ja tuotannonhallinta järjestelmille löytyy lukemattomia. Järjestelmäintegraatiot eivät ole ainoastaan konfiguraattoreita koskeva haaste. ”Viimeaikaiset tutkimukset kertovat järjestelmäintegraatioiden haasteista: 30–40 % yritysten IT-budjeteista käytetään integroinnin kehittämiseen, ylläpitämiseen ja tukeen. Vastaavasti 61 % tietohallintojohtajista kokee järjestelmien ja prosessien integraation olevan avainprioriteetti” [13]. Tässä opinnäytteessä ei tutkita yksittäisten integraatioiden luomista konfiguraattoreiden ja hallintajärjestelmien kesken. Mainitaan vain, että integraatioiden luominen voi olla tehtävälaajuudeltaan yhtä kattava kuin itse konfiguraattorin luominen. Kuten edellä mainittua, automatisoidaan asia kerrallaan, tuloksen ei tarvitse olla heti kaiken kattava. Tuotteiden valmistus- ja toimitusaika eivät kuitenkaan saa viivästyä järjestelmäintegraation puutteen vuoksi. Tilaus pitää pystyä käsittelemään pian, jotta se saadaan heti tuotantoon ja tämän myötä toimitettua asiakkaalle.

Projekti

Edellä olevasta voidaan huomata, että konfiguraattorin kehittäminen tuo mukanaan myös paljon muita mahdollisuuksia tehostaa yrityksen toimintaa. Jokaisella toimintaa tehostavalla toimella on kuitenkin selkeä tavoite ja tämän tavoitteen saavuttamisella on hinta. Antti Vanha-Viitakoski perustelee hinnan hyvin selkeällä tavoitteella: ”Investointi suunnittelun automatisointiin lisää yrityksen kilpailukykyä ja mahdollisuuksia vuorovaikutukseen asiakkaan, myynnin ja suunnittelun välillä. Pitkälle automatisoiduilla 3D-malleilla pystytään hyvin nopeasti tuottamaan selkeät ja havainnollistavat mallit jo esisuunnittelun aikana, mikä helpottaa keskustelua ja myös kaupan käyntiä asiakkaan kanssa” [18]. Oikeaan kohteeseen ja hyvin suoritettuna, konfiguraattori voi lyhentää suunnitteluun käytettävää työaikaa jopa 90 %.

Suunnittelun automatisoinnin hintalapun arvioiminen voi olla hankalaa. Tapauksissa, jossa osaaminen suunnittelun automatisointiin löytyy yrityksen sisältä, on hinnan muodostuminen erityisen hankalaa. Osaksi siitä syystä, että koko ohjelmointityö voidaan tehdä ”muun työn ohessa”, tai koska kehitysprojekti usein jatkuu pienempinä kehitysaskeleina käytön aikaisten huomioiden pohjalta. Jossain tapauksissa projektiin voidaan palkata esimerkiksi diplomi- tai lopputyöntekijä, jolloin hinnan muodostaminen on selkeämpää. Tämän kaltaisiin ohjelmistopohjaisiin kehitystoimiin löytyy myös paljon ohjelmointiprojekteja suorittavia yrityksiä, joiden kautta suoritettuna voidaan taata ohjelmiston vaatimuksien mukainen toiminta, vastuulliset tahot ja selkeä hinta. Solid Edge -ohjelmiston yhteyteen tällaisia projekteja ovat suorittaneet esimerkiksi Eneris Solutions Oy sekä Citrus Solutions Oy. Yrityksillä on usein myös aiempaa

kokemusta ohjelmoinnilla saavutettavista hyödyistä, ja ne osaavat arvioida ennalta projektin kannattavuutta.

Eräs projektissa huomioon otettava tekijä on myös vastuutekijät. Vastuulliset henkilöt tai tahot on hyvä tunnistaa ohjelmiston kehitykseen, testaukseen ja jatkotoimiin. Virheitä tai puutteita ohjelmistossa tulee alkuun olemaan lähes varmasti, vaikka kuinka paljon käyttöönottoa edeltävää testausta tehdään. Mikäli ohjelman on tehnyt yrityksen työntekijä, kannattaa varautua myös tilanteeseen, jossa hän syystä tai toisesta poistuu yrityksen palkkalistoilta. Jääkö konfiguraattori tuolloin tuon hetkiseen tilaan, vai varmistetaanko, että myös toinen työntekijä hallitsee ohjelmiston logiikan? Myös ohjelmistotalojen kanssa toimiessa tulee sopia erikseen jatkokehityksestä. Huomioitavia asioita ovat esimerkiksi: jäävätkö ohjelman luojalle kaikki oikeudet tuotteeseen, ostetaanko jatkokehitystä vaativat toimet lisäpalveluna, vai suoritetaanko ne itse?

3.3.2 Tuotekonfiguraattori

Eräs tämän opinnäytteen tavoitteista on tutkia mitä mahdollisuuksia on toteuttaa tuotekonfiguraattori, joka esivalintojen mukaisesti lähettää syötteet Solid Edge -suunnitteluohjelmistoon. Tämän saatuaan Solid Edge luo syötteiden mukaisen 3D-mallin tarvittavine työ-, osa- ja kokoonpanopiirustuksineen. Esivalinnat voidaan asettaa edellä kuvatun mukaisesti täyttymään myyntikonfiguraattorin syötteiden, automaattisesti kerätyn tiedon tai käyttäjän syötteiden mukaisesti. Alustavaksi tavoitteeksi opinnäytteen esimerkeille otettiin interaktiivisen konfiguraattorin toimintaperuste. Työn tulosten perusteella pyritään vapauttamaan suunnittelija tuotteen kehitykseen, kun hänen ei tarvitse luoda kyseisiä dokumentteja ja malleja itse. Suunnitteluautomaatio ei voi kuitenkaan poistaa suunnittelijan roolia täysin, sillä useissa tapauksissa hänen on alkuun pitänyt luoda mallit, jota tuotekonfiguraattori käyttää ja lopulta tarkistettava lopputuotteen toimivuus.

Opinnäytteen aiheeseen tutustuttaessa kohdattiin myös sekä Siemensin että kolmansien osapuolien luomia valmiita konfiguraattoriratkaisuja, joita Solid Edgen yhteyteen on saatavilla. Näistä lisää kappaleessa 3.4 Ulkoiset ohjelmistokehittäjät. Huomiona valmiista olemassa olevista ratkaisuista, useat ratkaisut käyttävät apunaan Microsoft Office Excel -taulukkolaskentaohjelmistoa. Exceliä voidaan käyttää sekä käyttöliittymänä että matemaattisena apuvälineenä.

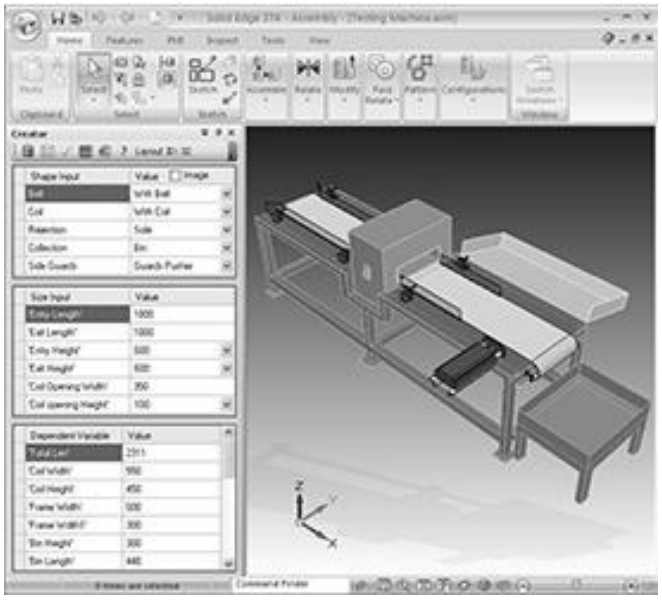
Esimerkkinä tuotekonfiguraattoreista edellä mainittu Riikku Rakenteet Oy:n käyttöön Citrus Solutions Oy:n luoma ”suunnitteluautomaatti”. Kyseinen automaatti on Riikku Rakenteet Oy:n suunnittelijoiden päivittäisessä käytössä, ja sen avulla saatu etu parvekeratkaisujen tarjouslaskentavaiheeseen on ollut merkittävä. Riikku valmistaa ja asentaa parvekekaiteita, -lasituksia ja julkisivujen alumiinirakenteita. Konfiguraattorin toiminta on selkeä. Suunnittelija valitsee myyjän esiselvityksen mukaiset toivotut kaide- ja lasityypit Riikku Rakenteiden valikoimasta, ja syöttää kunkin parvekkeen mitat järjestelmään. Tämän jälkeen kun esivalinnat on tehty, lähetetään syötteet Solid Edge -ohjelmaan, joka muokkaa valmiista osista syötteiden mittojen mukaiset kappaleet ja

yhdistää ne yksittäisen parvekkeen kokoonpanotiedostoksi. Tästä kokoonpanosta luodaan samalla osakuvat tuotantoon ja kokoonpanokuvat kokoonpanoa varten. Konfiguraattori ottaa myös huomioon saatavilla olevan kaidemateriaalin pituuden, ja kertoo kuinka paljon kyseisellä mitalla syntyy materiaalihukkaa, sekä antaa katkaisuluettelon. Koska kyseinen tuotekonfiguraattori on tyypiltään primäärinen, eli se ei tarkasta syötteiden yhteensopivuutta, on käyttäjän oltava asiantunteva suunnittelija. Lisää Riikku Rakenteet Oy:n suunnitteluautomaatista voi lukea Siemensin tekemästä case studystä liitteestä 1 (englanninkielinen).

3.4 Ulkoiset ohjelmistokehittäjät

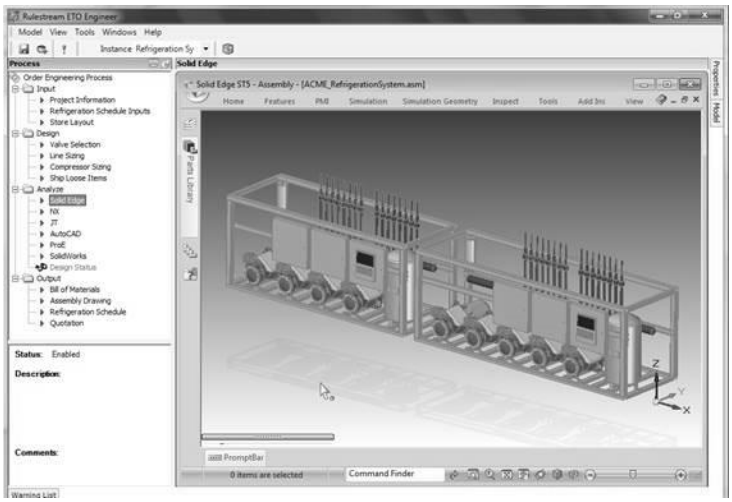
Tässä kappaleessa kerrotaan lyhyesti kolmannen osapuolen tarjoamista lisäosista Solid Edge -ohjelmiston tueksi. Kolmannella osapuolella tässä tapauksessa tarkoitetaan ohjelmiston tai palvelun tarjoajaa, joka ei ole ohjelmiston loppukäyttäjä tai sen tuottaja, eli Siemens PLM Software, vaan luo lisäarvoa alkuperäiseen ohjelmistoon. IDEAL PLM voidaan laskea kolmanneksi osapuoleksi, sillä tämä kouluttaa ja konsultoi ohjelmistokaupan lisäksi asiakkaitaan. Koska tarkoitus on kuvata lisäohjelmisto-osien tuottajia, ei IDEAL PLM toistaiseksi lukeudu tähän katergoriaan. Riippuen tämän opinnäytetyön alulle pistävistä toimista, näin voi tulevaisuudessa olla, jos suunnitteluautomaatioprojektit saadaan hyvin tuotteistettua ja ne nähdään kannattaviksi.

CAD-ohjelmistoihin lisäarvoa tuottava ohjelmistotarjonta on oma hyvin kattava teollisuuden ala. Kolmannen sektorin ohjelmistotuottajat ovat usein pieniä tahoja, ja he tuottavat usein samaa palvelua tai ohjelmaa myös muille ohjelmistoille. Siemens PLM Software on listannut ”Solid Edge partnereiden” luomat lisäosat kotisivuilleen. Lista sisältää 28 itsenäistä ohjelmistotuottajaa, joiden avulla Solid Edgen tukena voidaan esimerkiksi luoda kappaleiden koneistuskoodia (CAMWorks for Solid Edge), tai tuottaa ammattilaistason markkinointikuvia tuotteista (KeyShot). [19]



Kuva 3.7. CADECplus Configurator.

”Solid Edge Apps” -lista sisältää myös kolme Solid Edgen yhteydessä käytettävää konfiguraattoria: *CADCustomization*, *CADECplus Configurator* (kuva 3.7) ja *Edge Design Configurator*. Tutustuin opinnäytteen teon yhteydessä näihin ohjelmiin, sekä niiden tuottamaan lisäarvoon tämän opinnäytetyön yhteydessä. IDEAL PLM:n käytännön mukaisesti toistaiseksi ei ole tarkoitus edustaa muita kuin Siemensin tuotteita, ja täten ne ovat tämän työn kannalta merkityksettömiä. Jokaisella näistä kolmesta konfiguraattorista on hieman erilainen lähestymistapa automatisoida Solid Edgeä. Jo ohjelmien käyttöliittymät poikkeavat toisistaan, kun yksi käyttää hyväkseen Solid Edgen kanssa samaa käyttöliittymää, toinen internet-selainta ja kolmas käyttää hyväkseen Excel -taulukkolaskentaohjelmiston vahvuuksia käyttöliittymänään. Tutkimustuloksena voidaan myöntää, että valmiiden ohjelmistojen etuina ohjelmoitavaan automaatioratkaisuun nähden ovat sekä nopeampi käyttöönotto että ohjelmointiosaamisen tarpeen poisto. Molemmat syyt ovat painoarvoltaan raskaita, ja niitä IDEAL PLM:n tulee vielä harkita tulevaisuudessa, kun alkuun on saatu kokemusta ohjelmointiperäisten ratkaisujen käyttöönotosta sekä ylläpidosta. Solid Edge -asiakkaat voivat toki olla itsenäisesti yhteydessä palvelun tarjoajiin, ja harkita onko tuki ilman suomalaista edustajaa riittävä.



Kuva 3.8. Siemens PLM Software - Rulestream [26]

Siemens PLM Software on omistanut myyntikonfiguraattori *Rulestream*in vuodesta 2009 lähtien, ja kuvan 3.8 mukainen integraatio Solid Edgeen valmistuu ST6:n myötä vuonna 2013. Solid Edge asiakkaille yhteinen piirre on rajalliset resurssit, sillä yritykset eivät ole pääsääntöisesti suuria, ja kuten Solid Edgen hinnat, eivät ohjelmistoinvestoinnit voi olla suuria. Rulestream on tarkoitettu lähinnä suurille yrityksille, ja sen myyntiä tukevat ominaisuudet ovat kattavat. Tämä valitettavasti asettaa ohjelmistolle myös ison hintalapun mikä sulkee Rulestream -ohjelmiston pois potentiaalisena myyntikonfiguraattoriratkaisuna useille Solid Edge -asiakkaille. Tutustuin opinnäytetyön myötä myös Rulestreamin tarjoamiin vakuuttaviin ominaisuuksiin. Mainittakoon, että Rulestream on täysin itsenäinen sovellus ja on saatavilla muidenkin CAD-ohjelmien yhteyteen. [20]

4. SOLID EDGE AUTOMAATIO ALUSTANA

Diplomityön kohdistuminen Solid Edgen yhteyteen tuntui luontevalta ratkaisulta CAD-asiantuntijana työskenneltyäni. Siemensin tarjonta ohjelmaan sekä Teamcenter-integraatioon oli tuttu, mutta Solid Edgen yhteyteen ohjelmoitavat logiikat olivat niin allekirjoittaneelle kuin IDEAL PLM:lle jossain määrin rajalliset. Tämä diplomityö rajoittuu tutkimaan *.NET teknologia* pohjaisia ohjelmointikieliä, ja keskittyy pääsääntöisesti *Visual Basic .NET* -kieleen. Kuten tekstissä on aiemmin mainittu, suomalaiset Solid Edge automaatioprojektit on aiemmissa yhteyksissä suorittanut jokin muu ohjelmistotalo kuin IDEAL PLM.

Tutkittuani laajasti Solid Edgen ohjelmointi-kirjastoja ja eri mahdollisuuksia automatisoida tämän toimintaa, voidaan todeta, että kaikki mahdollinen ei ole kannattavaa. Solid Edgellä suunnittelu normaalisti on suhteellisen yksioikoista ja vaivatonta. Vaikka tietokone suorittaa komennot ihmistä nopeammin, on huomioitava myös koodin kirjoittamiseen ja testaamiseen vaadittava aika. Ennen ohjelmointiin ryhtymistä kannattaa tutustua Solid Edgen laajoihin ominaisuuksiin, jotka on tarkoitettu helpottamaan ja nopeuttamaan suunnittelijan työtä, kuten: *Variables*, taulukko, jonka avulla voi toteuttaa dimensioiden välisiä sääntöjä ja vertailuja, sekä yhdistää esimerkiksi Excel -taulukkolaskentaohjelmalla tehtyjä tuloksia kappaleisiin; *Family Of Parts*, jonka avulla voidaan luoda helposti ja nopeaa osaperheitä, joiden keskenäiset mitat tai ominaisuudet vaihtelevat; *Alternate Assemblies* avulla voi luoda vaihtoehtoisia kokoonpanoja, vaihtelevin osien tai niiden asennoin ja *Standard Parts* vakio-osakirjasto.

Versiorajoitus

Haluan huomauttaa tässä vaiheessa työtä, että tutkimustulosten yleistäminen koskemaan sekä uudempia että vanhempia ohjelmistoversioita sisältää joitakin rajoituksia. Ensimmäinen rajoitus tulee kunkin uuden julkaisun mukana tulevista uusista työkaluista, jotka lanseeraus tuo mukanaan. Esimerkiksi Solid Edge ST6 toi mukanaan suuren määrän uudistuksia pintamallinnukseen ja täten on luonnollista, etteivät aiemmat ohjelmistoversiot ymmärrä uusia pintamallinnuskomentoja, kuten ”Redefine Surface”.

Toisena rajoitteena on mahdollista, että olemassa olevien työkalujen toimintaan tehdään muutoksia uuden ohjelmistoversion myötä, ja täten myös työkalun ohjelmoitavaan kutsuun tulee muutoksia. Komento uudessa muodossaan voi esimerkiksi vaatia lisäparametrin määritykseensä, jonka avulla uusi toiminnallisuus on mahdollinen ja komento vanhassa ohjelmoitavassa muodossaan lakkaa toimimasta.

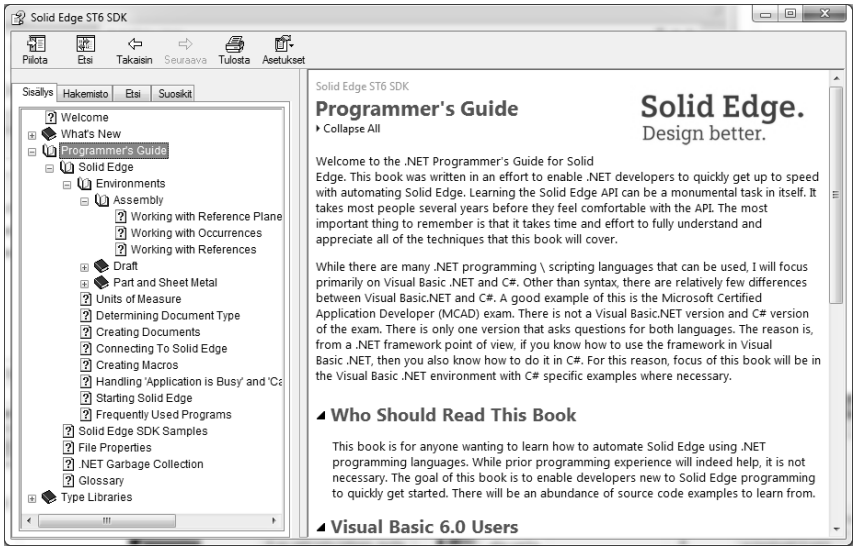
Kolmas rajoittava tekijä on Solid Edgen ohjelmointirajapinta, jonka avulla aliohjelmat kommunikoivat isäntäohjelman kanssa. Muutokset ohjelmointirajapintaan

voivat estää koko liitännäisen toiminnan. Esimerkkinä tällaisesta muutoksesta on ISEAddInEx objekti, jonka edeltäjä ISEAddIn objekti korvattiin, koska se ei tue 64-bittistä järjestelmää.

Viimeinen rajoite koskee Solid Edgen versiota. Riippuen onko kyseessä 32- vai 64-bittinen versio, on liitännäisten rekisteröinnissä pieniä eroja, lähinnä koskien rekisteröintityökalua.

4.1 Ohjelmointirajapinta

Solid Edgen ohjelmointirajapinta pohjautuu *Microsoft Component Object Technology*yn *(COM)*. COM -teknologian vahvuuksia ovat komponenttien uudelleen käytettävyys ja sitä tukevien ohjelmointikielien määrän laajuus. Keskustelin ennen diplomityön alkua henkilöiden kanssa, jotka olivat työskennelleet Solid Edgen ohjelmointirajapinnan kanssa. Kohdehenkilöihin lukeutui Eneris Solutions Oy, EDA Inc. ja Siemens PLM Softwaren edustajia. Jokaisen yrityksen edustaja kertoi Solid Edge API:n olevan hyvä ja erittäin kattava. Havannoin tämän todeksi oman tutkimukseni myötä, joskin tulee ottaa huomioon, etten ole ennen ohjelmoinut muiden isäntäohjelmien yhteyteen, eikä minulla täten ole vertailukohtia. Vaikka case-esimerkit toteuttavat CAD-ohjelmalle hyvin alkeellisia komentoja, kuten kappaleen pursotusta, osien lisäämistä kokoonpanoon ja paikoitusta piirrokseen, totesin, että Solid Edge voidaan asettaa tekemään kaikkia samoja komentoja koodin, kuin käyttöliittymänkin avulla, ellei enemmän.



