MALLINNUS- JA MATERIAALILASKENTA-OHJELMISTON KEHITYS

Oskari Suomalainen

Opinnäytetyö

Ammattikorkeakoulututkinto

Savonia
ammattikorkeakoulu

SAVONIA-AMMATTIKORKEAKOULU
OPINNÄYTETYÖ

Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Oskari Suomalainen	
Työn nimi Mallinnus- ja materiaalilaskentaohjelmiston kehitys	
Päiväys 31.5.2011	Sivumäärä/Liitteet 40+1
Ohjaaja(t) Lehtori Jukka Kinnunen	
Toimeksiantaja/Yhteistyökumppani(t) Timo Toropainen Foster Wheeler Energia Oy	

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää mallinnus- ja materiaalilaskentaohjelmistoa Foster Wheeler Energia Oy:lle. Ohjelmistoon oli tarkoitus luoda yksi uusi kiertopetikattilan painerungon komponentti.

Kehitys aloitettiin tutustumalla PDMS -ympäristöön, jonka aliohjelmana mallinnus- ja mallinnuslaskentaohjelmisto tulisi toimimaan. Alussa tuli myös suunnittella kuinka uusi kattilakomponentti tulisi luoda ohjelmakoodiin. Varsinainen ohjelmisto luotiin Visual Studio 2005:llä käyttämällä C#-ohjelmointikieltä ja sitä ajettiin Aveva PDMS 12 version aliohjelmana mallintamiseen.

Opinnäytetyön tuloksena mallinnus- ja materiaalilaskentaohjelmistoon on lisätty yksi uusi kiertopetikattilan komponentti. Komponentin mallinnus ja materiaalilaskenta saatiin tehtyä projektin alussa sovitulla tavalla.

Avainsanat
C#, PDMS, CFB, Convection Cage, Mallintaminen

SAVONIA UNIVERSITY OF APPLIED SCIENCES
THESIS

Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Oskari Suomalainen			
Title of Thesis Development of Modeling and Material Calculation Software			
Date	31.5.2011	Pages/Appendices	40+1
Supervisor(s) Lecturer Jukka Kinnunen			
Project/Partners Timo Toropainen Foster Wheeler Energia Oy			

Abstract

The goal of the thesis was to develop modeling and material calculations software for Foster Wheeler Energia Oy. The meaning was to create new component in the software for the pressure hull of circulating fluidized bed.

The development was started by making oneself familiar with PDMS environment under which the modeling and material calculations software will be run. Also a plan how to create a new boiler component for the software was made. The software was made using Visual Studio 2005 with C# programming language and it was run under Aveva PDMS 12 as a sub program to model components.

As a result of the thesis one component for the modeling and material calculations software was added. Modeling and material calculations of the component were as designed at the start of the project.

Keywords

C#, PDMS, CFB, Convection Cage, Modelling

ALKUSANAT

Insinöörityö on tehty Foster Wheeler Energia Oy:lle Varkauden toimipisteeseen. Haluan kiittää insinöörityön ohjaajaa lehtori Jukka Kinnusta sekä Foster Wheeler Energia Oy:ltä Timo Toropaista, Ari Ojalaa, Mikko Pakarista ja muita projektissa olleita henkilöitä, neuvoista, avusta ja mahdollisuudesta tehdä insinöörityö Foster Wheeler Energia Oy:lle.

Varkaudessa 31.5.2011

Oskari Suomalainen

SISÄLTÖ

LYHENTEET

API	=	Application Programming Interface. Rajapinta jonka avulla mahdollista käyttää muita ohjelmia ohjelmakoodissa
CFB	=	Circulating Fluidized Bed. Kiertopetikattila
Comos	=	Tuotteiden koko elinkaaren aikaiseen ylläpitoon ja suunnitteluun tarkoitettu hallintaohjelmisto.
CSV	=	Comma Separated Value. Tiedostomuoto, jossa esimerkiksi taulukosta luettu tieto eritellään puolipisteillä yhdeksi merkki jonoksi.
DSM	=	Design Standard Manual. FW:n käyttämä suunnitteluohjekokoelma.
IDE	=	Integrated Development Environment. Ohjelmistoympäristö, minkä avulla ohjelmoija luo uusia ohjelmia.
MVC	=	Model View Control. Ohjelmointiarkkitehtuuri
PDMS	=	Plant Design Management System. Kattiloiden ja tehdas kokonaisuuksien mallinnusohjelma.
UML	=	Unified Modelling Language. Mallinnuskieli, jolla pyritään kuvaamaan ohjelmien rakennetta