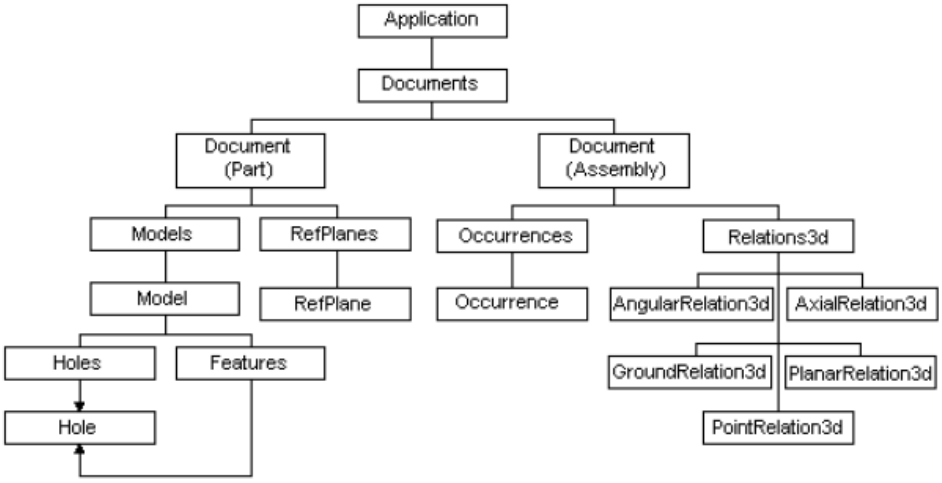
Kuva 4.1.1. Solid Edge Programmer's Guide

Siemens tukee vahvasti Solid Edgen jatkokehittämistä tarjoamalla ohjelman mukana tulevan ohjelmointi oppaan *(Programmer's Guide*, kuva 4.1.1), joka on saatavana myös ”kirjallisena” versiona Siemensin sivuilta [21]. Kyseinen dokumentti auttoi rajaamaan käytettävää ohjelmointikieltä, sillä siinä missä kuvan ”digitaalisen” -version ohjelmointiesimerkit ovat poikkeuksetta esitetty Visual Basic .NET -kielellä, löytyy kirjallisesta versiosta myös esimerkit *C Sharp (C#)* käyttäen. Käytin työskentelyssä

hyväkseni molempia dokumentteja, sillä kirjallista oli helpompi selata, mutta ohjelman mukana tulevan version yhteydessä esitettävät *tyyppikirjastot (Type Library)* taas olivat suureksi avuksi tulkitessa käytettäviä objekteja ja metodeja. Myöhemmin tässä luvussa esittelen saatavilla olevan *Spy for Solid Edge* -sovelluksen, joka auttaa *oliomallin* (eng. object model) opiskelussa. Tyyppikirjastoja ovat: [21]

- *Solid Edge Assembly Type Library*
- *Solid Edge Constant Type Library*
- *Solid Edge Draft Type Library*
- *Solid Edge Framework Type Library*
- *Solid Edge FrameworkSupport Type Library*
- *Solid Edge Geometry Type Library*
- *Solid Edge Part Type Library*

Jokaiseen Solid Edgen yhteyteen ohjelmoitavaan sovellukseen tulee sisällyttää vähintään Framework Type Library, jotta ohjelmat voivat ”keskustella” yhdessä, eli sovellus kykenee käsittelemään SE:n objekteja. Solid Edge tulee olla asennettu tai vähintään tyyppikirjastot sijoitettu tietokoneelle, jolla ohjelmointi tehdään, jotta tyyppikirjastoihin voidaan viitata. Lisäkirjastoja tulee sisällyttää sen mukaan, missä ympäristössä olevien *objektien (olioiden)* kanssa sovelluksen haluaa toimivan ja useat näistä sisältävät viittauksia edellä mainittuun [28]. Tyyppikirjastot on luotu jokaisen Solid Edge -mallinnusympäristön mukaan ja on huomioitava, että kokoonpano- (Assembly) ja piirros (Draft) -ympäristöt poikkeavat, niin ohjelman kuin ohjelmointi käyttöliittymän puolesta, muista. Tyyppikirjasto referenssistä lisää luvuissa, jossa käsittelen ohjelmointikieliä.



Kaavio 3. Solid Edge objekti hierarkia [29].

Solid Edgen objekti hierarkia voidaan esittää kaavion 1 mukaisesti puumallina. Muuttujien esittely ja ohjelman luonti alkaa aina Application -objektista ja etenee kaavion mukaan alaspäin. Hierarkiassa objektit on esitetty alkuun monikossa ja tämän

jälkeen yksikössä, kuten Documents ja Document. Monikkomuoto kertoo objektin olevan kokoelma ja näin ollen sisältävän yksikkömuodon objekteja. Esimerkiksi: Documents -objekti on Document objekteja sisältävä kokoelma. Toisin sanoen, esimerkiksi RefPlanes kokoelma objekti sisältää numeroidut RefPlane objektit. [29]

Utility APIs

Samoin kuin Solid Edge tuo asennuksen mukana apuohjelmia, kuten View and Markup, jotka antavat suunnittelutyöhön lisäarvoa, on ohjelmointipuolella myös hyödyllisiä apuohjelmia, joiden avulla tietoa ja tiedostoja voidaan käsitellä pikaisesti. Taulukossa 1 esitellyt kevytohjelmat suorittavat itselleen ominaisia toimintoja ilman Solid Edge ohjelman käynnistämistä.

Taulukko 1. Solid Edge Utility APIs

Nimi	Kuvaus	Tyyppikirjasto
SEInstallDataLib	Solid Edgen asennustietokanta	SEInstallData.dll
SolidEdgeFileProperties	SE tiedostojen ominaisuus objekti tiedot	PropAuto.dll
RevisionManager	Solid Edge Revision Manager objektit	RevMgr.tlb

SEInstallDataLib avulla voidaan kerätä tietoa, kuten asennuspäivämäärä tai versio, tietokoneelle asennetusta Solid Edgestä. SolidEdgeFileProperties:n avulla pystytään tarkastelemaan Solid Edge tiedostojen: osa (*.par*), piirros (*.dft*) ja kokoonpano (*.asm*), ominaisuuksia Solid Edgen ulkopuolella. Tämä on huomattavan paljon nopeampaa ja helpottaa ison tietomäärän käsittelyä. Solid Edge tukee *Microsoft Structured Storage API (MSDN)* standardia tiedon säilömisessä. Viimeisen, eli RevisionManagerin avulla voidaan hallinnoida tiedostojen ominaisuuksia, hallinnoida tiedostojen keskinäisiä sidoksia ja integroitua Solid Edgen yhteyteen saatavaan Insight -tiedonhallinta järjestelmään. [28]

Spy for Solid Edge

Pitkäaikaisen Solid Edge harrastajan ja ohjelmoijan Jason Newellin luoma Spy for Solid Edge -sovellus auttaa ohjelmoijaa ymmärtämään Solid Edgen ohjelmointirajapinnan toimintaa. Sovelluksen ensimmäinen versio on kehitetty jo vuonna 2005, ja nykyisessä avoimen lähdekoodin muodossaan se on ladattavissa CodePlex -sivustolta. [25]

Kuva 4.1.2. Spy for Solid Edge -logo [25].

Solid Edgen oliomalli on hyvin laaja, ja täten se on vaikea oppia ja visualisoida. Näistä syistä suosittelen ohjelmoijaa lataamaan Spy for Solid Edge -sovelluksen, mikäli hän aikoo ohjelmoida Solid Edgen yhteyteen. Sovellus yksinkertaistaa oliomallin opiskelua, sillä se tarjoaa lisäinformaatiota Solid Edgen objekteista, prosesseista ja tapahtumista, ollen näin hyödyksi kokeneillekin ohjelmoijille [25]. Löysin sovelluksen vasta diplomityön tuloksia kirjoittaessa, ja en tästä syystä käsittele sovellusta tämän laajemmin, sillä se ei ollut osa työn saavutuksia.

4.2 Käyttöliittymä

Vain harvat systeemit ovat 100% automatisoitu. Systeemi tarvitsee usein vähintään aloitussignaalin toimiakseen [22]. Tuotekonfiguraattorin tapauksessa aloitussignaali voi edellä mainitun mukaisesti tulla suoraan myyntikonfiguraattorilta, mutta usein se on käyttäjän antama komento, ja tämän antaakseen hän tarvitsee käyttöliittymän (User Interface, UI). Yksinkertaisimmillaan käyttöliittymä voi olla pikakuvake, komentokehoite tai yksittäinen valikon näppäin, joka käynnistää halutun tapahtumaketjun. Solid Edgen yhteyteen luotava aloituskomento voidaan toteuttaa kolmella tavalla: Aliohjelma, joka lähettää isäntäohjelmalle suoritettavat käskyt; Add-In, eli liitännäinen, joka toimii isäntäohjelman käyttöliittymässä tai Makro, komentoketju joka laukaistaan joko Solid Edgen sisältä tai sen ulkopuolelta. Kaikki kolme keinoa ohjata Solid Edgen toimintaa käsitellään myöhemmissä luvuissa.

Eräs syy miksi pidin Visual Basic .NET -ohjelmointikieltä potentiaalisena vaihtoehtona ohjelmointikieleksi, jolla toteutan työssä esitettävät esimerkki tapaukset ja tutkisin Solid Edgen ohjelmointirajapintaa, oli sen helppous luoda visuaalisia käyttöliittymiä (*Graphic User Interface*). En tiennyt opinnäytteen aloitusvaiheessa kuinka laajan itsenäisen sovelluksen tulisin toteuttamaan, sillä diplomityön aloitusvaiheessa oli vielä mahdollista, että tuotekonfiguraattori toteutetaan Solid Edge -asiakkaalle. Vaatimukset käyttöliittymälle ovat suoraan rinnastettavissa automatisoinnin tasoon. Kun ohjelma on yksinkertainen ei käyttöliittymän tarvitse olla turhan laaja ja

monipuolinen. Käyttöliittymän robustius on olennaista, sillä huono käyttöliittymä voi pilata hyvän ohjelman.

Luvussa 3.4 esiteltyjen tuotekonfiguraattoreiden lähestymistapa käyttöliittymään on jokaisessa ohjelmassa eri. Yksi toimii aliohjelmana, toinen liitännäisenä ja kolmas käyttää hyväkseen Excel -taulukkolaskentaohjelmaa. Viimeinen on varteen otettava vaihtoehto mikäli ohjelman tulee käsitellä paljon tietoa, kuten sidoksia, geometrisiä ehtoja, relaatioita ja dimensioita, jolloin voidaan käyttää taulukkolaskentaohjelman vahvuuksia hyväksi. Toki sama aliohjelma voi lähettää ja vastaanottaa komentoja samanaikaisesti usean ohjelman kesken, eli esimerkiksi aliohjelmaan kirjatut syötteet lähetetään alkuun Exceliin, jossa niillä suoritetaan laskutoimenpiteitä ja palautetaan takaisin aliohjelman kautta Solid Edgelle.

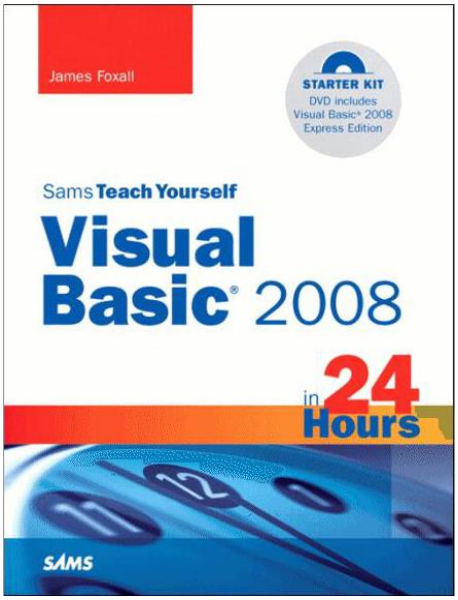
Aihetta ei käsitellä tämän enempää, sillä ihmisen ja tietokoneen välistä kommunikointia on tutkittu paljon, eikä tämän tutkimuksen pääpaino ole käyttöliittymässä. Yhteenvetona mainittakoon, että pidä käyttöliittymä mahdollisimman yksinkertaisena, ettei se turhauta käyttäjää, sillä se palvelee myös ylläpidettävyyttä ja päivitettävyyttä.

4.2.1 Visual Basic .NET

Tunnetusti tietokoneet ovat tyhmiä ja ne suorittavat vain tehtäviä joita käyttäjä sille asettaa. Siinä missä ihminen on päättäväinen ja joustava, eivät ohjelmat osaa tehdä muunlaisia päätelmiä kuin niiden on ohjelmoitu tekemään [22]. Kuten edellä mainittu, ohjelmointikokemukseni oli alkuun lähes olematon. Täten ennen ohjelmointiin pääsyä, minun tuli valita ohjelmointikieli, jonka opiskelen ja jolla suoritan ohjelmoinnin esimerkkitapauksiin. Vaikka minulla oli alkeet -tason tuntemus *C++* -ohjelmointikielestä, valitsin tässä työssä käytettäväksi ohjelmointikieleksi Visual Basic .NET:n. Valintaa tukevia syitä olivat: kielen matala haastellisuus, graafisten käyttöliittymien luomisen helppous, Siemensin tarjoama Solid Edgen ohjelmointiopas, joka sisältää esimerkkikoodeja Visual Basic .NET kielellä, Microsoftin tarjoama maksuton *ohjelmointiympäristö (Integrated Development Environment, IDE)* Visual Basic 2008 Express Edition (kuvakaappaus logosta kuva 4.2) ja viimeisimpänä: löysin kuvan 4.3 mukaisen erinomaisen oppaan kielen opiskeluun.

Kuva 4.2. Microsoft Visual Basic 2008 Express Edition [31].

Käsittelen tässä työssä lähinnä Visual Basic .NET -kielellä suoritettavaa ohjelmointia, mutta esittelen tämän luvun lopussa muutamia poikkeuksia koskien C# ja C++ -kieliä.



Kuva 4.3. Sams Teach Yourself Visual Basic 2008 in 24 Hours [23].

Luvussa 4.1 käsittelin Solid Edgen ohjelmointirajapintaa, ja kerroin sen olevan COM -teknologiapohjainen. Luvussa en voinut sukeltaa liian syvälle ohjelmointiin, sillä kielivalinta asettaa poikkeuksia ohjelmointirajapinnalle. Käytettäessä *Microsoft .NET framework* (.NET) -teknologiaan pohjautuvia ohjelmointikieliä, kuten Visual Basic .NET tai C#, tulee ohjelmointi projektiin lisätä referenssi Solid Edgen tyyppikirjastoihin. Tämä tehdään, koska COM ja .NET objektit eivät "keskustele" suoraan keskenään, vaan tarvitsevat yhteentoimiakseen tulkin. .NET rakenne on kehitetty käyttäen hyväksi COM:n keskeisiä teemoja, kuten komponenttien uudelleen käyttö ja ohjelmointikielen neutraalius. COM -komponentit voidaan tuoda .NET ympäristöön, jossa ne ovat esittelyn jälkeen käytettävissä interop-tulkkia hyödyntäen. Tulkkina toimii automaattisesti referenssien mukana tulevat *ohjelmoinnin yhteentoimivuuden kokoonpanot* (*Interoperability Assemblies)*. Tämä kokoonpano sisältää tyyppikirjastojen määritykset .NET -ohjelmointia tukevassa muodossa, ja se tallentuu ohjelmoinnin päätteeksi ohjelman kanssa samaan kansioon [28]. Huomaa, että käyttäessäsi esimerkiksi Visual Studio 2008 -ohjelmaa, tulee referenssin "Copy Local" -valintavaihtoehto olla valittuna, jotta kyseinen kokoonpano kopioituu myös ohjelmakansioon.

Microsoft .NET rakenne on web-palveluiden luomiseen, julkaisemiseen ja käyttämiseen luotu alusta. Se tarjoaa laajan standardisoidun monikielisen ympäristön ohjelmien ja palveluiden integroimiseksi. .NET rakenne tarjoaa epädeterministisen lähestymistavan muistin vapauttamiseen. Tämä tarkoittaa, että siinä missä muistin vapauttamiseksi determistisessä lähestymistavassa riittää kun ohjelmointiobjektin referenssiksi asettaa "ei mitään" (Nothing tai NULL), epädeterministisessä tavassa muuttujat "liputetaan" olevan valmiita poistettaviksi [23]. Vasta tämän jälkeen rakenne hävittää objektin käyttäen *roskien kerääjää (garbage collection)*. Automaattinen roskienkeräys pyrkii poistamaan automaattisesti muistista tiedot, joihin sovellus ei tule enää viittaamaan, ja vapauttamaan niiden käyttämän muistitilan uudelleen käytettäväksi

[11]. Käytännössä tämä tarkoittaa Visual Basic .NET:iä käytettäessä, että nollatessasi referenssejä ohjelman muistista, tulee käyttää sen referenssin tyhjäksi asettamisen lisäksi Marshal.ReleaseComObject() -metodia [21]. Suosittelen muistin tyhjennykseen seuraavaa syntaksia, joka alkuun tarkastaa sisältääkö **`olio`** viittauksia, jos sisältää niin objekti liputetaan käyttäen Marshal.ReleaseComObject() -metodia ja **`olio`** viittaus asetetaan tyhjäksi:

```
If Not (olio Is Nothing) Then
      Marshal.ReleaseComObject(olio)
      olio = Nothing
End If
```

Visual Basic .NET on korkean tason ohjelmointikieli, joka on kehittynyt aiemmasta *DOS (Disk Operating System - levynkäyttöjärjestelmä)* versiosta nimeltä *BASIC (Beginner's All-purpose Symbolic Instruction Code - Aloittelijan yleissymboolinen ohjekoodi)*. Olio-ohjelmointi muistuttaa Englannin kieltä ja se on hyvin suosittu, lähinnä sen opiskelun vaivattomuuden ja helpon syntaksin myötä. Visual Basic on tapahtumapohjainen ohjelmointikieli, eli ohjelmoijan luoman käyttöliittymän pohjalta tehtävät tapahtumat, kuten napin painallus tai syötteen kirjaus, laukaisevat koodin [21]. Luvussa 5, esiteltävät ohjelmat on toteutettu tätä Visual Basic .NET -kieltä käyttäen ja ohjelmien koodit ovat liitteinä. En ota tässä diplomityössä kantaa koodin kirjoittamisen käytäntöihin, sillä konetekniikan opiskelijana tietoni asiasta ovat rajalliset, eikä se ole osa työni aihetta.

C#

C Sharp on myös Microsoftin .NET-konseptia varten kehittämä kieli, joka yhdistää C++:n tehokkuuden ja *Java*-kielen helppokäyttöisyyden [11]. C# koodi ei poikkea paljon Visual Basic .NET:stä. Esimerkkinä tästä on Microsoftin sertifioitu ohjelmistokehittäjä (Microsoft Certified Application Developer, MCAD) koe, josta ei ole erikseen VB .NET ja C# koetta, vaan kokeessa esitettävät kysymykset koskevat molempia kieliä [21].

```
if (application != null)
{
Marshal.ReleaseComObject(application);
application = null;
}
```

Edellä oleva koodin pätkä kuvaa riittävästi VB .NET:n ja C#:n eroja ja yhtäläisyyksiä. Sillä myös C# pohjautuu .NET -teknologiaan, on oliot liputettava erikseen roskien kerääjää varten. Syntaksi on hieman erilainen, esimerkiksi VB .NET ei tue loogisia operaattoreita, kuten: != samoin kuin C# ja asettaessa objektin referenssin tyhjäksi, käytetään Visual Basic .NET:ssä sanaa Nothing ja C#:ssa null.

C++

Solid Edgeä voidaan automatisoida käyttäen mitä tahansa ohjelmointikieltä, joka tukee ActiveX automaatiota. C-kielestä kehitetty C++ on standardisoitu olio-ohjelmointikieli, jolla on kirjoitettu suuri osa mailman käytetyimmistä sovelluksista [11]. Kyseinen kieli oli eräänä vaihtoehtona opinnäytteen case-esimerkkien ohjelmointikieleksi, sillä Tampereen teknillisen yliopiston konetekniikan opiskelijana olen suorittanut ohjelmoinnin perusteet kurssin, joka opettaa kyseisen kielen perusteet. Täten muutamia ohjeita Solid Edgen automatisointiin tätä ohjelmointikieltä käyttäen.

C++ kaltaisilla COM -pohjaisilla kielillä Solid Edgeä automatisoitaessa suurin ero .NET pohjaiseen ohjelmointiin on, että objekteja ei tarvitse erikseen liputtaa roskien kerääjää varten. ActiveX -kielillä riittää kun objektin referenssin asettaa tyhjäksi [28]. Luodaksesi aliohjelman, joka toimii Solid Edgen kanssa, tulee myös C++:ssa viitata alkuun Solid Edgen tyyppikirjastoihin. Esimerkiksi: ”`#import "framework.tlb"`. Tämän lisäksi objbase.h ja comdef.h kirjastot pitää liittää ohjelmaan [29]. Taulukko 2 esittää koodin perusaskeleet, jotka vaaditaan C++ kielellä Solid Edgen automatisoimiseksi.

Taulukko 2. C++ -ohjelmoinnin perusaskeleet.

Alusta COM-objektit	`CoInitialize(NULL);`
Julista Solid Edge aplikaatio objekti	`SolidEdgeFramework::ApplicationPtr pSEApp;`
Viittaa aplikaatio objektin referenssiksi joko käynnissä oleva Solid Edge tai uusi Solid Edge -sovellus	`pSEApp.GetActiveObject("SolidEdge.Application");` or `pSEApp.CreateInstance("SolidEdge.Application");`
Ohjelman koodi	Kirjoita tähän haluamasi tehtävät
Poista COM-objektien viittaukset	`CoUninitialize();`

Solid Edgen “lapsi”-objekteihin päästään käyttämällä sen “vanhempien” ominaisuuksia tai metodeja. Ominaisuuksiin ja metodeihin päästään käsiksi käyttämällä “->” operaattoria. Esimerkiksi pProfileSets nimisen objektin viittaus osatiedoston profileset -kokoelmaan on: `pProfileSets = pPartDocument->ProfileSets;` Viimeisenä ohjeena C++ ohjelmointikielellä ei Solid Edgen automatisointiin: et voi käyttää alaviivaa (”_”) olioiden nimeämisessä. Esimerkiksi: [29]

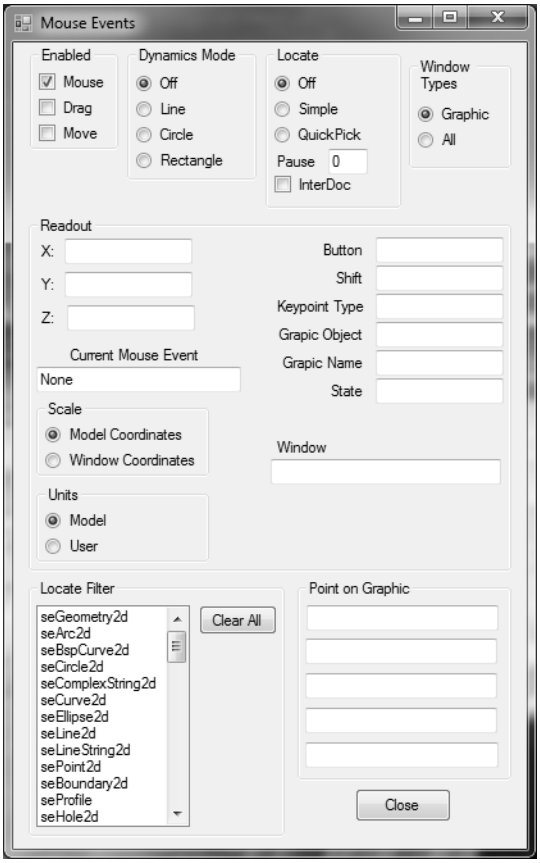
- Älä käytä "`_IApplicationAutoPtr`", vaan käytä "`ApplicationPtr`"
- Älä käytä "`_IPartDocumentAutoPtr`", vaan käytä "`PartDocumentPtr`"
- Älä käytä "`_IDMDBodyPtr`", vaan käytä "`BodyPtr`"
- Käytä "`ISEDocumentEventsPtr`"
- Käytä "`ISEMousePtr`"

Seuraavissa luvuissa erotellaan ja kerrotaan erilaisista sovellustyypeistä, joiden avulla Solid Edgeä voi automatisoida.

4.2.1.1 Makrot

Makro on tiettyyn alustaan tai ohjelmaan sidottu äärellinen joukko täsmällisiä komentoja, jotka ohjaavat päättyvän tehtävän suoritusta. Makron voidaan kuvitella olevan tietotekniikan vastine algoritmille, joka on yleismaailmallinen ohje tai resepti tietyn asian suoritukselle. [11]

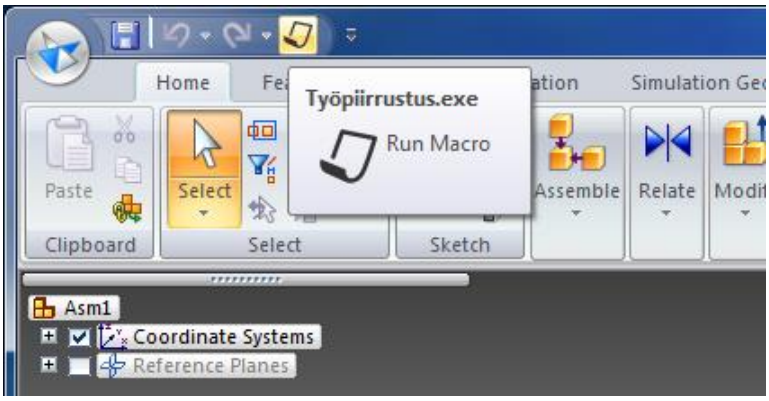
Solid Edgen yhteydessä makrot ovat yksinkertaisin tapa automatisoida ohjelman toimintaa. Sen vahvuutena on, että makron suoritus on huomattavasti nopeampaa kuin toistaa määrätyt komennot yksi kerrallaan manuaalisesti, sillä yksittäinen makro voi sisältää satoja peruskäskyjä. Makrot toimivat Solid Edgen yhteydessä ulkoisen sovelluksen toimintaperiaatteella, eli sovellus on isäntäohjelmasta erillinen *EXE (Executable File, suoritettava tiedosto)* -ohjelmistotiedostoformaatti. Solid Edge CAD -ohjelman asennuskansioiden mukana tulee muutamia valmiiksi ohjelmoituja sovelluksia, joiden avulla voidaan tutustua ohjelman automatisointiin. Kuvan 4.4 Mouse Events on sovelluksien makro esimerkki. Kyseistä makroa voidaan käyttää saadakseen Solid Edge ohjelmointirajapinnasta lisätietoa. Ohjelmat koodeineen löytyvät kansiosta: ”asennuspolku”\Solid Edge\Custom.



Kuva 4.4. Mouse Events -makro

Makrojen luontiin on useita eri keinoja. Helpoin on asentaa tietokoneelle internetistä ladattava makron nauhoitusohjelma, joka käynnistettäessään nauhoittaa kaikki käyttäjän syötteet, kuten hiiren tai näppäimistön napin painallukset. Siemens PLM Softwaren kilpailija Dassault Systèmes tarjoaa kuvatun mukaisen ”makro recorder” -toiminnon SolidWorks CAD -ohjelman mukana. Makrojen nauhoittamisen huonona puolena voidaan nähdä turhan pitkien ohjelmointikoodien syntyminen, sillä haluttujen toimintojen väliin mahtuu usein paljon turhaa, kuten hiiren osoittimen liikkeet.

Luvussa 5.3 esittelemäni kokoonpanopiirroksen luova makro on toteutettu käyttäen Microsoft Visual Basic 2008 Express Editionia. Makrot ovat oiva keino automatisoida asioita, jota suunnittelija(t) toistaa säännöllisin väliajoin. Rajoitteena tulee huomioida, että makro toistaa joka kerta vain samat tehtävät, eikä sen toimintaa voida käytön aikana ohjata. Visual Basic 2008 avulla tehtävät makrot tulee luoda käyttäen kuvassa 4.9 havaittavaa ”*Console Application* -formaattia. Kyseinen formaatti on sopiva, kun ohjelma ei tarvitse vuorovaikutusta käyttäjältä.



Kuva 4.5. Solid Edge Quick Access -palkki.

Makron voi suorittaa Solid Edgessä painamalla alkuun ohjelman vasemmassa yläkulmassa olevaa, Solid Edge logolla varustettua *Application* -nappia ja valitsemalla ”Run Macro”. Koska makron tarkoitus on helpottaa suunnittelutehtävää, jonka toistuvuus on suuri, suosittelen makrolle asetettavan pikanäppäimen helppoon sijaintiin, kuten kuvan 4.5 Työpiirrustus.exe tapauksessa on lisätty painike *Quick Access* -palkkiin.

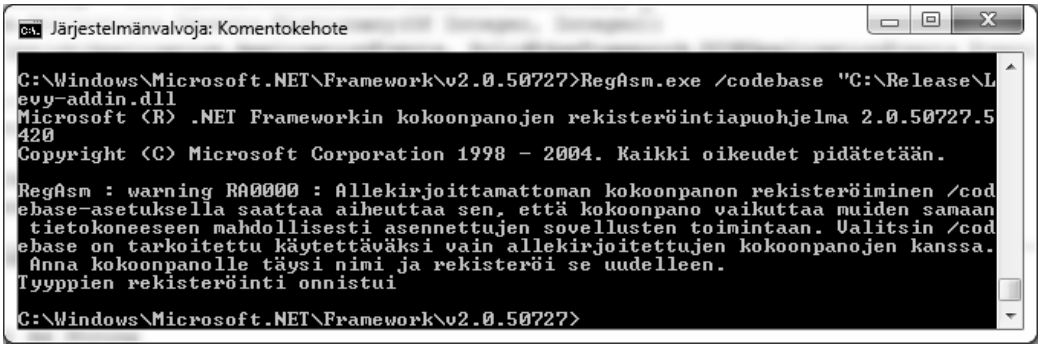
4.2.1.2 Add-In

Add-In, eli apuohjelma tai liitännäinen, toimii sisäisen sovelluksen periaatteella, eli se liitetään isäntäohjelmaan, eikä näin ollen toimi ilman käynnissä olevaa isäntäprosessia. Add-Inin vahvuutena on, ettei käyttäjän tarvitse poiketa isäntäohjelman käyttöliittymästä, sillä kaikki liitännäisen toiminnot voidaan toteuttaa samaan GUI:hin.

Ohjelmoinnin ja asennuksen kannalta Add-In on kolmesta esitellystä Solid Edgen automatisointikeinosta kaikkein työläin. Ohjelmoinnin alussa liitännäiselle on luotava uniikki globaali tunniste (*Globally Unique Identifier, GUID*), jota käyttäen se myöhemmässä vaiheessa rekisteröidään tietokoneelle. Koodin alussa implementoitava

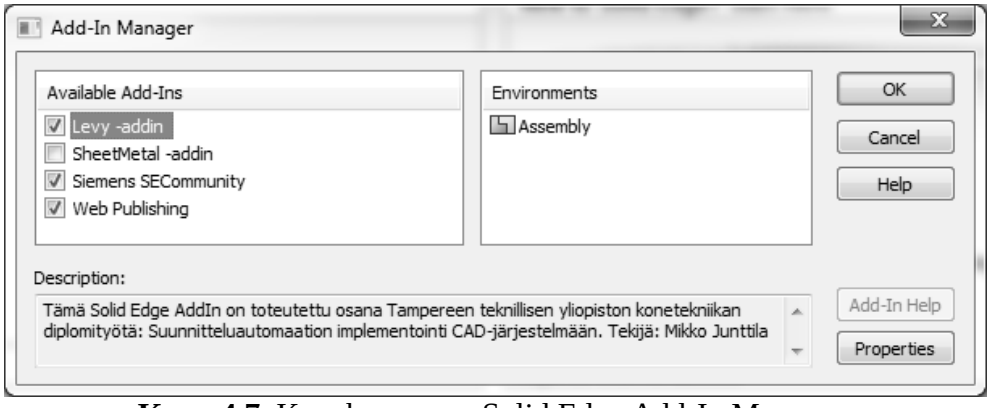
IsolidEdgeAddin (tai IsolidEdgeAddinEX) käyttöliittymä tuo mukanaan useita kymmeniä rivejä koodia. Rivit sisältävät paikat, joihin määritellään liitännäisen tapahtumat kun Solid Edge käynnistyy, määritetty mallinnusympäristö käynnistyy, ohjelma sammuu, sekä rekisteröinti että rekisteristä poistamiskomennot. Ohjelmalle luotavan uniikin tunnisteen lisäksi koodiin pitää lisätä GUID jokaiselle ympäristölle, jossa liitännäisen halutaan toimivan. Nämä ennalta määrätyt tunnisteet löytyvät liitteestä 2 ja ne ohjelmoidaan kohtaan, jossa määritetään rekisteröintikutsu: RegisterFunction.

Solid Edge Add-In on *DLL (dynamically linked library, dynaamisesti linkitetty kirjasto)* -formaatin tiedosto ja voit luoda tällaisen Visual Basic 2008:ssa käyttäen *Class Library* -projektiformaattia (havaittavissa kuvasta 4.9). Jotta apuohjelma toimii Solid Edgessä, tulee se lisätä tietokoneen rekisteriin käyttäen valitun Framework version Regasm.exe sovellusta. Alla olevassa kuvakaappauksessa näemme sekä komennon että palautteen luvun 5.2 Levy-addinin kirjaamiseksi rekisteriin.



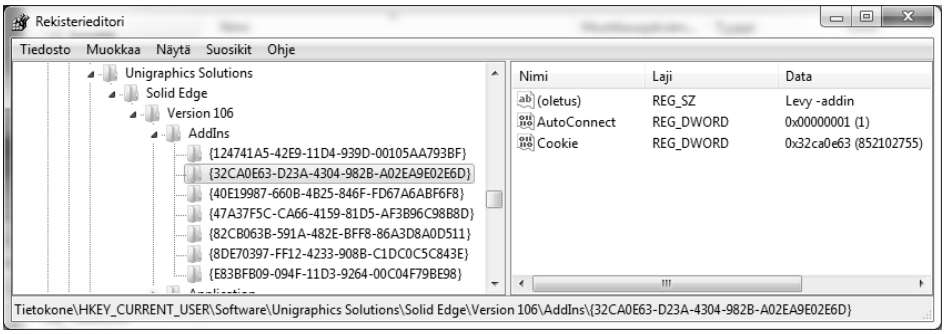
Kuva 4.6. Levy-addin.dll rekisteriin kirjaus.

Rekisteröinnin jälkeen Solid Edge voi ottaa yhteyttä liitännäiseen koodissa määritetyn mukaisesti, ohjelman käynnistyessä, käyttäjän tai erillisen ohjelman komennosta [29]. Rekisteriin kirjaaminen tulee suorittaa Solid Edgen mukaisessa bittijärjestelmässä. Eli vaikka Windows käyttöjärjestelmäni on 64bit niin SE:n ollessa 32bit, tulee liitännäinen kirjata rekisteriin 32bit RegAsm.exe:n avulla. Mikäli ohjelma on luotu oikein ja rekisteröinti onnistunut, näkyy apuohjelma Solid Edge Add-In Managerissa kuvan 4.7 mukaisesti. Kuvasta voidaan myös havaita ympäristöt (eng. Environments), joihin liitännäinen on rekisteröity.



Kuva 4.7. Kuvakaappaus: Solid Edge Add-In Manager.

Luodut apuohjelmat löytyvät myös Windows rekisterieditorin avulla alla olevan kuvakaappauksen mukaisesti.

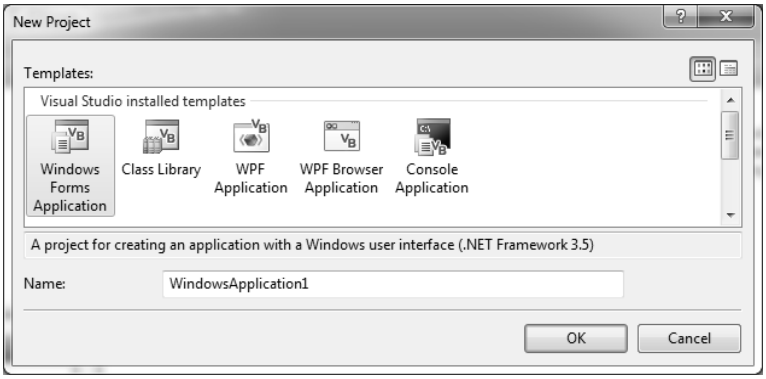


Kuva 4.8. Rekisterieditori - HKEY_CURRENT_USER / Software /

Rekisterieditorista voit havaita myös käytetyn uniikin rekisteröinti tunnuksen (GUID).

4.2.1.3 Aliohjelmat

Valitsin ensimmäiseksi Solid Edgen automatisoinnin case-esimerkiksi sovelluksen, joka toimii Solid Edgen ulkopuolella lähettäen komentoja isäntäohjelmalle. Ohjelman formaatiksi valitsin kuvan 4.9 mukaisen *Windows aplikaatio*n *(Windows Forms Application)*, sillä kyseiseen formaattiin voi luoda helposti käyttöliittymän, jonka avulla kommunikoida käyttäjän kanssa. Microsoft Visual Basic 2008 Express Editionin avulla sovelluksen luominen on helppoa. Ohjelmaa kirjoitettaessa sen toimintaa voi testata ja tallentaa ilman rajoituksia. Visual Basic 2008 tarjoaa useita työkaluja testausvaiheen ongelmatapausten selvittämiseksi.



Kuva 4.9. Kuvakaappaus: Microsoft Visual Basic 2008 Express Edition Forms

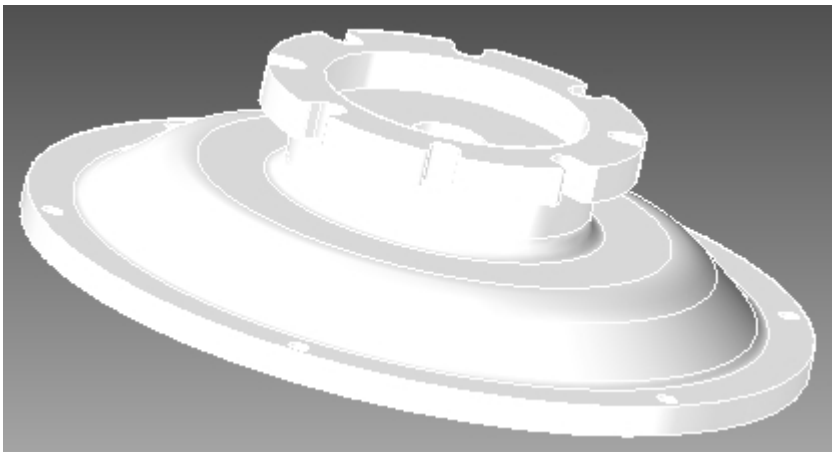
Windows applikaation vahvuuksiin lukeutuu käyttöliittymien luonnin lisäksi, ettei ohjelmaa tarvitse Add-In:n tapaan rekisteröidä tietokoneelle. Koska sovellus toimii makron tapaan ulkoisen sovelluksen periaatteella, sen voi käynnistää ilman isäntäohjelmaa ja halutessaan se voi kommunikoida usean isäntäohjelman kesken. Esimerkiksi aplikaatio voi suorittaa laskennan taulukkolaskentaohjelmassa ja käyttää tuloksia hyväksi CAD-ohjelmassa, eikä isäntäohjelmien tarvitse edes näyttäytyä käyttäjälle.

5. CASE-ESIMERKIT

Kuvitteellisessa suunnitteluketjussa ajatellaan yrityksen tuottavan luvun 3.1 esimerkin jäähdytysratkaisua. Tilaus tulee yritykselle kuvitteellisen myyntikonfiguraattorin kautta, jossa tilaaja syöttää jäähdytyskaappien määrän, niiden käyttöympäristön lämpötilan ja lämpötilan johon kaappien tulee viiletä. Systeemi laskee automaattisesti kaappien määrän ja lämpötilojen mukaan tarvittavien lauhduttimien ja moottorien määrän. Tässä vaiheessa tilaus tulee suunnittelijalle, joka hoitaa tilauksen mukaisten osien, kokoonpanojen ja dokumenttien luonnin käyttäen hyväkseen Solid Edge ohjelmaa ja yrityksen tarpeisiin luotuja tuotekonfiguraattoria sekä apuohjelmia.

5.1 Tuotekonfiguraattori

Tuotekonfiguraattori käyttää hyväkseen valmiiksi mallinnettuja 27:ää osaa, joiden avulla saadaan luotua 32 eri sähkömoottorikokoonpanoa. Osien mallinnuksessa on käytetty huomattavissa määrin harkintaa normaalitasojen suuntiin sekä koordinaatiston sijaintiin, jotta ne olisivat symmetrian kannalta ideaaliset. Valmiiden osien nimeämiskäytäntö on myös tarkoituksen mukaista. Alla oleva kuvateksti esimerkiksi kertoo osan olevan **etu**kansi **l**aippa kiinnityksellä, jonka materiaali on **al**umiini ja moottorin kokoluokka on **200**.



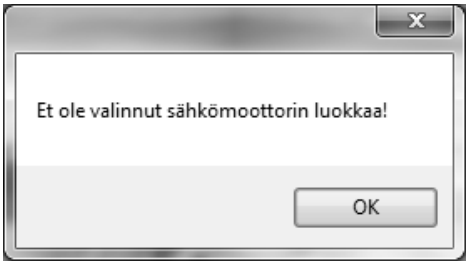
Kuva 5.1.1. etu_l_al_200

Suunnitteluautomaatiossa muuttujia olisi hyvä olla mahdollisimman vähän, ja niitä muuttujia, joita käyttäjältä otetaan vastaan, tulisi käyttää tehokkaasti. Tästä syystä osien mallinnuksessa on käytetty hyväksi erästä Solid Edgen keinoa paikoittaa osia kokoonpanoon. Paikoitustavan avulla osatiedoston origo voidaan liittää samaan sijaintiin ja orientaatioon kuin kokoonpanon origo. Näin säästetään ohjelmoitavien

rivien määrässä, joka on eduksi koodin ylläpidon ja selkeyden kannalta. Yleisesti paikoituksesta: älä käytä paikoituksessa elementtejä, kuten pintoja tai osia, jotka voivat liikkua tai eivät ole läsnä jokaisessa tapauksessa.

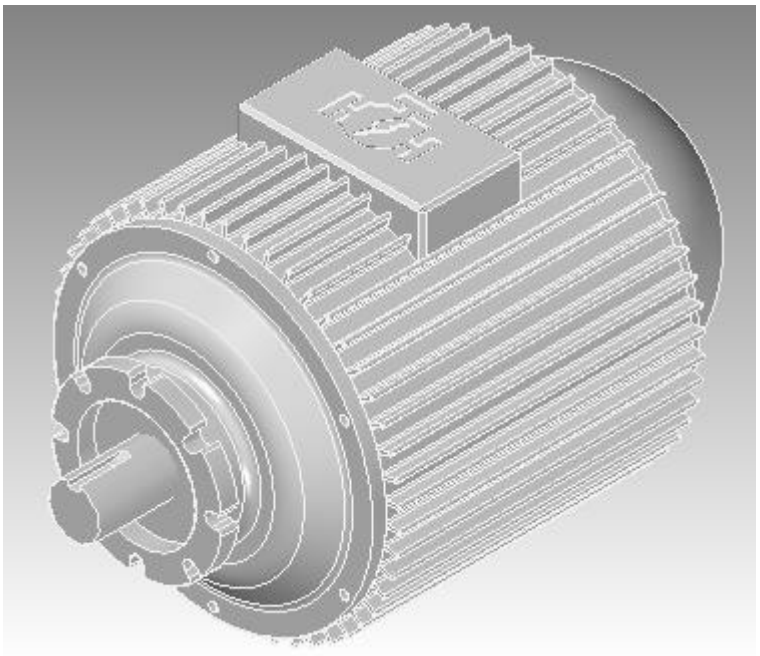
Kuva 5.1.2. Tuotekonfiguraattorin käyttöliittymä.

Tuotekonfiguraattorin käynnistyessä sovellus tarkastaa alkuun onko tietokoneella jo käynnissä oleva Solid Edge -prosessi käyttäen System.Diagnostics tyyppikirjaston Process.GetProcessesByName() metodia. Mikäli prosessi löytyy niin konfiguraattori ottaa tähän yhteyttä Marshal.GetActiveObject() metodilla. Jos prosessia ei löydy, niin uusi Solid Edge käynnistetään tuotekonfiguraattorin mukana, käyttäen Type.GetTypeFromProgID() ja Activator.CreateInstance() metodeja. Ohjelman käyttöliittymä (kuva 5.1.2) pyrittiin tekemään mahdollisimman yksiselitteiseksi ja jokainen valintavaihtoehto sisältää sisäisen ohjeistuksen, joka näytetään kun hiiri asetetaan vaihtoehdon päälle. Mikäli käyttäjä kuitenkin tekee virheellisiä tai puutteellisia syötteitä, ohjelma ilmoittaa tästä, kun käyttäjä painaa lopulta ”OK”. Tällöin tuotekonfiguraattori ei lähetä käskyjä Solid Edgeen, vaan palaa ennen napin painallusta edeltävään tilaan. Esimerkiksi kuvan 5.1.3 tapauksessa käyttäjä ei ole tehnyt valintaa ”Luokka:” alasvetovalikosta.