1 JOHDANTO

Insinöörityö tehtiin Foster Wheeler Energia Oy:lle yhtenä osana isompaa TEKES:n rahoittamaa projektia. Insinöörityö tavoitteena oli kehittää mallinnus- ja materiaalilaskentaohjelmisto CFB-kattiloita eli kiertopetikattiloita varten Foster Wheeler Energia Oy:lle. Ohjelmiston tarkoitus olisi nopeuttaa kattilaprojektien tarjousvaiheen mallinnusta ja materiaalilaskentaa. Insinöörityöni lopullisena tavoitteena oli saada lisättyä mallinnus ja materiaalilaskenta yhdelle kattilakomponentille sekä muuten kehittää ohjelmistoa. Insinöörityö aloitettiin tutustumalla edellisen ohjelmoijan tekemään ohjelmapohjaan, jonka pohjalta kasattiin oma komponentti ohjelmaan.

Insinöörityössä selitetään Foster Wheelerin toimintaa ja heidän kehittämää kiertopetikattilatekniikkaa. Pääpaino insinöörityössä on kuitenkin perehtyminen ohjelmakoodin luomiseen ja mitä työkaluja käytettiin ja miten sitä luotiin. Lopuksi on Convection Cage-komponentista tarkemmin, eli kuinka uusi komponentti lisättiin ohjelmaan ja kuinka se toimii.

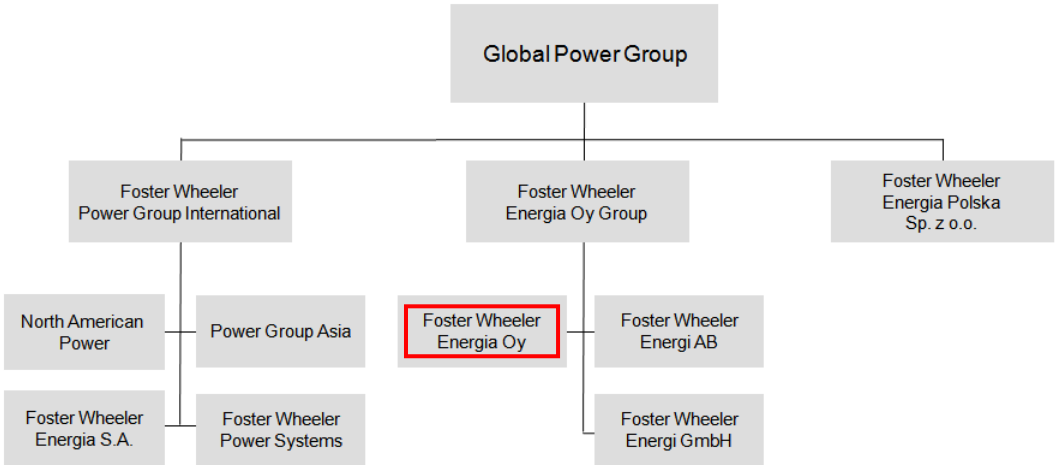
2 YRITYKSET

Insinöörityö oli osa isompaa projektia, FWE NW 2012 -projektia, joka tehtiin Foster Wheeler Energia Oy:lle Varkauden toimipisteeseen. Työnantajana toimi Savonia ammattikorkeakoulu Varkaudessa ja varsinainen työskentely tehtiin Foster Wheeler Energia Oy:llä.