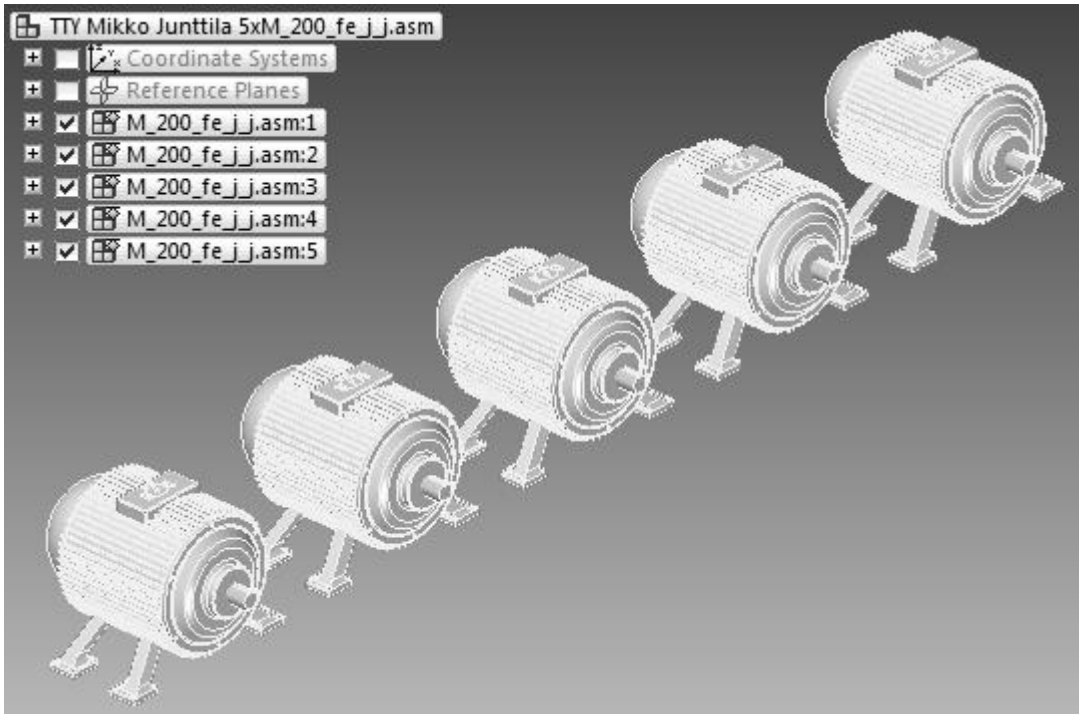
Kuva 5.1.3. Virheilmoitus.

Kun ohjelma on tarkastanut syötteet ja todennut ne virheettömiksi, luo tuotekonfiguraattori uuden Solid Edge -kokoonpanon, lisää valintavaihtoehtojen mukaiset osat, ja tallentaa kokoonpanon ennalta määrättyyn sijaintiin käyttäen nimeämispolitiikkaa: M-kirjain_moottorin luokka_materiaali_jäähdytysrivat_ kiinnityslaippa_kiinnitysjalka. Esimerkki onnistuneesta moottorikokoonpanosta kuvassa 5.1.4, josta voi havaita, että nimeämiskäytäntö ottaa huomioon myös jalan puuttumisen jättäen osan: _j nimen lopusta pois.



Kuva 5.1.4. M_180_fe_j_l

Kun moottorikokoonpano on luotu, kyseinen kokoonpano suljetaan ja tuotekonfiguraattori luo uuden kokoonpanon. Tähän uuteen kokoonpanoon tuodaan syötteen: ”Koneiden lukumäärä:” mukainen määrä sähkömoottoreita alikokoonpanona, ja jokainen moottori siirretään neljänkymmenen sentin päähän toisistaan, luoden kuvan 5.1.5 mukaisia kokoonpanoja. Kokoonpano tallennetaan nimeämiskäytännöllä: Yritys Nimi moottorien määrä x käytetyn moottorin nimi.



Kuva 5.1.5. TTY Mikko Junttila 5xM_200_fe_j_j

Ohjelman toiminta päättyy tarkastettuaan tuleeko sen sammua edellä kuvatut toiminnot suoritettuaan ”Sulje konfiguraattori?” -valintavaihtoehdon mukaisesti. Jos tuotekonfiguraattori valitaan jäävän toimintaan, voi käyttäjä sulkea Solid Edge -kokoonpanon painamalla ”Sulje kokoonpanot” -nappia ja/tai luoda uuden tuotekonfiguraation.

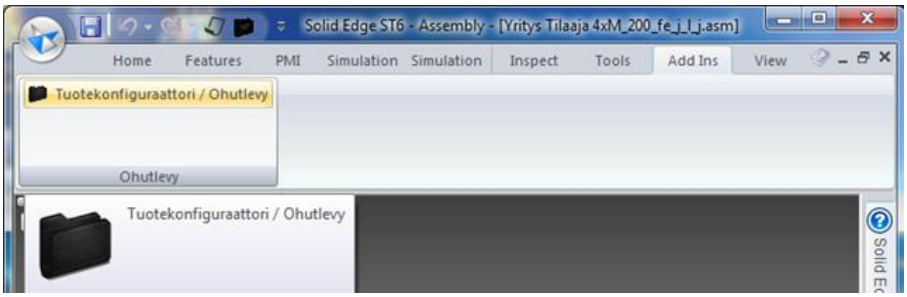
Mikäli edellä kuvattu suunnitteluautomaatio projekti toteutettaisiin asiakkaalle, olisi kannattavaa, että tuotekonfiguraattori suorittaisi myös lukujen 5.2 ja 5.3 vaiheet, sillä sen tarkoitus on nopeuttaa suunnittelijan työtä mahdollisimman paljon. Näin ei tässä tapauksessa toimittu, sillä halusin rakentaa esimerkeistä yhtenäisen jatkumon, ja tämän takia en sisällyttänyt yhteen ohjelmaan enempää vaiheita. Jo edellä kuvattu ohjelma toteuttaa kokoonpanon sekunneissa kun manuaalisesti tähän menisi vähintään useita, ellei kymmeniä, minuutteja.

Tuotekonfiguraattorin toiminta voidaan luokitella interaktiiviseksi, sillä se sisältää esimerkiksi sääntökannan osille, jotta ne ovat yhteensopivat, ja konfiguraattori ohjaa käyttäjän toimintaa sekä ohjein että sulkemalla käytöstä valintavaihtoehtoja, jotka eivät ole käytettävissä. Tästä esimerkkinä, jos käyttäjä valitsee syötteen ”Materiaali” arvoksi Alumiini, ei konfiguraattori näytä hänelle ”Jäähdytys” -valintavaihtoehtoon liittyviä objekteja. Tuotekonfiguraattorin sisältämä koodi löytyy liitteenä neljä.

5.2 Add-In ohutlevytyökalu

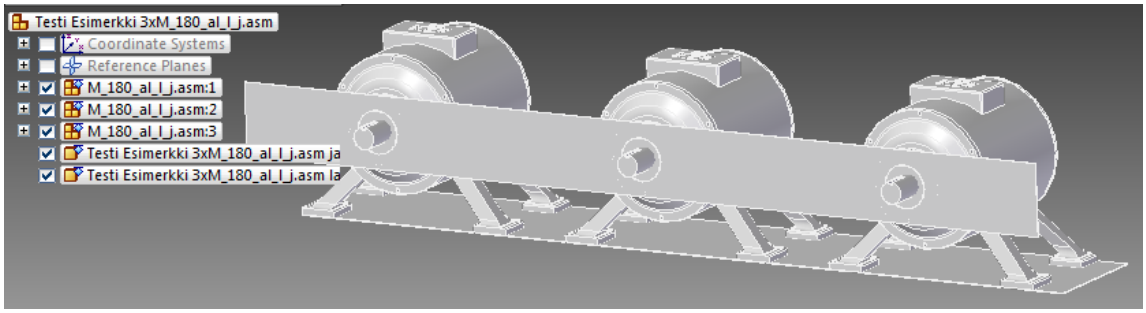
Tässä esimerkissä ohutlevyosia luova liitännäinen, joka rekisteröitiin luvussa 4.2.1.2, jatkaa kuvitteellista suunnitteluketjua siitä mihin luvun 5.1 tuotekonfiguraattori jäi luotuaan halutun määrän määrityksen mukaisia sähkömoottoreita. Kun Solid Edgeen

lisätään Add-In -liitännäisiä, ovat ne oletuksena käytettävissä ennalta määritetyissä mallinnusympäristöissä kuvan 5.2.1 mukaisesti Add-Ins -välilehdellä. Solid Edge -käyttöliittymä on käyttäjänsä muokattavissa ja halutessaan hän voi esimerkiksi siirtää lisäämänsä apuohjelman suorituspainikkeen haluamalleen välilehdelle, tai kuvan 5.2.1 mukaisesti myös Quick Access -palkkiin.



Kuva 5.2.1. Solid Edge kokoonpano - Liitännäiset - Ohutlevy

Add-In -liitännäisten luomisen haasteellisuudesta kertoo, että vaikka tämän esimerkin ohjelma on toiminnaltaan hyvin yksinkertainen, on sen koodi (liite 5) rivimäärältään kolmesta esitellystä suurin. Sisäisen sovelluksen periaatteella toimivat apuohjelmat voivat Microsoft Office -liitännäisten tapaan tuoda Solid Edgeen laajan määrän ikkunoita ja hallintatyökaluja, joiden avulla sovelluksen toimintaa määritetään. Tämän esimerkin käyttöliittymänä toimii vain kuvassa 5.2.1 esitetyt painikeet, jotka laukaisevat sovelluksen, sillä Add In -käyttöliittymien luominen vaatii käyttöliittymien sekä liitännäisten laajaa tuntemusta.

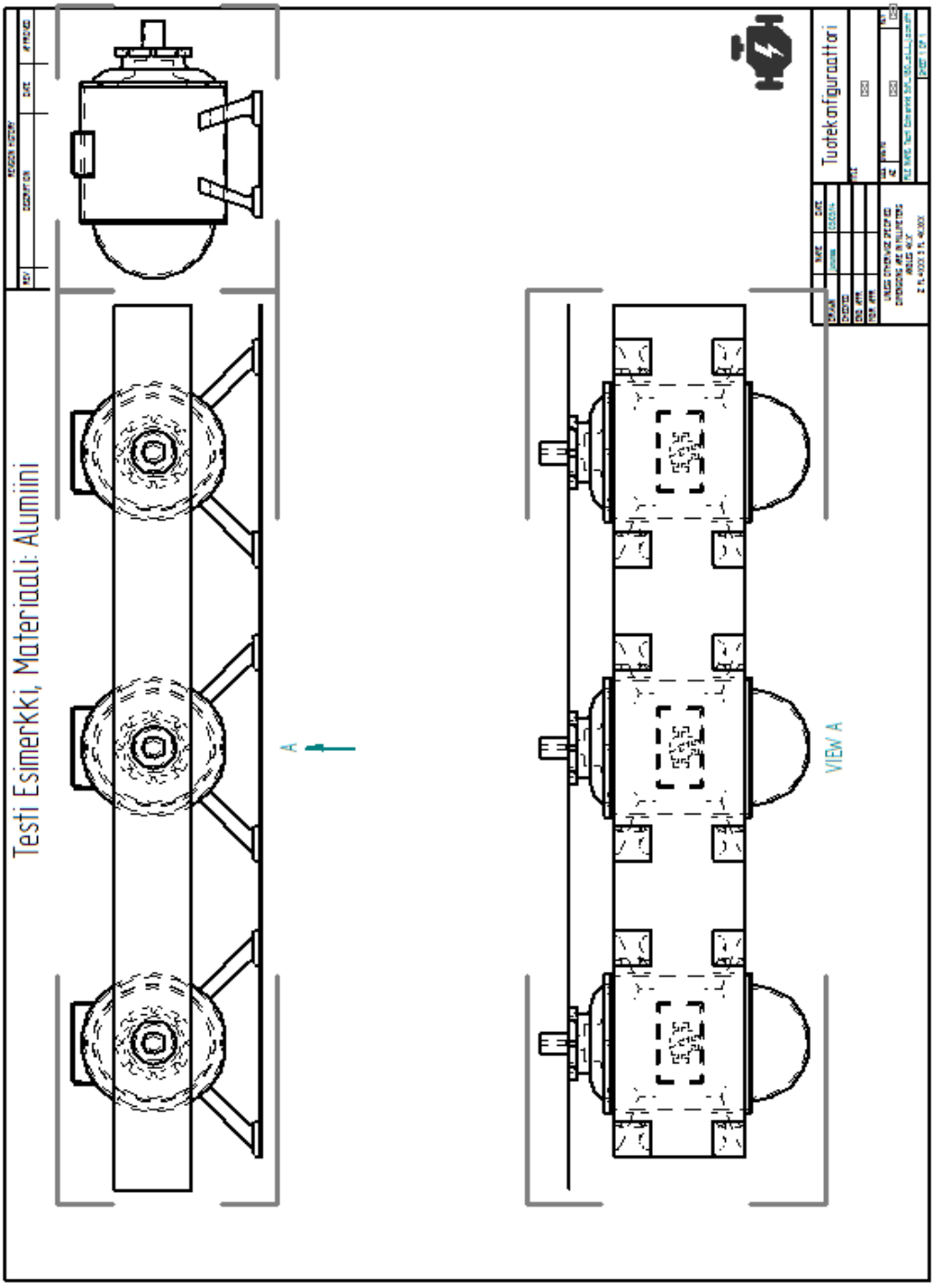


Kuva. 5.2.2. Jalka- ja laippakiinnityslevyt.

Esimerkin ohjelma luo avoinna olevaan sähkömoottorikokoonpanoon moottorien kiinnityksen mukaiset ohutlevykappaleet. Ohjelman käynnistyessä, se etsii kokoonpanon nimestä tiettyä rakennetta, jonka mukaan ohjelma ratkaisee montako sähkömoottoria kokoonpano sisältää, sekä ovatko käytetyt moottorit jalka- tai laippakiinnityksellä. Mikäli sähkömoottoreissa on laippakiinnitys, luodaan tämä ohutlevy samaan sijaintiin, kuin sen origo tulee olemaan kokoonpanossa. Jos moottoreiden kiinnitys on jalasta tuettu, luodaan uusi ohutlevy ja se paikoitetaan haluttuun sijaintiin käyttäen Move() -metodia. Molemmissa tapauksissa osatiedosto tallennetaan kokoonpanon yhteyteen ja suljetaan osatiedostot. Lopputuloksena kuvan 5.2.2 mukaisia kokoonpanoja, jossa moottorit on sidottu yhteen ohutlevykappalein.

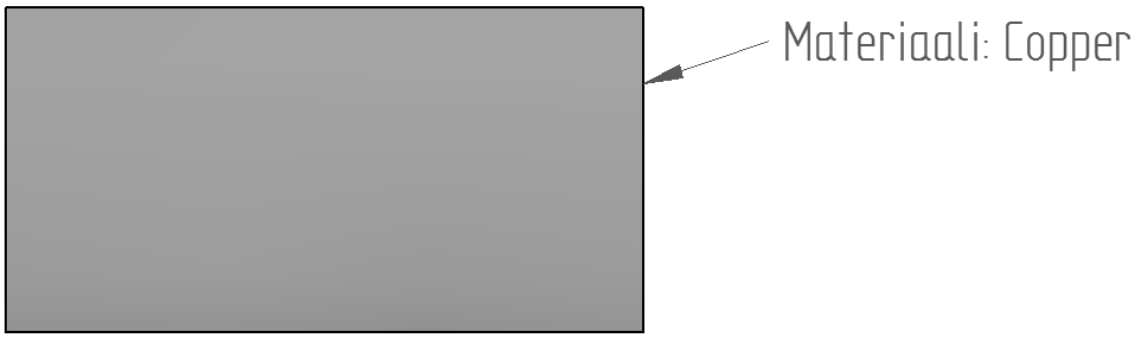
5.3 Kokoonpanopiirros-makro

Tämän case-esimerkin makro luo luvuissa 5.1 ja 5.2 oleville moottorikokoonpanoille kokoonpanopiirrokset. Suunnitteluprosessin automatisoinnin ei tarvitse olla laaja ja haastava projekti. Makron toiminta on hyvin yksinkertainen. Ohjelma laukaistaan kuvassa 4.2.4 esitellyn mukaisesti. Sovellus tulkitsee kokoonpanon nimestä moottorien lukumäärän, luo ja skaalaa kolme kuvantoa moottorien määrän mukaisesti. Alla olevassa kuvassa on makron luoma kokoonpanopiirros kolmen moottorin kokoonpanolle. Ohjelman koodin löydät liitteenä kuusi.



Kuva 5.3.1. Työpiirrustus.exe lopputulos.

Valitsin Solid Edgen työpiirroksen tarkoituksella makroesimerkin kohteeksi, sillä tämän avulla voin osoittaa, ettei kaikkea kannata jättää ohjelmoinnin varaan. Kuvan 5.3.1 oikeassa yläkulmassa olevassa tekstikentässä osa tekstistä on väriltään vaaleansinistä. Lisäksi tekstilaatikon yläpuolella on sähkömoottorin logo, jota olen käyttänyt luvun 5.1 tuotekonfiguraattorin ikonina. Nämä ovat hyviä esimerkkejä asioista, jotka on vaivattomampi ja nopeampi luoda suoraan käytettäviin piirustuspohjiin (eng. draft template). Vaikka koodin avulla voidaan luoda kuvan vasemmassa laidassa olevan mukaisesti tekstikenttiä (”Testi Esimerkki, Materiaali: Alumiini”), on koodin kirjoittaminen ja ylläpito työläämpää kuin muuttaa piirustuspohjia. Kuvassa vaaleansinisellä esiintyvät tekstit on luotu käyttäen Solid Edgen *kutsuja (eng. callout)*, joiden avulla Solid Edge hakee kutsun tiedot automaattisesti osa- ja kokoonpanotiedostoista. Esimerkkinä Solid Edgen kutsuista alla olevasta kuvasta on haettu osatiedoston materiaali (Copper), käyttäen kutsua: Materiaali: %{Material|GP}, jossa Materiaali on haluttu selvennysteksti, muoto %{} on viittaus, Material on kappaleen *ominaisuus (eng. Property)* ja |GP:llä viitataan kuvan kappaleeseen.



Kuva 5.3.2. Solid Edge Callout.

Kaikkien kolmen edellisissä luvuissa esiteltyjen automaattien luomat Solid Edge -mallit ovat täysin suunnittelijan muokattavissa, sillä vain harvoissa tapauksissa kaikki erikoistapaukset voidaan huomioida ennalta. Tästä syystä esimerkiksi kuvaan Työpiirrustus.exe lopputulos ei sisällytetty kokoonpanon dimensioita, sillä ei ole varmaa mille sidosryhmälle tämä kuva on tarkoitettu.

6. TULOKSET

Tässä diplomityössä esitetty kirjallinen selvitys tietokoneavusteisen suunnittelun automatisoinnista esimerkkeineen, on itsessään opinnäytteen tulos. Tavoitteena ei ollut luoda ohjelmointiopasta, sillä tähän tarkoitukseen löytyy tämän työn lähteinä käytettyjä englannin kielisiä oppaita.

Luvun 5.1 tuotekonfiguraattori onnistuttiin luomaan sille ennalta asetettujen vaatimusten mukaisesti. Ennakko-odotuksiin lukeutui interaktiivinen toiminta, joka varmistettiin sekä osille luodulla sääntökannalla että interaktiivisella käyttöliittymällä, joka paitsi muuttaa valittavana olevia komponentteja ennalta valittujen optioiden mukaan, myös tarkastaa syötteiden sisältämät virheet. Sovellus tukee myös opinnäytteen muuta kirjallista sisältöä, esimerkiksi soveltamalla sähkömoottoreihin modulaarisen tuotteen vaatimuksia. Esimerkin kohdetta harkittaessa otettiin huomioon tuotteen jako perus- ja valinnaisiin moduuleihin, jota sovellettiin sähkömoottorin tapauksessa lisäjäähdytyksen tarpeeseen. Lisämoduuli: jäähdytysrivat eivät ole moottorin toiminnan kannalta olennainen asia kuin äärimmäisissä olosuhteissa.

Ohutlevyjä tuottavan Add-In:n tavoitteena oli osoittaa uusien osatiedostojen luomisen mahdollisuus, sekä samanaikaisesti kokeilla eri keinoja paikoittaa osia kokoonpanoon. Apuohjelman myötä saatiin alustavaa tietoa Add-In -liitännäisten luomisen monipuolisuudesta, mutta samanaikaisesti onnistuttiin luomaan lähes automaattisesti toimiva sovellus, joka ei tarvitse toimiakseen käyttäjän syötteitä. Ainoa liitännäisen kautta opittu rajoite oli käyttöliittymän luonnin haasteellisuus.

Luvussa 5.3 esitelty makro on loistava esimerkki kuinka pieni panostus toistoa vaativaan tehtävään nopeuttaa suunnittelua huomattavasti. Makron toiminta jätettiin tarkoituksella lyhyeksi ja esimerkin-omaiseksi, sillä sovellusta ei kehitetty minkään yrityksen käyttöä varten. Mikäli sovellus olisi luotu jonkin yrityksen tarpeisiin, olisi Solid Edgen ohjelmointirajapinta mahdollistanut lisätä kuviin esimerkiksi mitoituksen ja osaluettelon, sekä niin halutessa toiselle sivulle (eng. sheet) kuvan yksittäisestä moottorista.

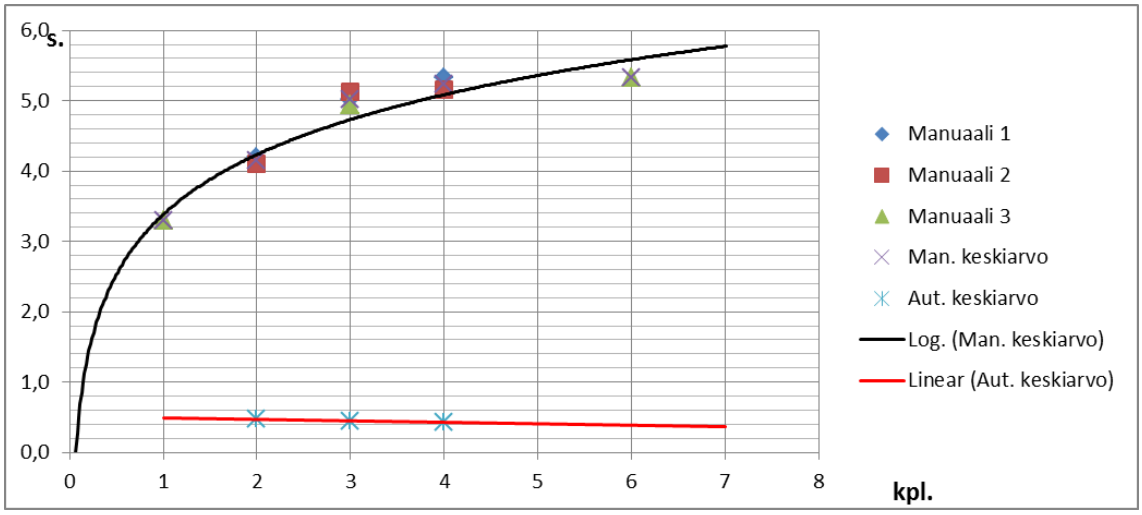
Taulukko 3. Kokoonpanojen testiajat.

lkm. (kpl)	Man. 1 (min.)	Man 2 (min.)	Aut. 1 (min.)	Aut 2. (min.)	Ero (%)
2	4,19	4,1	0,52	0,43	
3	5,01	5,12	0,51	0,37	
4	5,33	5,15	0,42	0,45	
Keskiarvo (min.)		4,82		0,45	90,7 %

Tämän opinnäytteen tuloksien analysointi vaatisi realistisessa implementointi tapauksessa laajaa testaamista ennen käyttöönottoa sekä työskentelyn aikaisia käyttäjäkokemuksia, ennen kuin työn onnistumista voitaisiin arvioida. Sillä case-esimerkit ovat täysin fiktiota, eikä niiden toimintaa ole optimoitu yhdistämällä kaikkien kolmen automaatin toiminta yhden sovelluksen alaiseksi, ei laajoista testeistä ole konkreettista hyötyä. Tästä huolimatta suoritin taulukon 3 mukaisen ryhmän testikokoonpanoja osoittaakseni kuinka yksinkertaisilla tehtävien automatisoinneilla suunnittelijan työtä tehostetaan. Testin tarkoituksena oli tutkia kuinka suuri ajallinen ero syntyy manuaalisesti ja automaattisesti suoritetulla työllä. Taulukon 3 sarakkeessa lkm. on sähkömoottorien määrä, sarakkeissa Man. X ovat manuaalisten ja Aut. X automaatin suoritteiden kestot. Molempien tapausten suoritus oli sama:

- ohjelma(t) käynnistettynä, mutta tyhjänä syötteistä
- ajanotto käyntiin
- uusi kokoonpano ja sähkömoottorin osien paikoitus
- nimellä tallennus
- uusi pääkokoonpano, johon lisätään haluttu määrä moottorikokoonpanoa
- nimellä tallennus
- luoda kuvan 5.3.1 mukainen kokoonpanopiirustus tekstikentän kera
- nimellä tallennus.

Ajanotto päätettiin, kun Solid Edge oli tallennuksen jälkeen vapaa ottamaan uusia komentoja vastaan. Kuten kuvaajasta yksi voidaan havaita, kasvaa manuaalisen työn kesto lähes logaritmisesti moottorien määrän mukaisesti. Automaatin tekemänä työn kesto rajoittuu valintavaihtoehtojen kompeksisuuden ja tietokoneen laskentatehon myötä lähes vakioarvoon. Tämän myötä olen todistanut opinnäytteen luvun 3.3.1.1 väitteen: onnistunut suunnitteluautomaation implementointi säästää 90% suunnittelijan työskentelyajasta.



Kuvaaja 1. Kokoonpanojen testiajat.

Testistä jätettiin tarkoituksella pois luvun 5.2 liitännäisen osio, sillä otettaessa osien luonti mukaan testiin, ero olisi vielä suurempi. Testitapauksia luodessa jätin ottamatta huomioon myös manuaalisessa työssä tehdyt virheelliset ajanottotapaukset. Virheellisellä testitapauksella tarkoitan käyttäjän tekemiä virheitä, kuten väärän napin painallus tai osan tuonti kokoonpanoon. Virheitä osui näinkin yksinkertaisessa tehtävässä usein. Rutiinin mukaisessa suunnittelutyössä eniten aikaa vievät, ei itse rutiinitehtävä, vaan niissä tehtyjen virheiden korjaaminen. Mitä myöhemmin virhe huomataan, sitä enemmän aikaa vievää sen korjaaminen on. Pahimmassa tapauksessa virhe syö myös muita resursseja ajan lisäksi. Näistä seikoista huolimatta halusin saada testiin mahdollisimman vertailukelpoiset tulokset. Tästä syystä käytin testin manuaalisessa suorituksessa hyväkseni myös kaiken Solid Edge asiantuntijan taitoni, jotta suoritus olisi mahdollisimman nopea. Virheettömistä suoritteista huolimatta, testin mukaan ero sadan erillisen kokoonpanon ja piirroksen luonnin välillä on liki suunnittelijan työpäivän mittainen (ero: seitsemän tuntia ja seitsemäntoista minuuttia).

Solid Edgen tyyppikirjastojen yhteydessä työskennellessä huomattiin eräs ongelma. Tarkoituksena oli luoda ohjelma, joka käyttää samanaikaisesti sekä vakio (constant) että piirros (draft) tyyppikirjastoa. Kyseisen ohjelman oli tarkoitus toteuttaa ohutlevykappaleeseen taitos käyttäen hyväkseen vakio tyyppikirjaston suuntavakioita: ”igRight”. Myöhemmässä vaiheessa ohjelman oli tarkoitus luoda piirroksen pääkuvannosta otettava sivukuvanto käyttäen hyväksi piirros tyyppikirjaston vakiota: ”igFoldToRight”. Tuntemattomasta syystä vakio tyyppikirjaston referenssi yliajaa piirros tyyppikirjaston komennot. Näin ollen piirroksien luonnissa käytettävät vakiot eivät toimi. Ongelman voi kiertää luomalla erikseen pääohjelma, joka kutsuu erillistä makroa suorittamaan osan tehtävistä, sillä eri sovellukset voivat käyttää eri tyyppikirjasto referenssejä.

Suurimmiksi ongelmiksi, jotka ovat Solid Edgen automaation esteenä tai rajoitteena, koettiin lähinnä ohjeiden puutteellisuus. Esimerkiksi edellä mainittujen ”ig”-alkuisten suuntavakioiden käyttöä ei ole selkeästi dokumentoitu, vaan testaus oli lähes ”yritys-erehdys” -pohjaista. Vaikka työn laajuuden kannalta olisi ollut mielekästä onnistua luomaan Solid Edgen käyttöliittymässä toimiva käyttöliittymä Add-In -liitännäistä varten, se oli työn tekoon käytettävän ajan, rajallisen ohjelmointikokemuksen ja dokumentaation myötä lähes mahdotonta. Toisaalta, tämä todistaa Electronic Data Servicesin väitteen, että: ”joillakin menee vuosia ennen kuin ovat ”sinut” Solid Edge ohjelmointirajapinnan kanssa”[28]. Samasta syystä, vaikka on mahdollista selvittää tietyn osan tai alikokoonpanon lukumäärä pelkästä kokoonpanotiedostosta, ohjelmoinnin vasta-alkajana käytin useissa tilanteissa muuttujien selvittämiseksi tiedoston nimeä. Tämä on samalla kannanotto yritysten suunnittelutiedon nimeämiskäytäntöön. Suunnittelunimikkeiden ylläpito on usein pienissä yrityksissä sekavaa. Asiaan kannattaa kiinnittää huomiota, sen mahdollistaessa paljon ja vähentäessä vielä enemmän päänvaivaa.

Diplomityön eräänlaisena tuloksena voidaan nähdä sen toistettavuus. Sillä kyseessä on opinnäytetyö, voidaan positiivisena huomiona mainita, että kaikki työssä käytetyt

ohjelmat ja oppaat (ohjelmointikielen opettavaa kirjaa lukuun ottamatta) ovat opiskelijoille maksuttomia. Suosittelen teknillisten alojen opiskelijoille, joilla on mielenkiintoa, muttei allekirjoittaneen tapaan lahjoja tai kärsivällisyyttä ohjelmointiin, tutustumista tässä oppaassa pääpainon saaneeseen Visual Basic .NET ohjelmointikieleen.

Päätelmät

Makrot ovat hyvä tapa automatisoida tapahtumia, joissa ei tarvita vuorovaikutusta käyttäjän kanssa. Tapauksissa, jossa käyttäjältä vaaditaan interaktiivisuutta, on vaihtoehtoja kaksi: ulkoinen sovellus, joka toimii omassa käyttöliittymässään, tai Solid Edgen käyttöliittymään integroitava Add-In. Vaikka nykytrendi on integroida kaikki toiminnot samaan käyttöliittymään, esimerkiksi Solid Edgen ”isoveljen” NX CADin käyttöliittymässä toimivat lisäosat kuten CAM; mikäli ohjelmoijalla ei ole laajaa kokemusta käyttöliittymistä ja niiden yhteyteen ohjelmoinnista, suosittelen luomaan erillisen ohjelman. Ulkoisen sovelluksen toimintaperiaatetta tukevat huomiot:

- Sovellusta ei tarvitse rekisteröidä erikseen jokaiselle tietokoneelle
- Ohjelma voidaan suorittaa ilman käynnissä olevaa isäntäohjelmaa
- Käyttöliittymän hallinta on helpompaa (erityisesti Visual Basic .NET ohjelmointikielellä)
- Sovelluksien luomiseen on ilmaisia ohjelmointiympäristöjä kuten Microsoft Studio Express
- Sovellus on yhteensopiva sekä 32- että 64-bittisen Solid Edge ja Windowsin kanssa.

Suunnittelun automatisointia pohtiessa tulee ottaa huomioon myös mitä ei voida ja kannata automatisoida. Ensijaisesti automaatio kannattaa kohdistaa asioihin, jotka ovat suunnittelijalle tylsiä ja rutiininomaisia tehtäviä, joita hän toistaa useita kertoja päivässä. Huomiota kannattaa kiinnittää myös työn mielekkyyteen. Mikäli suunnittelija nauttii tehtävästä, eikä sen automatisointi tuo suuria etuja, voidaan tehtävä jättää automaation ulkopuolelle, jolloin suunnittelijan keskittyminen tehtävään pysyy yllä. Haluan painottaa, että automaattien luomat osat, kokoonpanot ja piirrokset ovat täysin normaaleja Solid Edge -tiedostoja. Näin ollen suunnittelija voi muokata niitä jälkikäteen täysin normaaliin tapaan Solid Edgeä käyttäen.

Suuria kokonaisuuksia ja konfiguraattoreita automatisoitaessa kokonaisuus on järkevää koota yksittäisistä makroista, joilla on selkeä oma tehtävänsä. Tällaisia makroja ovat esimerkiksi: suunnittelutiedon käsittely, piirustuksen luonti, tiedoston tallennus haluttuun formaattiin jne. Myös koodin jako moduuleihin auttaa ohjelmointityössä. Selkeistä moduuleista seuraa, että koodin ylläpito helpottuu, kun muutokset kohdistuvat tiettyyn osioon, eikä muutostyö vaikuta ohjelman muuhun toimintaan. Mikäli yrityksessä on rajallinen ohjelmiston ylläpitobudjetti tai

ohjelmointikokemus, kannattaa suunnittelun automatisoinnissa harkita valmisohjelmia. Niiden tuomat edut: poistunut ohjelmointiosaamisen tarve, nopea käyttöönotto, helppo ja yksinkertainen ylläpito voivat säästää huomattavia resursseja.

Mielestäni Visual Basic .NET oli erittäin hyvä valinta opinnäytteessä käytettyjen esimerkkien ohjelmointikieleksi. Kieli on laajasti käytetty ja Internetistä löytyy paljon ohjelmointiohjeita, joiden avulla ongelmiin löytyy apua. Mikäli Solid Edgen yhteyteen haluaa ohjelmoida oman apuohjelman, kannattaa tutustua Spy for Solid Edge -sovellukseen [25] sen yksinkertaistaessa oliomallin opiskelua.

Jatkotoimet

Laajoista käyttöönottoa edeltävistä testeistä huolimatta sovelluksissa on lähes poikkeuksetta parantamisen varaa. Niiden toiminnassa voi olla virheitä, tai toimintaan kohdistuva tarve on muuttunut. Tästä syystä on hyvä pitää sovelluksen toiminta yksinkertaisena. Näin käyttäjä ei turhaudu, ja samalla yksinkertaisuus palvelee koodin ylläpidettävyyttä ja sovelluksen päivitystä. Kun sovellusta on käytetty jonkin aikaa, on hyvä kerätä ohjelman käytettävyydestä palautetta ja pyrkiä kehittämään sitä.

Seuraavissa luvuissa käsittelen hieman seikkoja, jotka alkoivat kiinnostamaan itseäni tämän työn pohjalta, sekä näkökulmia, joita tulee ottaa huomioon suunnitteluautomaation implementointia harkittaessa.

Nimikkeiden hallinta

Työskennellessäni CAD-asiantuntijana kohtasin usein asiakkaita, joiden suunnittelutiedossa nimikkeiden ja revisioiden hallinta oli puutteellista. Pienet yritykset turvautuvat usein tässä asiassa taulukkolaskentaohjelmien tai muistin varaan. En ota kantaa nimikkeiden nimeämiskäytäntöön, sillä aiheesta on kirjoitettu kokonaisia diplomitöitä, mutta tämän opinnäytteen myötä huomaan, että nimikkeiden uniikki numerointi ja revisioiden seuranta on yksinkertaista automatisoida. Asia vähentäisi paljon Solid Edge -käyttäjien päänvaivaa, ja manuaalisesti ylläpidettävät Excel-taulukot voisi unohtaa. Mikäli yritys luo käyttöönsä esimerkin mukaisen tuotekonfiguraattorin, kannattaa sovelluksen yhteyteen rakentaa nimikkeistön ja revisioiden hallintaan tarkoitettu osio, ja implementoida tämä samalla myös muuhun suunnittelutyöhön.

Mikäli yrityksessä on useita suunnittelijoita, ei suunnittelutiedon oikeellisuutta voida usein varmistaa pelkällä revision hallinnalla. Useissa kohtaamissani yrityksissä useilla henkilöillä on käyttöoikeudet verkkolevyllä olevaan suunnittelutietoon, jota he siirtävät tarpeen mukaan tietokoneelleen ja takaisin verkkopolkuun. Mikäli tiedon kulkua ei seurata, on mahdollisuus suunnittelutiedon samanaikaiseen käyttövirheeseen. Mikäli nimikkeitä ja revisioita hallitaan edellä kuvatun mukaisella sovelluksella, tulee sovelluksen myös samanaikaisesti seurata suunnittelutiedon liikettä, jottei kaksi suunnittelijaa tee yhdenaikaisia muutoksia, josta toinen yliajetaan tietoa palautettaessa. Virheen mahdollisuus poistetaan usein yrityksistä PDM-järjestelmällä. Tuotetiedon hallintajärjestelmä tuo mukanaan omat haasteensa, jotka osoitan luvussa integraatiot.

Myyntikonfiguraattori

Tässä opinnäytetyössä tutkittiin myyntikonfiguraattoreiden toimintaa vain teoreettisella tasolla. Mikäli yritys haluaa tehostaa suunnitteluprosessiaan paljon, ja sen tuote soveltuu konfiguroitavaksi, kannattaa harkita tuoko internetissä toimiva myyntikonfiguraattori yritykselle lisämyyntiä tai uusia asiakkaita. Seuraavassa asioita, joita Mika Aho listaa konferenssipaperissaan hyvän konfiguraattorin ominaisuuksiksi [6]:

- Ratkaisuavaruuden ja riippuvuussuhteiden esittäminen
- Kokeilemisen kautta oppiminen (trial-and-error -tyyppinen ongelmanratkaisu)
- Palaute (simulaatio, virtuaalinen prototyyppi)
- Selkeä, looginen rakenne ja hyvä käytettävyys
- Kaksisuuntainen IT-integraatio; tuotetiedot PDM:ään ja automaattinen ylläpidettävyys PDM:stä
- Hinta ja toimitusaikatiedot näkyvillä
- Teknisesti nykyaikainen
- Herättää ja luo luottamusta

Allekirjoittanut koki uuden ohjelmointikielen opiskelun mieleiseksi, ja sen pohjautuessa .NET teknologiaan, on mielenkiintoista tutkia miten Internet-sovellukset toimivat ja miten Visual Basic .NET soveltuu internetpohjaisten ohjelmien luontiin.

Integraatiot

Tietokoneavusteisen suunnittelun eräänä haasteena on sen tuottaman tiedon määrä ja sen hallinnointi. Pienissä yrityksissä suunnittelutietoa hallinnoidaan usein muistin varassa. Tiedostot tallennetaan haluttuun verkkopolkuun tai kovalevylle, ja nimike- sekä revisiotiedoista pidetään taulukkoa. Tällainen tiedonhallinta toimii usein nimeämällä tietyt tuotteet projektien tai asiakkaiden mukaan. Resurssienhallintaan perustuva dokumenttien hallinta on riskialtista, sillä se ei sisällä linkkejä osa- ja kokoonpanotiedostojen välillä, ja näin ollen yhden dokumentin siirtyessä kansiosta toiseen tai hävitessä, ei alkuperäinen rakenne enää toimi. Tuotetiedon hallinta vaikeutuu ajan myötä tuoterakenteiden monimutkaistuessa ja vaatimusten määrän ja laadun kasvaessa. Nämä ongelmat on usein ratkaistu tuotetiedonhallinta (Product Data Management, PDM) ohjelmistoilla.

Siemens PLM Softwaren ratkaisu tuotetiedonhallintaan on tuotteen elinkaaren hallintajärjestelmä (Product Lifetime Management, PLM) Teamcenter. Järjestelmä mahdollistaa sekä tuote- että suunnittelutiedon hallinnan yhdestä sijainnista, samanaikaisesti ratkaisten yksiselitteisen nimikkeiden ja revisioiden hallinnan. Kaikki tuotekohtainen tieto, attribuutit ja tiedostot on sijoitettu samaan järjestelmään. Teamcenterin avulla palvelimelle sijoitettu informaatio voidaan helposti jakaa käyttäjiensä kesken globaalisti, mahdollistaen yhdenaikaisen suunnittelun. Tämä

helpottaa tuotteisiin liittyvää päätöksentekoprosessia, selkeyttää käyttäjän työnkuvaa ja kannustaa käyttämään tuotekohtaista dataa uudelleen.

Yritysten automatisoidessa suunnitteluohjelmiaan, kannattaa projektin alussa harkita myös tuotannonohjausjärjestelmien (Enterprise Resource Management, ERP) mukaan ottaminen. ERP -järjestelmien toimittajia on lähes rajattomasti, ja ne tarjoavat eri variaation työkaluja. Osa ERP:stä sisältää myös jonkin asteisen PDM-ratkaisun, ja usein järjestelmä hallinnoi myös nimikkeitä sekä revisioita. Mikäli yritys ei ole implementoinut toiminnanohjausjärjestelmää, on tämä hyvä aika tutustua aiheeseen.

Solid Edgeä automatisoidessa Teamcenter integraatio on jo olemassa, sillä tuotteet ovat saman yrityksen omaisuutta. Solid Edgen ohjelmointikirjasto tukee myös tätä integraatiota (Solid Edge Embedded Client, SEEC), muttei minulla ollut tämän opinnäytteen yhteydessä mahdollisuutta tutkia integraation automatisointia. Mikäli PDM:n tarjoaa jokin muu ohjelmistotalo, on integraation olemassaolo ja sen automatisointi täysi kysymysmerkki. Sama tilanne koskee ERP-toimittajia; mikäli toimittaja ei ole integroinut käytettävää sunnitteluohjelmistoa järjestelmäänsä, on sen luominen itse liki mahdotonta. Teamcenter puolestaan voidaan integroida lähes mihin tahansa ERP-järjestelmään, ja järjestelmien nimikkeiden hallinta sovittaa yhteen. Mikäli olemassa olevaa integraatiota ei ole, voidaan se luoda Teamcenter -jälleenmyyjän toimesta. Sekä ERP- että PDM-järjestelmien impelentointi tuo valitettavasti mukanaan suuren hintalapun, todennäköisesti suuremman kuin Solid Edgen automatisointi. Mutta mikäli automatisointi on tehty ennen järjestelmien vaatimuksien tuntemusta, eivät automaatit tai konfiguraattorit täydellä varmuudella toimi enää uuden järjestelmän käyttöönoton jälkeen.

Loppusanat

Olen työn tuloksiin pääsääntöisesti tyytyväinen. Harmilliseksi jäi, ettei työtä voitu tuottaa jonkin yrityksen tarpeisiin, sillä aihe oli todella mielenkiintoinen ja tällöin ohjelmat olisivat saaneet kunnon vaatimusmäärittelyn, eivätkä sovellukset olisi jääneet vain luotujen tapaisesti esimerkkien asteelle. Olisi ollut mielekästä sisällyttää tuotekonfiguraattori esimerkkiin myös Excel -taulukkolaskentaa. Tämä osio ohjelmoinnista jätettiin kuitenkin pois, sillä se ei rajoitu Solid Edgen myötä millään keinoa. Samoin Add-In -esimerkkiin olisi ollut mielekästä sisällyttää interaktiivisia komponentteja, jotka toimivat Solid Edge käyttöliittymässä, mutta Add-In käyttöliittymän luonti koettiin liian haastavaksi. Hyvän käyttöliittymän luominen on ”oma taiteen lajinsa” ja kuten työssä mainitsin: suunnittelun automatisoinnin tavoitteena ei tule olla kaikki tai ei mitään, vaan automatisoidaan se mitä pystytään!

Työn tekeminen on ennen kaikkea tuottanut hyötyä allekirjoittaneelle, sillä sen myötä saavutettu uusi osaaminen ohjelmointikielen kautta on varmasti hyödyksi työelämässä. Valitettavaksi jäi, että tuotannollisten ja taloudellisten perusteiden myötä päättyneen työsuhteen takia työn alkuperäinen tarkoitus lisätä IDEAL PLM:n sisäistä osaamista, ei tullut saavutetuksi.

LÄHTEET

[1]. Siemens Automation, kotisivut. [WWW]. [Viitattu 11.2.2014]. Saatavissa: http://www.automation.siemens.com/mcms/automation/en/Pages/automation-technology.aspx

[2]. Siemens Product Lifecycle Management Software Fact Sheet. [WWW]. [Viitattu 24.4.13]. Saatavissa: http://www.plm.automation.siemens.com/en_us/about_us/facts_philosophy/fact_sheet.shtml

[3]. IDEAL PLM, kotisivut. [WWW]. [Viitattu 20.8.13]. Saatavissa: http://www.ideal.fi/

[4]. Autio, A. ja Hasari, H. 1999. Koneenpiirustus ammattikorkeakouluille ja teknisille oppilaitoksille. 1. Uudistettu painos. Helsinki, Otava.

[5]. Kuva 3.1. 3Dconnexion 3D-hiiret. [WWW]. [Viitattu 10.12.2013]. Saatavissa: http://www.3dconnexion.com/

[6]. Mika Aho, 2008. Tietovarastointiratkaisut massaräätälöinnin kongiguraattoreiden tukena. Konferenssipaperi, Tampere University of Technology. [Viitattu 1.9.2013]. Saatavissa: http://www.drmika.com/download/Aho_-_Tietovarastointiratkaisut_massaraataloinnin_konfiguraattoreiden_tukena.pdf

[7]. Aimo Pere, 2004. Muunnokset Jarno Vänni, 2009. Koneenpiirustus 1 & 2. 10. painos. Espoo, Kirpe Oy.

[8]. Mikell P. Groover, 2008. Pearson International Edition: Automation, Production Systems, and Computer-Integrated Manufacturing. Third Edition. New Jersey, Pearson Education Inc.

[9]. Jukka Korpela. Pienehkö sivistyssanakirja. [WWW]. [Viitattu 20.8.2013]. Saatavissa: http://www.cs.tut.fi/~jkorpela/siv/laaja.html

[10]. Hietikko, Esa. 2007. Autodesk Inventor. Helsinki, Readme.fi.