Yritykset toimivat yhteistyössä Tekesin rahoittamassa Digitaalinen tuoteprosessi -projektissa, jolla pyritään vahvistamaan yritysten kasvu-mahdollisuuksia ja kilpailukykyä. Projektin tarkoitus on digitalisoida nykyinen dokumenttikeskeinen hallinta luoden Foster Wheeler Energia Oy:lle digitaalisen tuotemallin, joka kattaisi koko laitosprojektin elinkaaren.[1]

2.1 Foster Wheeler

Foster Wheeler perustettiin jo vuonna 1884, mutta vasta vuoden 1927 yritysfuusion jälkeen tunnettiin nykyisellä nimellä. Vuonna 2009 Foster Wheeler työllisti noin 13000 henkilöä 28 eri maassa ja liikevaihto oli yli viisi miljardia dollaria. Foster Wheeler Energia Oy on osa maailmanlaajuista Foster Wheeler AG -konsernia. Konserni muodostuu kahdesta osasta The Global Engineering & Construction (E&C) Groupista ja The Global Power Groupista.



Kuva 1. The Global Power Groupin rakenne

Foster Wheeler Energia Oy:llä on toimipisteet Espoossa ja Varkaudessa. Yrityksellä on myös tytäryhtiöt Saksassa ja Ruotsissa. Yhteisesti niitä kutsutaan Foster Wheeler Energia Oy Groupiksi. FWEOY Group:llä työskentelee noin 500 henkilöä, joista suurin osa Suomessa ja liikevaihto vuonna 2007 oli yli 400 miljoonaa euroa. (Kuva 1)

FWEOY Group on keskittynyt höyrykattiloiden suunnitteluun ja valmistamiseen. Yritys pyrkii kehittämään tehokkaita ja luontoa säästäviä ratkaisuja. Kiertopetikattila tekniikassa (CFB) FWEOY Group on maailman kärkeä tekniikassa noin 50% markkinaosuudella (2009). FWEOY Group tarjoaa myös muita tuotteita ja palveluita, kuten kuplapetikattiloita, kaasuttimia, jätelämpökattiloita, kattilalaitosten toimituksia sekä kunnossapitoa, huoltoa ja koulutusta. [2,3]

2.2 Savonia ammattikorkeakoulu

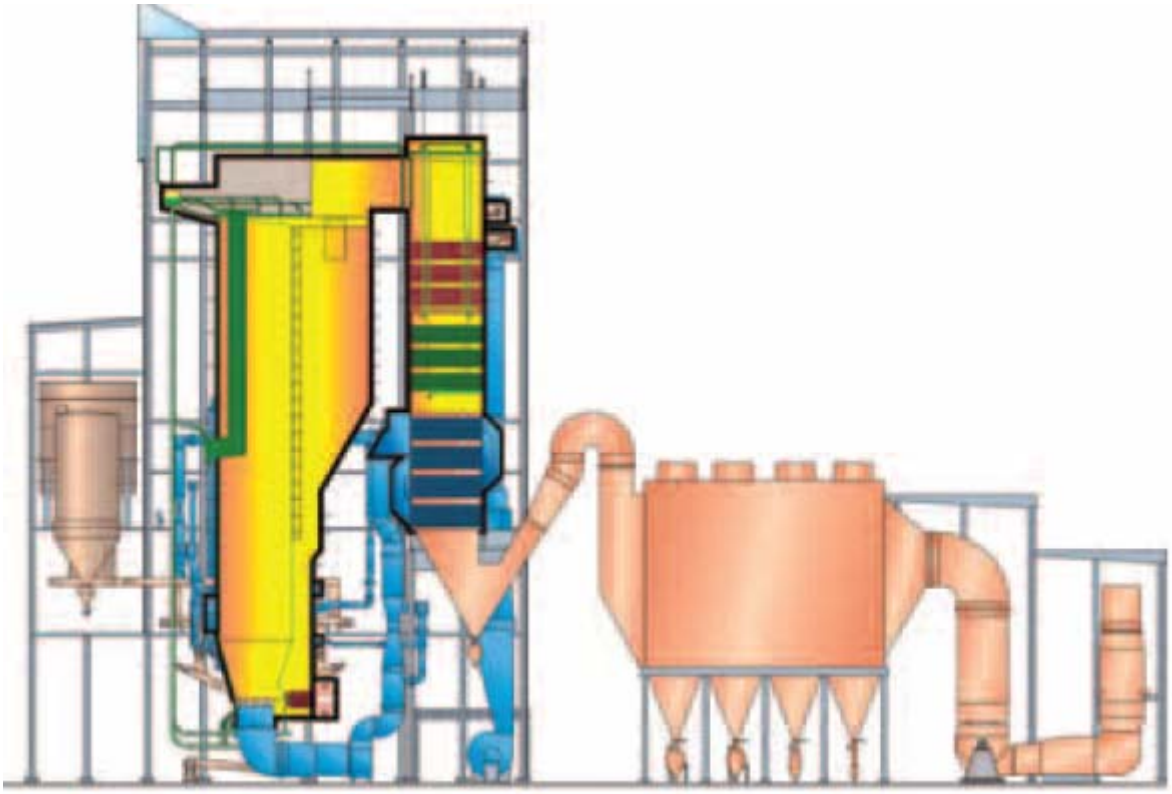
Savonia ammattikorkeakoulu perustettiin 90-luvulla ja tunnettiin ennen vuotta 2004 nimellä Pohjois-Savon ammattikorkeakoulu. Tänä päivänä Savonia on yksi suurimmista ammattikorkeakouluista Suomessa. Savonia toimii viidessä kunnassa Pohjois-Savon alueella. Kunnat ovat Varkaus, Kuopio, Iisalmi, Kiuruvesi ja Lapinlahti, mutta vain kolmessa ensimmäisenä mainitussa on koulutusyksiköt. Koulutusaloja Savoniassa on tarjolla seitsemän ja siellä opiskelee noin 6000 opiskelijaa. [4]