[11]. Wikipedia. [WWW]. [Viitattu 27.8.2013]. Saatavissa: http://fi.wikipedia.org

[12]. EDA, Edge Design Configurator, Solid Edge kotisivut. [WWW]. [Viitattu 29.8.2013]. Saatavissa:

http://www.plm.automation.siemens.com/en_us/products/velocity/solidedge/applications/edge-design-configurator.shtml

[13]. Ahoniemi l., Mertanen M., Mäkipää M., Sievänen M., Suomala P. ja Ruohonen M., 2007. Massaräätälöinnillä kilpailukykyä. Helsinki, Teknologiainfo Teknova Oy.

[13]. C-Advice Oy, Suunnittelun automatisointi. [PDF]. [Viitattu 9.11.2013]. Saatavissa: http://www.c-advice.com/Dokumentit/C-Advice_Autom.pdf

[14]. Power-Plaza Gear Unit Configuration, Kumera Oy. [WWW]. [Viitattu 10.12.2013]. Saatavissa: https://www.power-plaza.com/KumeraPowerPlaza/PP_Configuration.aspx

[15]. Mertanen, M., 2007. Massaräätälöinnin nykytila suomalaisissa teollisuusyrityksissä. Diplomityö. Tampereen teknillinen yliopisto, tuotantotalouden osasto, 104 sivua.

[16]. Kuva 3.3.3. Profin Oy, Myyntikonfiguraattori. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://www.profin.fi/suunnitteluohjelma

[17]. Salminen, P. 1990. Tuotteiden ja toiminnan laadun kehittäminen. Helsinki: Metalliteollisuuden Kustannus Oy.

[18]. Antti Vanha-Viitakoski. 2011. Suunnitteluautomaatin kehittäminen teollisuuspuhaltimien suunnitteluun. Opinnäytetyö, Seinäjoen ammattikorkeakoulu, Kone- ja tuotantotekniikan koulutusohjelma, Tekniikan yksikkö. [Viitattu 9.12.2013]. Saatavissa: http://publications.theseus.fi/bitstream/handle/10024/28639/Vanha-Viitakoski_Antti.pdf?sequence=1

[19]. Solid Edge Applications, Siemens Product Lifecycle Management Software. [WWW]. [Viitattu 10.12.2013]. Saatavissa: http://www.plm.automation.siemens.com/en_us/products/velocity/solidedge/applications/index.shtml

[20]. Siemens Product Lifecycle Management Software Press Release. 6.10.2009. Siemens PLM Software Purchases Rulestream Engineer-to-Order (ETO) Software Technology and Brand Assets. [Lehdistötiedote]. [Viitattu 12.8.13]. Saatavissa: http://www.plm.automation.siemens.com/en_us/about_us/newsroom/press/press_release.cfm?Component=87264&ComponentTemplate=822

[21]. Siemens Product Lifecycle Management Software. .NET Programmer’s Guide Solid Edge with Sychronous Technology API. [PDF]. [Viitattu 12.2.2014]. Saatavissa:

http://www.plm.automation.siemens.com/zh_cn/Images/Solid_Edge_API_tcm78-125829.pdf

[22]. Thomas B. Sheridan. 2002. Humans and Automation: Design and Research Issues. New York: Wiley; Santa Monica (CA): Human Factors and Ergonomics Society, cop.

[23]. James Foxall. 2008. Sams Teach Yourself Visual Basic 2008 in 24 Hours. 1. painos. Sams, Indianapolis, Indiana, 46240 USA

[24]. Theseus-julkaisuarkisto - ammattikorkeakoulujen opinnäytetyöt ja julkaisut verkossa. [WWW]. [Viitattu 17.2.2014]. Saatavissa: https://www.theseus.fi/

[25]. CodePlex. [WWW]. [Viitattu 17.2.2014]. Saatavissa: http://www.codeplex.com/

[26]. Siemens PLM Solid Edge Community. [WWW]. [Viitattu 17.2.2014]. Saatavissa: http://community.plm.automation.siemens.com/t5/Solid-Edge/ct-p/solid-edge

[27]. Codeproject, Solid Edge ST Addins. [WWW]. [Viitattu 20.7.2014]. Saatavissa: http://www.codeproject.com/Articles/33576/Solid-Edge-ST-Addins-Part-I

[28]. Electronic Data Services. 2004. Programmer's Guide Customizing Solid Edge. Version 15. EDS, 13736 Riverport Drive, Maryland Heights, MO 63043. [PDF]. [Viitattu 25.2.2014]. Saatavissa: http://rct2.narod.ru/solidedge/ProgGuide.pdf

[29]. A.C. Putman & K. Willmert. 2011. Automation of Solid Edge Using External Clients Written in C++. AMO - Advanced Modeling and Optimization, Volume 13, Number 3. Mechanical & Aeronautical Engineering Department, Clarkson University, 8 Clarkson Ave, Potsdam, NY, 13699

[30]. Siemens Global Technical Access Center. [WWW]. [Viitattu 20.8.2013]. Saatavissa: http://support.industrysoftware.automation.siemens.com/gtac.shtml

[31]. Microsoft Visual Basic 2008 Express Edition -ohjelmointiympäristö. [Viitattu 19.3.2013]. Saatavissa: http://www.visualstudio.com/

SIEMENS

LIITE 1: RIIKKU RAKENTEET CASE STUDY

Specialty construction

Riikku Rakenteet

Using Solid Edge enables specialty construction firm to improve productivity and quality, expand internationally

Product

Solid Edge

Business challenges

Implement software solution that facilitates quality and efficiency

New design and production support for international expansion

Challenging implementation schedules

Keys to success

Easy-to-use design software that increases productivity

Programmable APIs and the versatile tools of Solid Edge

Strong local technology support from IDEAL PLM

Results

Substantially improved efficiency across operations

Project plans that used to take 2 days now take 2 hours

Improved product quality, continuous product innovation

Significantly reduced material waste

Faster time-to-market, improved business opportunities

"The use of Solid Edge definitely boosts our business."

Joni Mäkiranta
Director
Riikku Rakenteet

Project plans that used to take 2 days now take 2 hours

Meeting customer needs

Riikku Rakenteet Ltd. is an aluminum and glass construction company that designs, manufactures, sells and installs a variety of construction products, including aluminum exterior glass walls, light ceilings and doors. Riikku Rakenteet's customers are large property contractors, construction companies, balcony builders and renovators. By using Solid Edge® software from product lifecycle management (PLM) specialist Siemens PLM Software, Riikku Rakenteet is implementing its vision for international expansion.

"At a very early stage, we recognized that we needed software that would allow us to serve our customers better with higher quality and more efficiency," says Joni Mäkiranta, director at Riikku Rakenteet. "The use of Solid Edge definitely boosts our business."

Riikku Rakenteet's primary market is Finland, but it also conducts business in

"At a very early stage, we recognized that we needed software that would allow us to serve our customers better with higher quality and more efficiency. The use of Solid Edge definitely boosts our business."

Joni Mäkiranta
Director
Riikku Rakenteet

Sweden and Russia. It aims to grow and become a significant manufacturer of facade and balcony products in Europe. In 2012, the company invested in new production equipment and methods to achieve its internationalization targets. Riikku Rakenteet realized that it needed tools that would enable it to design products more quickly so it could meet the needs of its most demanding customers.

Riikku Rakenteet chose Solid Edge to meet these challenges because it is a hybrid 2D/3D computer-aided design (CAD) system that uses synchronous technology for product development, design and manufacturing, making it an exceptional fit for the company's wide-ranging and diverse needs.

Increased productivity

"The friendly, browser-based user interface of Solid Edge enables rapid learning, thus increasing resource productivity and accelerating the design process," says Mäkiranta. "We can also get to market more quickly and with the highest-quality products."

The application programming interface interfaces (APIs) and versatile tools of Solid Edge were two other important reasons why Riikku Rakenteet selected the design software.

In using Solid Edge, Riikku Rakenteet found that the software was easy to learn and use, even for those who did not have strong technical skills. In fact, a user does not need to possess specific know-how to effectively use 3D and plan tailored solutions. For example, the sales force found that it could easily design customized products using Solid Edge.

"Customized products bring their own challenges to the design process at our company," says Mäkiranta. "Each window or balcony glass is unique when it comes to its dimensions, even if it does not seem so when viewing the exterior of an apartment building."

Solid Edge offers a cost-effective, industry-leading 3D modeling program for mechanical designers. It includes history-free, feature-based synchronous technology functionality, which makes design planning faster and more efficient, and allows models created with other system templates to be edited. Moreover, the use of synchronous technology was especially important is helping Riikku Rakenteet produce better balcony designs, and do so notably faster.

"It used to take us two days to do project calculations. With Solid Edge, it takes us just two hours."

Joni Mäkiranta
Director
Riikku Rakenteet

Solutions/Services

Solid Edge
www.siemens.com/solidedge

Customer's primary business

Riikku Rakenteet Ltd. is an aluminum and glass construction company. Its customers are large property contractors, construction companies, balcony builders and renovators. Established in 2005, Riikku Rakenteet's operations cover the lifecycle of metal and glass building facades and balconies from design to installation.
www.riikku.fi

Customer location

Alavus
Finland

"The friendly, browser-based user interface of Solid Edge enables rapid learning, thus increasing resource productivity and accelerating the design process. We can also get to market more quickly and with the highest-quality products."

Joni Mäkiranta
Director
Riikku Rakenteet

Reducing material waste

With Solid Edge, all important design information is addressed immediately in the targeted environment, thus eliminating the need to rely on someone's memory and save data multiple times. As a result, the risk of human error at Riikku Rakenteet has been notably reduced, and processes have become measurably more efficient.

"The time savings realized for project calculations are very significant," says Mäkiranta. "It used to take us two days to do project calculations. With Solid Edge, it takes us just two hours."

Now, with the help of the web-based calculation application, all dimensions of a balcony can be customized. "We also use Solid Edge to optimize waste reduction, so customers receive the most effective pricing," says Mäkiranta.

Traditionally, aluminum and glass construction companies have not used such advanced design automation. Riikku Rakenteet's use of Solid Edge is an outstanding example of how sophisticated software applications can be utilized to deliver substantial time, cost and quality gains in Finland.

Providing strong local support

"Internationalization has brought the need for time- and place-independent applications," says Mäkiranta. "We can now effectively present our solutions across multiple geographic markets, in large part due to the use of Solid Edge, which was delivered to us by IDEAL PLM."

Mäkiranta notes that IDEAL PLM, a Siemens PLM Software partner with more than 20 years of experience in product innovation and development, played an important role when Riikku Rakenteet was selecting an appropriate design tool. IDEAL PLM is not just a system supplier. It also helps customers use the system properly to gain the greatest benefits.

"We are very familiar with our customer's business, and we work together to consider new ways to operate and develop planning methods," says Miikka Lintusaari, Solid Edge business leader at IDEAL PLM. "Riikku Rakenteet has been open to thinking about new, innovative ways of operating, and that's been critical to their success."

Siemens PLM Software

Americas +1 314 264 8287
Europe +44 (0) 1276 413200
Asia-Pacific +852 2230 3308

www.siemens.com/plm

Z4 36365 8/13 A

LIITE 2: CATID:T SOLID EDGE MALLINNUSYMPÄRISTÖILLE

Category ID	GUID	Notes
CATID_SEApplication	{26618394-09D6-11d1-BA07-080036230602}	Application Environment
CATID_SEAssembly	{26618395-09D6-11d1-BA07-080036230602}	Traditional Assembly Document Env.
CATID_SEMotion	{67ED3F40-A351-11d3-A40B-0004AC969602}	
CATID_SEPart	{26618396-09D6-11d1-BA07-080036230602}	Traditional Part Document Env.
CATID_SEProfile	{26618397-09D6-11d1-BA07-080036230602}	
CAT-ID_SEFeatureRecognition	{E6F9C8DC-B256-11d3-A41E-0004AC969602}	
CATID_SESheetMetal	{26618398-09D6-11d1-BA07-080036230602}	Traditional SheetMetal Document E.
CATID_SEDraft	{08244193-B78D-11d2-9216-00C04F79BE98}	Draft Document Environment
CATID_SEWeldment	{7313526A-276F-11d4-B64E-00C04F79B2BF}	
CATID_SEXpresRoute	{1661432A-489C-4714-B1B2-61E85CFD0B71}	XpresRoute / "Tubing" Env.
CATID_SEExplode	{23BE4212-5810-478b-94FF-B4D682C1B538}	
CATID_SESimplify	{CE3DCEBF-E36E-4851-930A-ED892FE0772A}	
CATID_SEStudio	{D35550BF-0810-4f67-97D5-789EDBC23F4D}	
CATID_SELayout	{27B34941-FFCD-4768-9102-0B6698656CEA}	
CATID_SESketch	{0DDABC90-125E-4cfe-9CB7-DC97FB74CCF4}	Sketch / "LayoutInPart" Environment
CATID_SEProfileHole	{0D5CC5F7-5BA3-4d2f-B6A9-31D9B401FE30}	
CATID_SEProfilePattern	{7BD57D4B-BA47-4a79-A4E2-FFD43B97ADF}	

CATID_SEProfileRevolved	{FB73C683-1536-4073-B792-E28B8D31146E}	
CAT-ID_SEDrawingViewEdit	{8DBC3B5F-02D6-4241-BE96-B12EAF83FAE6}	
CATID_SERefAxis	{B21CCFF8-1FDD-4f44-9417-F1EAE06888FA}	
CAT-ID_SECuttingPlaneLine	{7C6F65F1-A02D-4c3c-8063-8F54B59B34E3}	
CAT-ID_SEBrokenOutSectionProfile	{534CAB66-8089-4e18-8FC4-6FA5A957E445}	
CATID_SEFrame	{D84119E8-F844-4823-B3A0-D4F31793028A}	
CATID_2dModel	{F6031120-7D99-48a7-95FC-EEE8038D7996}	
CATID_EditBlockView	{892A1CDA-12AE-4619-BB69-5156C929832}	
CAT-ID_SEComponentSketchInPart	{FAB8DC23-00F4-4872-8662-18DD013F2095}	
CAT-ID_SEComponentSketchInAsm	{86D925FB-66ED-40d2-AA3D-D04E74838141}	
CATID_SEHarness	{5337A0AB-23ED-4261-A238-00E2070406FC}	
CATID_SEAll	{C484ED57-DBB6-4a83-BEDB-C08600AF07BF}	All Environments
CAT-ID_SEAllDocumentEnvrionments	{BAD41B8D-18FF-42c9-9611-8A00E6921AE8}	All Document Environments
CATID_SEDMPart	{D9B0BB85-3A6C-4086-A0BB-8A1AAD57A58}	Synchronous Part Document Env.
CATID_SEDMSheetMetal	{9CBF2809-FF80-4dbc-98F2-B82DABF3530F}	Synchronous SheetMetal Document Environment
CATID_SEDMAssembly	{2C3C2A72-3A4A-471d-98B5-E3A8CFA4A2BF}	Synchronous Assembly Document Environment

LIITE 3: SOLID EDGE TYYPPIKIRJASTOT

Tiedostot löytyvät Solid Edge asennuskansiossa olevasta Program -kansiosta.

Tiedosto	Nimi	Kuvaus	LIBID
assembly.tlb	SolidEdgeAssembly	Solid Edge Assembly Type Library	A2259BE4-7E1B-4893-AFF1-96C721E58FDD
assemblysync.tlb	SolidEdgeAssemblySync	Solid Edge Assembly Sync Type Library	3E2B3BD4-F0B9-11D1-BDFD-080036B4D502
constant.tlb	SolidEdgeConstants	Solid Edge Constants Type Library	C467A6F5-27ED-11D2-BE30-080036B4D502
draft.tlb	SolidEdgeDraft	Solid Edge Draft Type Library	3E2B3BDC-F0B9-11D1-BDFD-080036B4D502
framewrk.tlb	SolidEdgeFramework	Solid Edge Framework Type Library	8A7EFA3A-F000-11D1-BDFC-080036B4D502
fwksupp.tlb	SolidEdgeFrameworkSupport	Solid Edge FrameworkSupport Type Library	943AC5C6-F4DB-11D1-BE00-080036B4D502
geometry.tlb	SolidEdgeGeometry	Solid Edge Geometry Type Library	3E2B3BE1-F0B9-11D1-BDFD-080036B4D502
InstallData.tlb	SEInstallDataLib	Solid Edge Install Data Library	42E04299-18A0-11D5-BBB2-00C04F79BEA5
Part.tlb	SolidEdgePart	Solid Edge Part Type Library	8A7EFA42-F000-11D1-BDFC-080036B4D502
PartSync.tlb	SolidEdgePartSync	Solid Edge Part Sync Type Library	EAB2DA59-5457-41D9-AEEA-9EC9D424A208

LIITE 4: TUOTEKONFIGURAATTORIN KOODI

```
Imports System.Runtime.InteropServices
Imports System.Diagnostics
Imports SolidEdgeFramework
'Summary:
'Tämä ohjelma on luotu osana Tampereen teknillisen yliopiston
'diplomityötä: Suunnitteluautomaation implementointi CAD-
'järjestelmään. Tämä ohjelma lähettää komentoja samalla
'tietokoneella olevalle Solid Edge ohjelmalle ja luo käyttäjän
'valintojen mukaisen konfiguraation sähkömoottoreista.
'Tekijä: Mikko Junttila

Public Class MainForm
    'Nimetään objektit, josta alkuun Solid Edgen vaatimat
viittaukset
    Dim objApp As SolidEdgeFramework.Application
    Dim objDocuments As SolidEdgeFramework.Documents = Nothing
    Dim objAssembly As SolidEdgeAssembly.AssemblyDocument = Nothing
    Dim objAssembly1 As SolidEdgeAssembly.AssemblyDocument = Noth-
ing
    Dim objOccurrences As SolidEdgeAssembly.Occurrences = Nothing
    Dim objOccurrence As SolidEdgeAssembly.Occurrence = Nothing
    Dim objType As Type

    'Lisätään objekteihin myös ohjelmassa käytettyjä muuttujia
    Dim prosessi() As Process
    Dim luokka As String = ""
    Dim materiaali As String = ""
    Dim rivat As String = ""
    Dim jalka As String = ""
    Dim laippa As String = ""
    Dim kokonaisuus As String
    Dim firma As String
    Dim asiakas As String
    Public i As Integer
    Public j As Integer

    Private Sub MainForm_FormClosing(ByVal sender As Object, ByVal
e As System.Windows.Forms.FormClosingEventArgs) Handles
Me.FormClosing
        If Not (objAssembly1 Is Nothing) Then
            Marshal.ReleaseComObject(objAssembly1)
            objAssembly1 = Nothing
        End If
    End Sub


    Public Sub MainForm_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
```

```
        'Kun ohjelma käynnistetään, tämä tarkastaa onko Solid Edge
jo käynnissä
        'ja ottaa siihen yhteyden, jos ei, niin uusi Solid Edge
käynnistetään.
        Try
            'Koitetaan ottaa yhteys auki olevaan Solid Edgeen
            prosessi = Process.GetProcessesByName("edge")
            If prosessi.Count > 0 Then
                objApp = Mar-
shal.GetActiveObject("SolidEdge.Application")
            Else
                'Haetaan objType ProgID metodilla
                objType =
Type.GetTypeFromProgID("SolidEdge.Application")
                'Käynnistetään uusi Solid Edge instanssi
                objApp = Activator.CreateInstance(objType)
            End If
            'Asetetaan SE käyttäjälle näkyväksi
            objApp.Visible = True
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try
    End Sub

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e
As  _
                            System.EventArgs) Handles btnOK.Click
        'Komennot lähetään Solid Edgelle kun Tuotekonfiguraattorin
        '"OK" -näppäintä painetaan.

        'Alkuun tarkastetaan sisältävätkö syötteet puutteita tai
virheitä.
        'Jos virheitä löytyy, kerrotaan virheestä käyttäjälle ja
palataan
        'tuotekonfiguraattoriin.

        'Luokka:
        If Not (cboLuokka.SelectedIndex = 0 Or cboLuok-
ka.SelectedIndex = 1) Then
            MessageBox.Show("Et ole valinnut sähkömoottorin
luokkaa!")
            Exit Sub
        End If

        'Materiaali:
        If Not (cboMaterial.SelectedIndex = 0 Or cboMateri-
al.SelectedIndex = 1) Then
            MessageBox.Show("Et ole valinnut sähkömoottorin
materiaalia!")
            Exit Sub
        End If

        'Jäähdytys
        If cboMaterial.SelectedIndex = 0 Then
            If Not (cboRivat.SelectedIndex = 0 Or
cboRivat.SelectedIndex = 1) Then
                MessageBox.Show("Et ole valinnut jäähdytystä!")
                Exit Sub
```

```
            End If
        End If
        If (cboMaterial.SelectedIndex = 1 And
cboRivat.SelectedIndex = 0) Then
            MessageBox.Show("Alumiinimoottori saatavana vain ilman
jäähdytystä.")
            cboRivat.Visible = False
            lblJaa.Visible = False
            picRivat.Visible = False
            cboRivat.SelectedIndex = 1
            Exit Sub
        End If

        'Moottorien lukumäärä
        If txtLKM.Text = "" Then
            MessageBox.Show("Syötä sähkömoottorien lukumäärä.")
            Exit Sub
        End If
        If Not (IsNumeric(txtLKM.Text)) Then
            MessageBox.Show(txtLKM.Text & " ei ole numero")
            Exit Sub
        End If
        If CInt(txtLKM.Text) < 1 Or CInt(txtLKM.Text) > 7 Then
            MessageBox.Show("Moottorien lukumäärän tulee olla
enemmän kuin" & _
                            " yksi (1) ja vähemmän kuin neljä (7)")
            Exit Sub
        End If



        'Kun syötteet on hyväksytty lähetetään komentoja Solid
Edgelle.
        'Luodaan halutun moottorin alikokoonpano:
        Dim moottori As String
        moottori = ("M" & luokka & materiaali & rivat & laippa &
jalka)
        'Annetaan ohjelman levähtää hetki
        objApp.DoIdle()
        Try
            'Haetaan Documents-kokoelman referenssi
            objDocuments = objApp.Documents
            'Luodaan kokoonpanodokumentti
            objAssembly = objDocu-
ments.Add("SolidEdge.AssemblyDocument")
            'Haetaan Occurences-kokoelman referenssi
            objOccurrences = objAssembly.Occurrences
            'Lisätään kokoonpanoon halutun mukaiset osat:
            'Kuori:
            objOccurrence =
objOccurrences.AddByFilename("D:\SE\Osat\kuori" & jalka _
                                                        &
materiaali & luokka & rivat & ".par")
            'Takaosa
            objOccurrence =
objOccurrences.AddByFilename("D:\SE\Osat\taka" & materiaali _
                                                    & luokka &
".par")
```

```
            'Etuosa
            objOccurrence =
objOccurrences.AddByFilename("D:\SE\Osat\etu" & laippa & materiaali
_
                                                                & luokka &
".par")
            'Akseli
            objOccurrence = objOccurrenc-
es.AddByFilename("D:\SE\Osat\akseli.par")

            'Vaihdetaan kuvannoksi: ISO view.
            ob-
jApp.StartCommand(CType(SolidEdgeConstants.PartCommandConstants.Par
tViewISOView,  _

SolidEdgeFramework.SolidEdgeCommandConstants))
            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()
            'Sammutetaan varoitukset, sillä jos kokoonpanoa on
käytetty aiemmin
            'esittää SE varoituksen päälle tallentamisesta
            objApp.DisplayAlerts = False
            'Tallennetaan moottorikokoonpano
            objAssembly.SaveAs("D:\SE\Moottorikokoonpanot\" &
moottori & ".asm")
            'Palautetaan varoitukset päälle
            objApp.DisplayAlerts = True
            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()
            objAssembly.Close()



            'Luodaan pääkokoonpano, johon lisätään alikokoonpanona
haluttu
            'määrä moottoreita.
            'Avataan uusi assembly -dokumentti
            objAssembly1 = objDocu-
ments.Add("SolidEdge.AssemblyDocument")
            'Haetaan Occurences-kokoelman referenssi
            objOccurrences = objAssembly1.Occurrences
            For Me.i = 1 To CInt(txtLKM.Text)
                'Lisätään luotu moottorin alikokoonpano nykyiseen
kokoonpanoon
                objOccurrence = objOccurrences.AddByFilename _
                ("D:\SE\Moottorikokoonpanot\" & moottori & ".asm")
                If Me.i > 1 Then
                    'Siirretään yksittäistä moottoria 40
senttimertiä x-akselin suunnassa
                    objOccurrence.Move((i - 1) * 0.4, 0, 0)
                End If
            Next Me.i
            'Vaihdetaan kuvannoksi: ISO view.
            ob-
jApp.StartCommand(CType(SolidEdgeConstants.PartCommandConstants.Par
tViewISOView,  _

SolidEdgeFramework.SolidEdgeCommandConstants))
```

```
            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()


            'Tarkastetaan onko käyttäjä kirjannut yrityksen ja oman
nimensä
            'jos ei, niin käytetään oletus nimiä.
            firma = txtYritys.Text
            If firma = "" Then
                firma = "Yritys"
            End If
            asiakas = txtNimi.Text
            If asiakas = "" Then
                asiakas = "Tilaaja"
            End If

            'Sijoitetaan kokonaisuuden nimi muuttujaan kokonaisuus
            kokonaisuus = (firma & " " & asiakas & " " &
txtLKM.Text & "x" & moottori)

            'Sammutetaan varoitukset, jotta jos kokoonpanoa on
käytetty aiemmin
            'esittää SE varoituksen päälle tallentamisesta
            objApp.DisplayAlerts = False
            'Tallennetaan kokoonpano haluttuun sijaintiin nimen
muodolla:
            'esimerkkiyritys oy\matti meikäläinen
3xm_180_fe_j_l_j.asm
            objAssembly1.SaveAs("D:\SE\Kokoonpanot\" & kokonaisuus
& ".asm")
            'Palautetaan varoitukset päälle
            objApp.DisplayAlerts = True


        Catch ex As Exception
            Console.WriteLine(ex.Message)
        Finally
            'Tyhjennetään purskurimuisti
            If Not (objOccurrence Is Nothing) Then
                Marshal.ReleaseComObject(objOccurrence)
                objOccurrence = Nothing
            End If
            If Not (objOccurrences Is Nothing) Then
                Marshal.ReleaseComObject(objOccurrences)
                objOccurrences = Nothing
            End If
            If Not (objAssembly Is Nothing) Then
                Marshal.ReleaseComObject(objAssembly)
                objAssembly = Nothing
            End If
            If Not (objDocuments Is Nothing) Then
                Marshal.ReleaseComObject(objDocuments)
                objDocuments = Nothing
            End If
            If Not (objApp Is Nothing) Then
                Marshal.ReleaseComObject(objApp)
                objApp = Nothing
            End If
```

```
            If Not (objDocuments Is Nothing) Then
                Marshal.ReleaseComObject(objDocuments)
                objDocuments = Nothing
            End If
            prosessi = Nothing
            luokka = Nothing
            materiaali = Nothing
            rivat = Nothing
            jalka = Nothing
            laippa = Nothing
            kokonaisuus = Nothing
            firma = Nothing
            asiakas = Nothing
            i = Nothing
            j = Nothing
        End Try

        'Tarkastetaan onko chcSulje valittu?
        If chcSulje.Checked Then
            'Jos on, niin suljetaan tuotekonfiguraattori OK:ta
painaessa
            Me.Close()
        ElseIf Not (chcSulje.Checked) Then
            'Jos ei, niin jätetään konfiguraattori käyntiin
            'Asetetaan aiempien kokoonpanojen sulkunappi näkyviin
            btnSulje.Visible = True
            j = j + 2
            'Otetaan uudelleen kontakti SE:hen.
            objApp = Mar-
shal.GetActiveObject("SolidEdge.Application")
        End If

    End Sub
    'Tässä osiossa käydään läpi käyttäjän syötteitä

    Private Sub cboMaterial_TextChanged(ByVal sender As Object,
ByVal e As System.EventArgs) _
    Handles cboMaterial.TextChanged
        'Jos valitaan kokoelman ensimmäinen vaihtoehto (Valurauta),
asetetaan
        'toinen combobox näkyviin ja kirjataan muuttujaan
materiaali "fe"
        If cboMaterial.SelectedIndex = 0 Then
            cboRivat.Visible = True
            lblJaa.Visible = True
            picRivat.Visible = True
            materiaali = "_fe"
            'Jos valitaan toinen vaihtoehto (Alumiini)
            'kirjataan muuttujaan materiaali "al"
        ElseIf cboMaterial.SelectedIndex = 1 Then
            materiaali = "_al"
        Else
            'Muussa tapauksessa rivat comboboxia ei esitetä
            cboRivat.Visible = False
            lblJaa.Visible = False
            picRivat.Visible = False
        End If
    End Sub
```

```
    Private Sub cboLuokka_TextChanged(ByVal sender As Object, ByVal
e As System.EventArgs) _
    Handles cboLuokka.TextChanged
        'Kirjataan valittu luokka ko. muuttujaan
        If cboLuokka.SelectedIndex = 0 Then
            luokka = "_180"
        ElseIf cboLuokka.SelectedIndex = 1 Then
            luokka = "_200"
        End If
    End Sub

    Private Sub cboRivat_TextChanged(ByVal sender As Object, ByVal
e As System.EventArgs) _
    Handles cboRivat.TextChanged
        'Kirjataan ripojen valinta ko. muuttujaan
        If cboRivat.SelectedIndex = 0 Then
            rivat = "_j"
        Else
            rivat = ""
        End If
    End Sub

    Private Sub chcJalka_Click(ByVal sender As Object, ByVal e As
System.EventArgs) _
    Handles chcJalka.Click
        'Kirjataan jalan valinta ko. muuttujaan
        If chcJalka.Checked Then
            jalka = "_j"
        ElseIf Not (chcJalka.Checked) Then
            jalka = ""
        End If
    End Sub

    Private Sub chcLaippa_Click(ByVal sender As Object, ByVal e As
System.EventArgs) _
    Handles chcLaippa.Click
        'Kirjataan laipan valinta ko. muuttujaan
        If chcLaippa.Checked Then
            laippa = "_1"
        ElseIf Not (chcLaippa.Checked) Then
            laippa = ""
        End If
    End Sub

    Private Sub btnHelp_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnHelp.Click
        MessageBox.Show("Tämä ohjelma on luotu osana Tampereen
teknillisen yliopiston" & _
                        "diplomityötä: Suunnitteluautomaation
implementointi CAD-" & _
                        "järjestelmään. Tämä ohjelma lähettää
komentoja samalla" & _
                        "tietokoneella olevalle Solid Edge
ohjelmalle ja luo käyttäjän" & _
                        "valintojen mukaisen konfiguraation
sähkömoottoreista." & _
                        "Tekijä: Mikko Junttila")
```

```
    End Sub

    Private Sub btnSulje_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnSulje.Click
        Marshal.GetActiveObject("SolidEdge.Application")
        objAssembly1.Close()
        btnSulje.Visible = False
    End Sub
End Class
```

LIITE 5: ADD-IN OHUTLEVYTYÖKALUN KOODI

```
Imports System.Runtime.InteropServices
Imports Microsoft.Win32
Imports System.Reflection
Imports System.Collections
Imports System.Collections.Generic
Imports System.Text
Imports SolidEdgeFramework
Imports SolidEdgeFrameworkSupport
Imports SolidEdgeConstants
Imports SolidEdgeGeometry
Imports SolidEdgePart
Imports SolidEdgeAssembly

<GuidAttribute("32CA0E63-D23A-4304-982B-A02EA9E02E6D"), _
ProgIdAttribute("Ohutlevy AddIn.jununee"), _
ComVisible(True)> _
Public Class Ohutlevy
    Implements SolidEdgeFramework.ISolidEdgeAddIn, SolidEdgeFrame-
work.ISEAddInEvents, SolidEdgeFramework.ISEApplicationEvents
    Private objAddin As SolidEdgeFramework.AddIn
    Private objApplication As SolidEdgeFramework.Application
    Private WithEvents objAppEvents As SolidEdgeFrame-
work.DISEApplicationEvents_Event
    Friend objenvironmentCommandMap As Sys-
tem.Collections.Generic.Dictionary _
    (Of String, System.Collections.Generic.Dictionary(Of Integer,
Integer))
    Dim WithEvents objButtonEvents As SolidEdgeFrame-
work.CommandBarButtonEvents
    'Private WithEvents objEvents As SolidEdgeFrame-
work.ISEAddInEvents
    Private WithEvents AddInCmdEvents As SolidEdgeFrame-
work.ISEAddInEvents_Event

    Public Sub OnConnection(ByVal Application As Object, ByVal Con-
nectMode As SolidEdgeFramework.SeConnectMode, _
                            ByVal AddInInstance As SolidEdgeFrame-
work.AddIn) _
                            Implements SolidEdgeFrame-
work.ISolidEdgeAddIn.OnConnection
        objAddin = AddInInstance
        objApplication = CType(Application, SolidEdgeFrame-
work.Application)
        objenvironmentCommandMap = New Sys-
tem.Collections.Generic.Dictionary _
        (Of String, System.Collections.Generic.Dictionary(Of Inte-
ger, Integer))
        objAppEvents = CType(objApplication.ApplicationEvents,
SolidEdgeFramework.DISEApplicationEvents_Event)
```

```
        Try
            AddInCmdEvents = AddInInstance.AddInEvents
        Catch ex As Exception
        End Try
        AddInInstance.GuiVersion = 1
        AddInInstance.Description = "Ohutlevy"
    End Sub
    Public Sub OnConnectToEnvironment(ByVal EnvCatID As String,
ByVal pEnvironmentDispatch As Object, _
                                      ByVal bFirstTime As Boolean)
Implements SolidEdgeFramework. _
                                      ISolidEdgeAdd-
In.OnConnectToEnvironment
        Dim commandName(0) As String
        Dim commandID(0) As Integer
        Dim dictionary As New Sys-
tem.Collections.Generic.Dictionary(Of Integer, Integer)
        Try
            commandName(0) = (String.Format("SheetLogo64{0}
Tuotekonfiguraattori / Ohutlevy", _
                                            ControlChars.Lf, Con-
trolChars.Lf, ControlChars.Lf))
            commandID(0) = 1000
            Dim ToolBarLarge, ToolBarSmall As Integer
            ToolBarLarge = 2
            ToolBarSmall = 1
            objAddin.AddCommand(EnvCatID, commandName(0), comman-
dID(0))

            ob-
jAddin.SetAddInInfo(Marshal.GetHINSTANCE(Me.GetType().Module).ToInt
32(), _
                                    EnvCatID, "Ohutlevy",
ToolBarSmall, ToolBarLarge, ToolBarSmall, _
                                    ToolBarLarge, 2, commandName,
commandID)

            dictionary.Add(commandID(0), commandID(0))
            objenvironmentCommandMap.Add(EnvCatID, dictionary)

            Dim ctr As SolidEdgeFramework.ISECommandBarButton

            ctr = objAddin.AddCommandBarButton(EnvCatID,
"Ohutlevy", 1000)

            objButtonEvents = ctr.CommandBarButtonEvents

            ctr.Style = SolidEdgeFrame-
work.SeButtonStyle.seButtonIconAndCaption

            objAddin.Visible = True

            ctr.LoadFace("C:\Users\jununee\Documents\TTY\Dippa
2\Kuvatiedostot\SheetLogo64.bmp")

        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
```

```
    End Sub

    Public Sub OnDisconnection(ByVal DisconnectMode As Solid-
EdgeFramework.SeDisconnectMode) _
    Implements SolidEdgeFramework.ISolidEdgeAddIn.OnDisconnection
        If Not (objAddin Is Nothing) Then
            objAddin = Nothing
        End If
        If Not (objApplication Is Nothing) Then
            objApplication = Nothing
        End If
    End Sub
    <ComRegisterFunctionAttribute()> _
    Public Shared Sub RegisterFunction(ByVal t As Type)
        Dim baseKey As RegistryKey = Nothing
        Dim summaryKey As RegistryKey = Nothing
        Dim title As AssemblyTitleAttribute
        Dim description As AssemblyDescriptionAttribute
        Try
            baseKey = Registry.ClassesRoot.CreateSubKey("CLSID\{" +
t.GUID.ToString + "}")
            baseKey.SetValue("AutoConnect", 1)
            If
t.Assembly.IsDefined(GetType(AssemblyTitleAttribute), True) Then
                title = AssemblyTitleAttrib-
ute.GetCustomAttribute(t.Assembly, GetType(AssemblyTitleAttribute))
                baseKey.SetValue("409", title.Title)
            End If
            If
t.Assembly.IsDefined(GetType(AssemblyDescriptionAttribute), True)
Then
                description = AssemblyDefaultAliasAttrib-
ute.GetCustomAttribute(t.Assembly, Get-
Type(AssemblyDescriptionAttribute))
                summaryKey = baseKey.CreateSubKey("Summary")
                summaryKey.SetValue("409", description.Description)
                summaryKey.Close()
            End If
            'Addin
            baseKey.CreateSubKey("Implemented Categories\{26B1D2D1-
2B03-11d2-B589-080036E8B802}")
            'Assembly
            baseKey.CreateSubKey("Environment Categories\{26618395-
09D6-11d1-BA07-080036230602}")

        Catch ex As Exception

        Finally
            If Not (summaryKey Is Nothing) Then
                summaryKey.Close()
            End If
            If Not (baseKey Is Nothing) Then
                baseKey.Close()
            End If
        End Try
    End Sub
    <ComUnregisterFunctionAttribute()> _
    Public Shared Sub UnregisterFunction(ByVal t As Type)
```

```
        Try
            Registry.ClassesRoot.DeleteSubKeyTree("CLSID\{" +
t.GUID.ToString + "}")
        Catch ex As Exception
        End Try
    End Sub


    'Addin Events
    Public Sub OnCommand(ByVal CommandID As Integer) Implements
SolidEdgeFramework.ISEAddInEvents.OnCommand
        Ohutlevy()
    End Sub

    Public Sub OnCommandHelp(ByVal hFrameWnd As Integer, ByVal
HelpCommandID As Integer, ByVal CommandID As Integer) _
    Implements SolidEdgeFramework.ISEAddInEvents.OnCommandHelp

    End Sub

    Public Sub OnCommandUpdateUI(ByVal CommandID As Integer, ByRef
CommandFlags As Integer, ByRef MenuItemText As String, _
                                    ByRef BitmapID As Integer) Imple-
ments SolidEdgeFramework.ISEAddInEvents.OnCommandUpdateUI

    End Sub

    Public Sub AfterActiveDocumentChange(ByVal theDocument As Ob-
ject) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterActiveDocumentChange

    End Sub

    Public Sub AfterCommandRun(ByVal theCommandID As Integer) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterCommandRun

    End Sub

    Public Sub AfterDocumentOpen(ByVal theDocument As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterDocumentOpen

    End Sub

    Public Sub AfterDocumentPrint(ByVal theDocument As Object,
ByVal hDC As Integer, _
                                    ByRef ModelToDC As Double, ByRef
Rect As Integer) _
                                    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterDocumentPrint

    End Sub

    Public Sub AfterDocumentSave(ByVal theDocument As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterDocumentSave
```

```
    End Sub

    Public Sub AfterEnvironmentActivate(ByVal theEnvironment As Ob-
ject) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterEnvironmentActivate

    End Sub

    Public Sub AfterNewDocumentOpen(ByVal theDocument As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterNewDocumentOpen

    End Sub

    Public Sub AfterNewWindow(ByVal theWindow As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterNewWindow

    End Sub

    Public Sub AfterWindowActivate(ByVal theWindow As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.AfterWindowActivate

    End Sub

    Public Sub BeforeCommandRun(ByVal theCommandID As Integer) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeCommandRun

    End Sub

    Public Sub BeforeDocumentClose(ByVal theDocument As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeDocumentClose

    End Sub

    Public Sub BeforeDocumentPrint(ByVal theDocument As Object,
ByVal hDC As Integer, _
                                   ByRef ModelToDC As Double, ByRef
Rect As Integer) _
                                   Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeDocumentPrint

    End Sub

    Public Sub BeforeDocumentSave(ByVal theDocument As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeDocumentSave

    End Sub

    Public Sub BeforeEnvironmentDeactivate(ByVal theEnvironment As
Object) _
```

```
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeEnvironmentDeactivate

    End Sub

    Public Sub BeforeQuit() Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeQuit

    End Sub

    Public Sub BeforeWindowDeactivate(ByVal theWindow As Object) _
    Implements SolidEdgeFrame-
work.ISEApplicationEvents.BeforeWindowDeactivate

    End Sub

    Private Sub objButtonEvents_Click() Handles objButtonE-
vents.Click
        Ohutlevy()
    End Sub

    Private Sub Ohutlevy()
        'Tämä rutiini tuottaa halutut ohulevy-kappaleet ja lisää ne
kokoonpanoon.
        Dim objApp As SolidEdgeFramework.Application = Nothing
        Dim objDocuments As SolidEdgeFramework.Documents = Nothing
        Dim objAssembly As SolidEdgeAssembly.AssemblyDocument =
Nothing
        Dim objSheet As SolidEdgePart.SheetMetalDocument = Nothing
        Dim objProfileSets As SolidEdgePart.ProfileSets = Nothing
        Dim objProfileSet As SolidEdgePart.ProfileSet = Nothing
        Dim objProfiles As SolidEdgePart.Profiles = Nothing
        Dim objProfile As SolidEdgePart.Profile = Nothing
        Dim objRefplanes As SolidEdgePart.RefPlanes = Nothing
        Dim objPPlane As SolidEdgePart.RefPlane = Nothing
        Dim objRelations2d As SolidEdgeFrameworkSupport.Relations2d
= Nothing
        Dim objRelation2d As SolidEdgeFrameworkSupport.Relation2d =
Nothing
        Dim objLines2d As SolidEdgeFrameworkSupport.Lines2d = Noth-
ing
        Dim objLine2d As SolidEdgeFrameworkSupport.Line2d = Nothing
        Dim objCircles2d As SolidEdgeFrameworkSupport.Circles2d =
Nothing
        Dim objCircle As SolidEdgeFrameworkSupport.Circle2d = Noth-
ing
        Dim objModels As SolidEdgePart.Models = Nothing
        Dim objModel As SolidEdgePart.Model = Nothing
        Dim objCuts As SolidEdgePart.ExtrudedCutouts = Nothing
        Dim objCut As SolidEdgePart.ExtrudedCutout = Nothing
        Dim objOccurrences As SolidEdgeAssembly.Occurrences = Noth-
ing
        Dim objOccurrence As SolidEdgeAssembly.Occurrence = Nothing
        Dim aProfiles As Array
        Dim objType As Type

        Dim prosessi() As Process
        Dim tiedosto As String
```

```
        Dim maara As String
        Dim lkm As Integer
        Dim paikka As Integer
        Dim materiaali As String
        Dim luokka As Integer
        Dim laippa As Boolean
        Dim jalka As Boolean
        Dim leveys As Double
        Dim i As Integer

        Try
            Try
                'Koitetaan ottaa yhteys auki olevaan Solid Edgeen
                prosessi = Process.GetProcessesByName("edge")
                If prosessi.Count > 0 Then
                    objApp = Mar-
shal.GetActiveObject("SolidEdge.Application")
                Else
                    Console.WriteLine("Solid Edge ei ole päällä")
                End If
                'Asetetaan SE käyttäjälle näkyväksi
                objApp.Visible = True
                'Otetaan referenssi dokumentti kokoelmaan
                objDocuments = objApp.Documents
                'Tsekataan, että löytyy aktiivinen kokoonpano
                If objDocuments.Count > 0 Then
                    'Yritetään ottaa yhteyttä aktiiviseen
kokoonpanoon
                    objAssembly = CType(objApp.ActiveDocument,
SolidEdgeFramework.SolidEdgeDocument)
                End If
                'Jos aktiivista kokoonpanoa ei ole, niin herjataan
                If objAssembly Is Nothing Then
                    Throw New System.Exception("No active docu-
ment.")
                End If
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try


            'Tarkastetaan tiedoston nimi
            tiedosto = objAssembly.Name
            Console.WriteLine(tiedosto)
            'Tarkastetaan montako osaa kokoonpanossa on
            paikka = InStr(tiedosto, "x")
            maara = Microsoft.VisualBasic.Mid(tiedosto, paikka - 1,
1)
            Console.WriteLine(maara & " konetta")
            lkm = CDbl(maara)
            leveys = (0.4 + (0.8 * (lkm - 1))) / 2
            paikka = InStr(tiedosto, "fe")
            If paikka = Nothing Then
                materiaali = "Alumiini"
            Else
                materiaali = "Valurauta"
            End If
            Console.WriteLine(materiaali)
```

```
            'moottori = ("M" & luokka & materiaali & rivat & laippa
& jalka)
            'kokonaisuus = (firma & " " & asiakas & " " &
txtLKM.Text & "x" & moottori)
            paikka = InStr(tiedosto, "180")
            If paikka = Nothing Then
                luokka = 200
            Else
                luokka = 180
            End If
            Console.WriteLine("Moottoriluokka: " & luokka)
            paikka = InStr(tiedosto, "_l_")
            If paikka = Nothing Then
                laippa = False
            Else
                laippa = True
            End If
            Console.WriteLine("Laippakiinnitys: " & laippa)
            If Microsoft.VisualBasic.Right(tiedosto, 5) = "j.asm"
Then
                jalka = True
            Else
                jalka = False
            End If
            Console.WriteLine("Jalkakiinnitys: " & jalka)


            'Tarkastetaan onko kokoopanossa jalka tai
laippakiinnitystä
            If laippa = False And jalka = False Then
                Console.WriteLine("Kokoonpanossa ei laippa- tai
jalkakiinnitystä")
                Exit Sub
            End If

            If laippa = True Then
                'Referenssi dokumenttikokoelmaan
                objDocuments = objApp.Documents
                'Luodaan uusi Sheet Metal -osa
                objSheet = objDocu-
ments.Add("SolidEdge.SheetMetalDocument")
                'Referenssi ProfileSets kokoelmaan
                objProfileSets = objSheet.ProfileSets
                'Lisätään Profiilisetti
                objProfileSet = objProfileSets.Add()
                'Haetaan refenssi Profiili kokoelmaan
                objProfiles = objProfileSet.Profiles
                'Referenssi Refplanes kokoelmaan
                objRefplanes = objSheet.RefPlanes
                'Luodaan uusi taso
                objPPlane = ob-
jRefplanes.AddParallelByDistance(ParentPlane:=objRefplanes.Item(3),
_
                                                                Dis-
tance:=(((luokka / 2) + 53) / 1000), _
                                                                Nor-
malSide:=SolidEdgePart. _
```

```
                                                                    Ref-
erenceElementConstants.igTangentToFace)
                'Lisätään profiili
                objProfile = objProfiles.Add(objPPlane)
                'Referenssi Lines2d kokoelmaan
                objLines2d = objProfile.Lines2d
                'Lisätään halutut viivat
                objLine2d = objLines2d.AddBy2Points(-0.2, -0.05,
leveys, -0.05)
                objLine2d = objLines2d.AddBy2Points(leveys, -0.05,
leveys, 0.05)
                objLine2d = objLines2d.AddBy2Points(leveys, 0.05, -
0.2, 0.05)
                objLine2d = objLines2d.AddBy2Points(-0.2, 0.05, -
0.2, -0.05)
                'Sidotaan viivat relaatioilla yhteen
                objRelations2d = objProfile.Relations2d
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(1), 1, objLines2d.Item(2), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(2), 1, objLines2d.Item(3), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(3), 1, objLines2d.Item(4), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(4), 1, objLines2d.Item(1), 0)

                'Suljetaan profiili
                objPro-
file.End(SolidEdgePart.ProfileValidationType.igProfileClosed)
                'Piilotetaan profiili näkyvistä
                'objProfile.Visible = False

                'Vaihdetaan kuvannoksi: ISO view.

objApp.StartCommand(CType(SolidEdgeConstants.PartCommandConstants.P
artViewISOView,  _
                                        SolidEdgeFrame-
work.SolidEdgeCommandConstants))

                'Referenssi models kokoelmaan
                objModels = objSheet.Models
                'Luodaan profiilin pohjalta tab-piirre
                objModel = objModels.AddBaseTab(objProfile, Solid-
EdgePart.FeaturePropertyConstants.igRight)
                'Piilotetaan profiili
                objProfile.Visible = False
                'Annetaan ohjelman levähtää hetki
                objApp.DoIdle()

                'Leikataan levyyn reikä

                For i = 0 To lkm - 1
                    'Referenssi ProfileSets kokoelmaan
                    objProfileSets = objSheet.ProfileSets
                    'Lisätään Profiilisetti
                    objProfileSet = objProfileSets.Add()
                    'Haetaan refenssi Profiili kokoelmaan
                    objProfiles = objProfileSet.Profiles
```

```
                    'Referenssi Refplanes kokoelmaan
                    objRefplanes = objSheet.RefPlanes
                    'Luodaan uusi taso (voidaan käyttää origossa
olevaa tasoa
                    'sillä leikataan through all
                    objPPlane = objRefplanes.Item(3)
                    'Lisätään profiili
                    objProfile = objProfiles.Add(objPPlane)
                    'Piirretään ympyrä
                    objCircles2d = objProfile.Circles2d
                    objCircle = objCircles2d.AddByCenterRadius((i *
0.4), 0, 0.03)
                    'Referenssi models kokoelmaan
                    objModels = objSheet.Models
                    'Piilotetaan profiili
                    objProfile.Visible = False
                    'Referenssi model kokoelmaan
                    objModels = objSheet.Models
                    objModel = objModels.Item(1)
                    'Leikkaus
                    objCuts = objModel.ExtrudedCutouts
                    objCut = objCuts.AddThroughAll(objProfile,
SolidEdgePart.FeaturePropertyConstants.igLeft, _

SolidEdgePart.FeaturePropertyConstants.igRight)
                Next i

                'Sammutetaan varoitukset, jotta jos kokoonpanoa on
käytetty aiemmin
                'esittää SE varoituksen päälle tallentamisesta
                objApp.DisplayAlerts = False
                'Tallennetaan 2D-kuva
                objSheet.SaveAs("D:\SE\Kokoonpanot\" & tiedosto & "
laippa.psm")
                'Palautetaan varoitukset päälle
                objApp.DisplayAlerts = True

                objSheet.Close()
            End If

            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()

            If jalka = True Then
                'Referenssi dokumenttikokoelmaan
                objDocuments = objApp.Documents
                'Luodaan uusi Sheet Metal -osa
                objSheet = objDocu-
ments.Add("SolidEdge.SheetMetalDocument")
                'Referenssi ProfileSets kokoelmaan
                objProfileSets = objSheet.ProfileSets
                'Lisätään Profiilisetti
                objProfileSet = objProfileSets.Add()
                'Haetaan refenssi Profiili kokoelmaan
                objProfiles = objProfileSet.Profiles
                'Referenssi Refplanes kokoelmaan
                objRefplanes = objSheet.RefPlanes
                'Luodaan uusi taso
```

```
                'Lisätään profiili
                objProfile = objProfiles.Add(objRefplanes.Item(1))
                'Referenssi Lines2d kokoelmaan
                objLines2d = objProfile.Lines2d
                'Lisätään halutut viivat
                objLine2d = objLines2d.AddBy2Points(-0.154, -0.085,
leveys, -0.085)
                objLine2d = objLines2d.AddBy2Points(leveys, -0.085,
leveys, 0.085)
                objLine2d = objLines2d.AddBy2Points(leveys, 0.085,
-0.154, 0.085)
                objLine2d = objLines2d.AddBy2Points(-0.154, 0.085,
-0.154, -0.085)
                'Sidotaan viivat relaatioilla yhteen
                objRelations2d = objProfile.Relations2d
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(1), 1, objLines2d.Item(2), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(2), 1, objLines2d.Item(3), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(3), 1, objLines2d.Item(4), 0)
                objRelation2d = objRela-
tions2d.AddKeypoint(objLines2d.Item(4), 1, objLines2d.Item(1), 0)

                'Suljetaan profiili
                objPro-
file.End(SolidEdgePart.ProfileValidationType.igProfileClosed)
                'Piilotetaan profiili näkyvistä
                'objProfile.Visible = False

                'Vaihdetaan kuvannoksi: ISO view.

objApp.StartCommand(CType(SolidEdgeConstants.PartCommandConstants.P
artViewISOView,  _
                                        SolidEdgeFrame-
work.SolidEdgeCommandConstants))

                'Referenssi models kokoelmaan
                objModels = objSheet.Models
                'Luodaan profiilin pohjalta tab-piirre
                objModel = objModels.AddBaseTab(objProfile, Solid-
EdgePart.FeaturePropertyConstants.igRight)
                'Piilotetaan profiili
                objProfile.Visible = False

                'Sammutetaan varoitukset, jotta jos kokoonpanoa on
käytetty aiemmin
                'esittää SE varoituksen päälle tallentamisesta
                objApp.DisplayAlerts = False
                'Tallennetaan 2D-kuva
                objSheet.SaveAs("D:\SE\Kokoonpanot\" & tiedosto & "
jalka.psm")
                'Palautetaan varoitukset päälle
                objApp.DisplayAlerts = True

                objSheet.Close()
            End If
```

```
            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()

            'Otetaan referenssi dokumentti kokoelmaan
            objDocuments = objApp.Documents
            'Tsekataan, että löytyy aktiivinen kokoonpano
            If objDocuments.Count > 0 Then
                'Yritetään ottaa yhteyttä aktiiviseen kokoonpanoon
                objAssembly = CType(objApp.ActiveDocument, Solid-
EdgeFramework.SolidEdgeDocument)
            End If

            'Jos kokoonpanossa on jalka, niin lisätään pohjalevy
            If jalka = True Then
                'Haetaan Occurences-kokoelman referenssi
                objOccurrences = objAssembly.Occurrences
                'Lisätään kokoonpanoon halutun mukaiset osat:
                'Kuori:
                objOccurrence =
objOccurrences.AddByFilename("D:\SE\Kokoonpanot\" & tiedosto & "
jalka.psm")
                objOccurrence.Move(0, 0, -0.14)
            End If

            'Annetaan ohjelman levähtää hetki
            objApp.DoIdle()

            'Jos kokoonpanossa on laippa, niin lisätään laipan levy
            If laippa = True Then
                'Haetaan Occurences-kokoelman referenssi
                objOccurrences = objAssembly.Occurrences
                'Lisätään kokoonpanoon halutun mukaiset osat:
                'Kuori:
                objOccurrence = objOccurrenc-
es.AddByFilename("D:\SE\Kokoonpanot\" & tiedosto & " laippa.psm")
            End If

            'Sammutetaan varoitukset, jotta jos kokoonpanoa on
käytetty aiemmin
            'esittää SE varoituksen päälle tallentamisesta
            objApp.DisplayAlerts = False
            'Tallennetaan 2D-kuva
            objAssembly.SaveAs("D:\SE\Kokoonpanot\" & tiedosto)
            'Palautetaan varoitukset päälle
            objApp.DisplayAlerts = True

        Catch ex As Exception
            Console.WriteLine(ex.Message)
            'Vapautetaan muisti
        Finally
            If Not (objApp Is Nothing) Then
                Marshal.ReleaseComObject(objApp)
                objApp = Nothing
            End If
            If Not (objDocuments Is Nothing) Then
                Marshal.ReleaseComObject(objDocuments)
                objDocuments = Nothing
            End If
```

```
If Not (objAssembly Is Nothing) Then
    Marshal.ReleaseComObject(objAssembly)
    objAssembly = Nothing
End If
If Not (objSheet Is Nothing) Then
    Marshal.ReleaseComObject(objSheet)
    objSheet = Nothing
End If
If Not (objProfileSets Is Nothing) Then
    Marshal.ReleaseComObject(objProfileSets)
    objProfileSets = Nothing
End If
If Not (objProfileSet Is Nothing) Then
    Marshal.ReleaseComObject(objProfileSet)
    objProfileSet = Nothing
End If
If Not (objProfiles Is Nothing) Then
    Marshal.ReleaseComObject(objProfiles)
    objProfiles = Nothing
End If
If Not (objProfile Is Nothing) Then
    Marshal.ReleaseComObject(objProfile)
    objProfile = Nothing
End If
If Not (objRefplanes Is Nothing) Then
    Marshal.ReleaseComObject(objRefplanes)
    objRefplanes = Nothing
End If
If Not (objPPlane Is Nothing) Then
    Marshal.ReleaseComObject(objPPlane)
    objPPlane = Nothing
End If
If Not (objRelations2d Is Nothing) Then
    Marshal.ReleaseComObject(objRelations2d)
    objRelations2d = Nothing
End If
If Not (objRelation2d Is Nothing) Then
    Marshal.ReleaseComObject(objRelation2d)
    objRelation2d = Nothing
End If
If Not (objLines2d Is Nothing) Then
    Marshal.ReleaseComObject(objLines2d)
    objLines2d = Nothing
End If
If Not (objLine2d Is Nothing) Then
    Marshal.ReleaseComObject(objLine2d)
    objLine2d = Nothing
End If
If Not (objModels Is Nothing) Then
    Marshal.ReleaseComObject(objModels)
    objModels = Nothing
End If
If Not (objModel Is Nothing) Then
    Marshal.ReleaseComObject(objModel)
    objModel = Nothing
End If
If Not (objCircles2d Is Nothing) Then
    Marshal.ReleaseComObject(objCircles2d)
```

```
                objCircles2d = Nothing
            End If
            If Not (objCircle Is Nothing) Then
                Marshal.ReleaseComObject(objCircle)
                objCircle = Nothing
            End If
            If Not (objCuts Is Nothing) Then
                Marshal.ReleaseComObject(objCuts)
                objCuts = Nothing
            End If
            If Not (objCut Is Nothing) Then
                Marshal.ReleaseComObject(objCut)
                objCut = Nothing
            End If
            If Not (objOccurrences Is Nothing) Then
                Marshal.ReleaseComObject(objOccurrences)
                objOccurrences = Nothing
            End If
            If Not (objOccurrence Is Nothing) Then
                Marshal.ReleaseComObject(objOccurrence)
                objOccurrence = Nothing
            End If
            aProfiles = Nothing
            objType = Nothing
            prosessi = Nothing
            jalka = Nothing
            laippa = Nothing
            luokka = Nothing
            materiaali = Nothing
            lkm = Nothing
            maara = Nothing
            leveys = Nothing
            i = Nothing
        End Try
    End Sub

    Private Sub objButtonEvents_Help(ByVal hFrameWnd As Integer,
ByVal uHelpCommand As Integer) Handles objButtonEvents.Help
        System.Windows.Forms.MessageBox.Show("Tämä ohutlevy -addin
luo tietyllä nimisäännöllä " _
                                             & "tallennettuun
moottorikokoonpanoon jalka- ja laippalevyn")
    End Sub

    Private Sub AddInCmdEvents_OnCommand(ByVal CommandID As Inte-
ger) Handles AddInCmdEvents.OnCommand
        Ohutlevy()
    End Sub
End Class
```