Savonian tarkoituksena oli tehdä projektissä yhteistyötä Foster Wheelerin kanssa. Savonian tehtävänä oli hoitaa projektin johtaminen ja työntekijöiden toimittaminen, Foster Wheelerin toimiessa työpaikkana ja asiantuntijana. Yhteistyön kautta Savonian tavoite olisi lisätä osaamistaan eri alueilla ja jatkossa käyttää uutta osaamista omissa palveluissaan ja siirtää sitä eteenpäin.

3 KIERTOPETIKATTILA

Foster Wheeler Energia Oy on ollut kehittämässä kiertopetikattilatekniikkaa alusta lähtien. Kiertopetikattilatekniikan kehitys alkoi 1960-luvulla Suomessa ja ensimmäinen energiaa tuottava kiertopetikattila valmistui vuonna 1979. Tänä päivänä on menossa kiertopetikattilatekniikan toinen sukupolvi ja uusia tekniikoita kehitetään parantamaan kattiloita kokoajan. Ensimmäinen toisen sukupolven kiertopetikattila valmistui vuonna 1992. (Kuva 2)

Uusin tekniikka on ylikriittinen, joka parantaa hyötysuhdetta 5-10% ja sitä hyödynnettiin ensimmäisen kerran 2009. Kiertopetikattilat on pyritty kehittämään hyvällä hyötysuhteella verrattuna aikaisempiin teknologioihin. Kiertopetikattiloita tarjotaan 5 MW aina 800 MW asti ja teholtaan suurin valmistettu kattila on 460 MW. [5]



Kuva 2. Kiertopetikattila

3.1 Kiertopetikattilatekniikan etuja

Kiertopetikattilatekniikalla on monia etuja, jotka ovat tehneet siitä yhden suosituimmista kattilatekniikoista. Yksi isoimmista eduista tänä päivänä on ympäristöystävällisyys. Tekniikan avulla on saatu vähennettyä NOx ja SOx eli typpi- ja rikkioksidipäästöjä merkittävästi verrattuna aikaisempiin teknologioihin. Tämän vuoksi kiertopetikattilatekniikka saavuttaa helposti nykyaikaiset tiukat päästövaatimukset. Myöskään ylimäärisiä ja monimutkaisia kemikaalien puhdistusjärjestelmiä ei tarvita. Kiertopetikattiloilla on erittäin korkea käytettävyysaste, yli 98%, joten ne ovat erittäin luotettavia. Kattilat voidaan räätälöidä juuri sopivaksi käyttötarkoitusta ja -paikkaa varten, kuitenkin säilyttäen yksinkertaisen rakenteen.

Toinen isoimmista eduista on monipuolinen polttoaineen käyttö ja monen eri polttoaineen yhdenaikainen käyttäminen. Petimateriaalin suuri lämpö-kapasiteetti auttaa tasaamaan polttoaineen laatuheilahteluita ja mahdollistaa huonolaatuisten (esimerkiksi kosteiden) polttoaineiden käytön. [5]

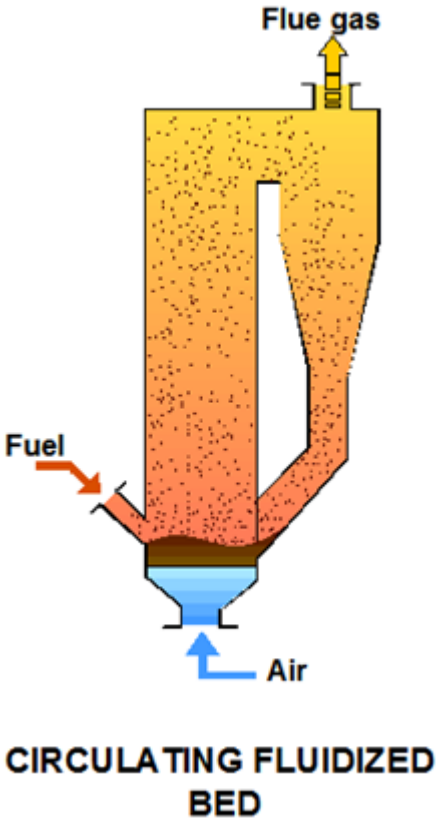
Esimerkkejä käytettävistä polttoaineista:

- Hiili (kivihiili, ruskohiili, jätehiili)
- Öljy
- Puumateriaali (purkupuu, hake, puujäte)
- Maatalousjäte
- Turve
- Paperijäte
- Kaasu

3.2 Toimintaperiaate

Kiertopetikattilan toimintaperiaate on lyhyesti ja yksinkertaisesti selitettynä seuraavanlainen:

Polttoaine (Fuel) syötetään noin 750-950°C kattilaan, jossa se sekoittuu hiekkaan ja kalkkikiveen (Ruskea kerros). Hiekka toimii lämmöntasaajana ja kalkkikiveä käytetään sitomaan rikkiä. Ilmaa (Air) puhaltamalla pohjasta hiekka saadaan kiertämään kattilaa savukaasujen seassa. Hiekka erotetaan savukaasusta (Flue gas) syklonissa ja palautetaan takaisin tulipesään. (Kuva 3)



Kuva 3. Toimintaperiaate

4 TYÖVÄLINEET

Pääasiallisina työvälineinä käytettiin Aveva PDMS -mallinnusohjelmaa ja Microsoft Visual Studiota, jossa ohjelmointikielenä toimi C#. Vähemmän käytössä oli Excel, jota suurimmaksi osaksi käytettiin ohjelmassa rajapinnan kautta.

4.1 AVEVA PDMS

Avevan jo vuonna 1967 julkaisema tietokantapohjainen tehtaiden ja laitosten mallinnusohjelma, jota myös kutsutaan 3D CAD:ksi. PDMS tulee sanoista Plant Design Management System. Tietokantapohjaisuuden ansiosta ohjelmasta saa helposti esimerkiksi raportteja ja työpiirrustuksia mallinnetuista kuvista. PDMS on myös erittäin monipuolinen ja räätälöitävissä yrityksen käyttötarkoituksiin sopivaksi. [6]

4.2 Microsoft Visual Studio

Visual Studio on Microsoftin kehittämä ohjelmointiympäristö (IDE). Se julkaistiin ensimmäisen kerran vuonna 1997 ja se tukee Visual Basic, C++, C# ja J# kieliä. Ohjelmalla voidaan kehittää konsoli-, graafisia käyttöliittymä- sekä verkkosovelluksia ja sovellukset toimivat kaikilla Windows -alustoilla.