LIITE 6: KOKOONPANOPIIRROS-MAKRON KOODI

```
Imports System.Runtime.InteropServices
Imports System.IO
Imports System.Text
Imports System.Diagnostics
Imports SolidEdgeFramework
Imports SolidEdgeDraft
'Tämä makro on toteutettu osana Tampereen teknillisen yliopiston
konetekniikan
'diplomityötä. Makro luo Solid Edgellä työpiirrustuksen tietyllä
nimeämiskäytännöllä
'luodusta kokoonpanosta, kun lähtötilanteessa ollaan ko. kokoonpano
auki.
'Tekijä: Mikko Junttila

Module Piirros
    Dim objApp As SolidEdgeFramework.Application = Nothing
    Dim objDocuments As SolidEdgeFramework.Documents = Nothing
    Dim objAssembly As SolidEdgeAssembly.AssemblyDocument = Nothing
    Dim objDraft As SolidEdgeDraft.DraftDocument = Nothing
    Dim objModelLinks As SolidEdgeDraft.ModelLinks = Nothing
    Dim objModelLink As SolidEdgeDraft.ModelLink = Nothing
    Dim objDrawingViews As SolidEdgeDraft.DrawingViews = Nothing
    Dim objDrawingView1 As SolidEdgeDraft.DrawingView = Nothing
    Dim objDrawingView2 As SolidEdgeDraft.DrawingView = Nothing
    Dim objDrawingView3 As SolidEdgeDraft.DrawingView = Nothing
    Dim objSheets As SolidEdgeDraft.Sheets = Nothing
    Dim objSheet As SolidEdgeDraft.Sheet = Nothing
    Dim objtextBoxes As SolidEdgeFrameworkSupport.TextBoxes = Noth-
ing
    Dim objtextBox As SolidEdgeFrameworkSupport.TextBox = Nothing
    Dim dimStyle As SolidEdgeFrameworkSupport.DimStyle = Nothing
    'Dim objPartsLists As SolidEdgeDraft.PartsLists = Nothing
    'Dim objPartsList As SolidEdgeDraft.PartsList = Nothing
    'Public objOsa As PartsList
    Dim objType As Type
    Dim prosessi() As Process
    Dim tiedosto As String
    Dim maara As String
    Dim lkm As Integer
    Dim paikka As Integer
    Dim materiaali As String

    Sub Main()
        Try
            'Koitetaan ottaa yhteys auki olevaan Solid Edgeen
            prosessi = Process.GetProcessesByName("edge")
            If prosessi.Count > 0 Then
```

```
                objApp = Mar-
shal.GetActiveObject("SolidEdge.Application")
            Else
                'Kerrotaan, ettei avonaisia Edgejä ole
                Console.WriteLine("Ei avonaisia Solid Edge -
prosesseja")
            End If
            objApp.Visible = True
            'Otetaan referenssi dokumentti kokoelmaan
            objDocuments = objApp.Documents
            'Tsekataan, että löytyy aktiivinen kokoonpano
            If objDocuments.Count > 0 Then
                'Yritetään ottaa yhteyttä aktiiviseen kokoonpanoon
                objAssembly = CType(objApp.ActiveDocument,
SolidEdgeFramework.SolidEdgeDocument)
            End If
            'Jos aktiivista kokoonpanoa ei ole, niin herjataan
            If objAssembly Is Nothing Then
                Throw New System.Exception("No active document.")
            End If



            'Tarkastetaan tiedoston nimi
            tiedosto = objAssembly.Name
            Console.WriteLine(tiedosto)
            'Tarkastetaan montako osaa kokoonpanossa on
            paikka = InStr(tiedosto, "x")
            maara = Microsoft.VisualBasic.Mid(tiedosto, paikka - 1,
1)
            Console.WriteLine(maara & " konetta")
            lkm = CInt(maara)
            paikka = InStr(tiedosto, "fe")
            If paikka = Nothing Then
                materiaali = "Alumiini"
            Else
                materiaali = "Valurauta"
            End If
            Console.WriteLine(materiaali)



            'Luodaan 2D-kuvat
            objDraft = objDocuments.Add("SolidEdge.DraftDocument")
            'Otetaan aktiviinen "sivu" referenssiksi
            objSheet = objDraft.ActiveSheet
            'Haetaan model link -kokoelma referenssiksi
            objModelLinks = objDraft.ModelLinks
            'Haetaan kokoonpanon tiedostonimi ja sijainti

            'Lisätään linkki kokoonpanoon
            objModelLink = objModelLinks.Add("D:\SE\Kokoonpanot\" &
tiedosto)
            'Otetaan piirrustus näkymä kokoelma referenssiksi
            objDrawingViews = objSheet.DrawingViews
            'Asetetaan piirrustusnäkymä1
            objDrawingView1 =
objDrawingViews.AddPartView(objModelLink, SolidEdgeDraft. _
```

```
ViewOrientationConstants.igFrontView, 1 / lkm, 0.2, _

0.34, SolidEdgeDraft.PartDrawingViewTypeConstants. _

sePartDesignedView)
            'Asetetaan piirrustusnäkymä2
            objDrawingView2 =
objDrawingViews.AddByAuxiliaryFold(objDrawingView1, 0.4, 0.34,
0.12, 0.34, 0.4, 0.12)
            objDrawingView2.ShowEdgesHiddenByOtherParts = False
            'Asetetaan piirrustusnäkymä3
            objDrawingView3 =
objDrawingViews.AddByFold(objDrawingView1,
FoldTypeConstants.igFoldRight, 0.47, 0.34)
            objDrawingView3.ShowEdgesHiddenByOtherParts = False




            'Lisätään piirrokseen tekstikenttä, jossa lukee
yrityksen sekä tilaajan tiedot
            'ja materiaali
            ' Get a reference to the active sheet.
            objSheet = objDraft.ActiveSheet

            'Haetaan referenssi textbox kokoelmaan
            objtextBoxes = CType(objSheet.TextBoxes, Solid-
EdgeFrameworkSupport.TextBoxes)

            Console.WriteLine("Creating a new textbox: ")

            paikka = InStr(tiedosto, "x")
            maara = Microsoft.VisualBasic.Left(tiedosto, paikka -
3)

            'Lisätään teksti laatikko
            objtextBox = objtextBoxes.Add(0.2, 0.4, 0)
            objtextBox.TextScale = 3
            objtextBox.VerticalAlignment = SolidEdgeFrameworkSup-
port.TextVerticalAlignmentConstants.igTextHzAlignVCenter
            objtextBox.Text = (maara & ", Materiaali: " &
materiaali)




            'Sammutetaan varoitukset, jotta jos kokoonpanoa on
käytetty aiemmin
            'esittää SE varoituksen päälle tallentamisesta
            objApp.DisplayAlerts = False
            'Tallennetaan 2D-kuva
            objDraft.SaveAs("D:\SE\Kokoonpanot\" & tiedosto &
".dft")
            'Palautetaan varoitukset päälle
            objApp.DisplayAlerts = True
```

```
Catch ex As Exception
    Console.WriteLine(ex.Message)
Finally
    'Varmistetaan, ettei muistiin jää mitään turhaa
    If Not (objtextBox Is Nothing) Then
        Marshal.ReleaseComObject(objtextBox)
        objtextBox = Nothing
    End If
    If Not (objtextBoxes Is Nothing) Then
        Marshal.ReleaseComObject(objtextBoxes)
        objtextBoxes = Nothing
    End If
    If Not (dimStyle Is Nothing) Then
        Marshal.ReleaseComObject(dimStyle)
        dimStyle = Nothing
    End If
    If Not (objSheet Is Nothing) Then
        Marshal.ReleaseComObject(objSheet)
        objSheet = Nothing
    End If
    If Not (objSheets Is Nothing) Then
        Marshal.ReleaseComObject(objSheets)
        objSheets = Nothing
    End If
    If Not (objDrawingView3 Is Nothing) Then
        Marshal.ReleaseComObject(objDrawingView3)
        objDrawingView3 = Nothing
    End If
    If Not (objDrawingView2 Is Nothing) Then
        Marshal.ReleaseComObject(objDrawingView2)
        objDrawingView2 = Nothing
    End If
    If Not (objDrawingView1 Is Nothing) Then
        Marshal.ReleaseComObject(objDrawingView1)
        objDrawingView1 = Nothing
    End If
    If Not (objDrawingViews Is Nothing) Then
        Marshal.ReleaseComObject(objDrawingViews)
        objDrawingViews = Nothing
    End If
    If Not (objModelLink Is Nothing) Then
        Marshal.ReleaseComObject(objModelLink)
        objModelLink = Nothing
    End If
    If Not (objModelLinks Is Nothing) Then
        Marshal.ReleaseComObject(objModelLinks)
        objModelLinks = Nothing
    End If
    If Not (objAssembly Is Nothing) Then
        Marshal.ReleaseComObject(objAssembly)
        objAssembly = Nothing
    End If
    If Not (objDraft Is Nothing) Then
        Marshal.ReleaseComObject(objDraft)
        objDraft = Nothing
    End If
```

```
            If Not (objDocuments Is Nothing) Then
                Marshal.ReleaseComObject(objDocuments)
                objDocuments = Nothing
            End If
            If Not (objApp Is Nothing) Then
                Marshal.ReleaseComObject(objApp)
                objApp = Nothing
            End If
            objType = Nothing
            prosessi = Nothing
            tiedosto = Nothing
            maara = Nothing
            lkm = Nothing
            paikka = Nothing
            materiaali = Nothing
        End Try
    End Sub
End Module
```