4.2.1 C#

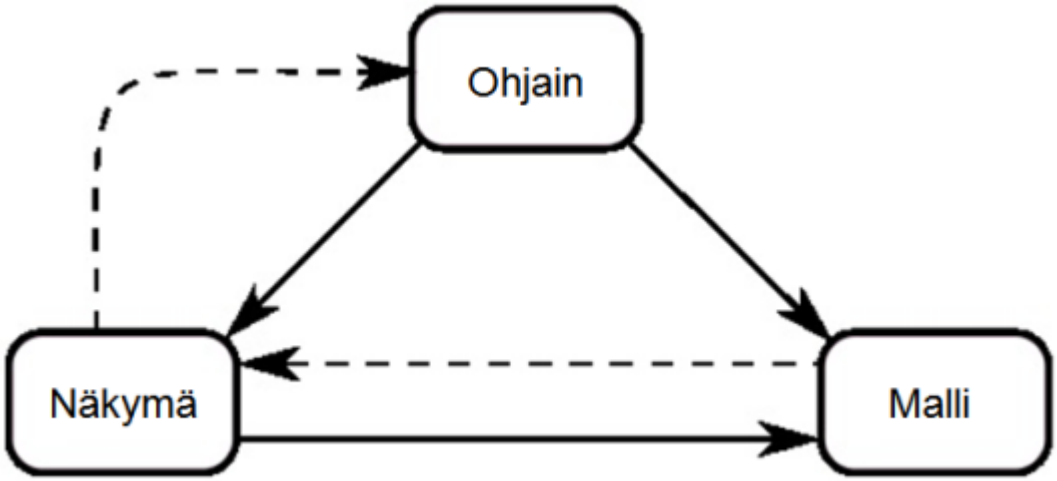
C# on Microsoftin kehittämä ohjelmointikieli. Kielessä yhdistyy C++ tehokkuus ja Javan helppokäyttöisyys. Ensimmäinen versio julkaistiin vuonna 2002 ja uusin versio on vuonna 2010 julkaistu C# 4.0.

5 OHJELMOINTI ARKKITEHTUURIA JA RAJAPINNAT

Teoriassa ohjelma oli tarkoitus tehdä MVC -arkkitehtuurin mukaan, jolloin vain viereiset moduulit kommunikoisivat keskenään. Kuitenkin käytännössä ohjelman moduulit sulautuivat osittain yhteen ja ohjelman muokkaaminen MVC -mallin mukaiseksi myöhemmin olisi ollut käytettävän ajan puitteissa vaikeaa.

5.1 MVC -arkkitehtuuri

MVC -arkkitehtuuri on yksi ohjelmistoarkkitehtuurityyleistä ja johtaa juurensa norjalaisen Trygve Reenskaugin kehittämään malliin vuodelta 1979. MVC lyhennys tulee sanoista Model, View ja Controller eli suomeksi malli, näkymä ja ohjain. Käytännössä tarkoittaa ohjelman jakamista kolmeen erilliseen moduuliin. Materiaalilaskentaohjelmassa lisättiin myös Database eli tietokantamoduuli. Kuvassa 4 on esitetty perinteinen MVC-malli.



Kuva 4. Perinteinen MVC -malli

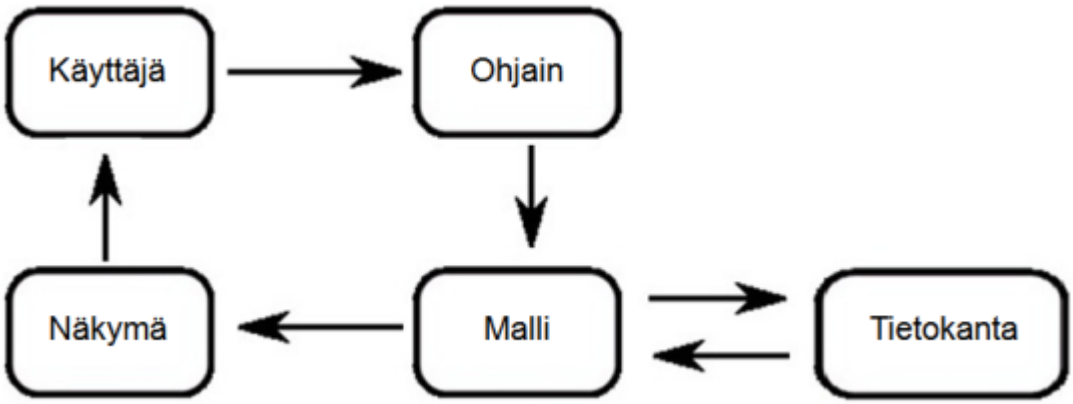
Jako moduuleihin tuo isoimmissa graafisissa ohjelmissa paljon hyötyjä pidemmällä aika välillä. Muutoksien tekeminen ja testaaminen ohjelmakoodiin helpottuu huomattavasti, kun moduulit ovat toisistaan riippumattomia. Kun tekee muutoksen esimerkiksi malliin ei tarvitse miettiä tarvitseeko ohjaimessa tai näkymässä käydä myös tekemässä muutoksia suorien riippuvuuksien puuttuessa. Malli suunnitellaan itsenäiseksi ja täten ei ole riippuvainen näkymästä tai ohjaimesta, joten se voidaan testata myös itsenäisenä.

Ohjelmaan on tällöin mahdollista suunnitella useampia käyttöliittymiä, esimerkiksi normaali graafinen Windows -ohjelma ja web -käyttöliittymä ja ohjaimia tekemättä muutoksia malliin.

MVC -arkkitehtuurin haittapuolina voidaan mainita ohjelmakoodin lisääntynyt monimutkaisuus. Suuret muutokset mallin rajapintaan, jota näkymä ja ohjain käyttävät, vaativat muutoksia myös ohjaimen ja näkymän ohjelmakoodiin. Haittapuolena voidaa myös mainita ohjaimen ja näkymän vaikea tarkka erottaminen käytännössä. [7]

5.1.1 Moduulien tehtävät ja sisältö

MVC -arkkitehtuuri toimii tavallisesti seuraavasti (Kuva 5):
Käyttäjä luo tapahtumia esimerkiksi painamalla nappia käyttöliittymästä. Ohjain käsittelee kyseiset tapahtumat muuttamalla ne mallin ymmärtämään muotoon ja lähettää tapahtumat mallille. Malli vaihtaa tietoa tarpeen mukaan tietokannan kanssa ja lähettää tiedot näkymälle. Näkymä päivittää tarvittavat tiedot ja näyttää ne käyttäjälle ja kierto voi alkaa uudestaan.



Kuva 5. MVC -malli käytännössä

Malli sisältää ohjelman keskeisen datan ja funktiot. Myös suurin osa ohjelmakoodista tulee tavallisesti malliin. Näkymässä on ulkoasuun liittyvät ohjelmakoodin osat ja ohjaimessa tiedonvälitykseen ja -kääntämiseen tarvittavat osat. Tietokantamoduuli sisältää PDMS ja Excel -rajapintojen ohjelmakoodit.

Kuten aikaisemmin mainittiin, ei materiaalilaskentaohjelmassa noudatettu puhtaasti MVC -arkkitehtuuria, joten moduulit sulautuivat osittain yhteen. Tähän vaikutti osittain eri ohjelmoijien tyylit tuottaa ohjelmakoodia sekä ohjelmakoodin osittainen sekavuus. Ohjain on melkein kokonaan näkymän sisällä ja pieni osa ohjaimesta on sulautunut malliin. Ohjain tekee myös suoria kutsuja tietokantamoduuliin.

5.2 Rajapinnat

Rajapinta eli API (Application Programmin Interface) on käsite, joka kuvaa esimerkiksi ohjelman vuorovaikutusta funktioilla toisen ohjelman komponenttien kanssa. Rajapinnan käyttö siis mahdollistaa ulkopuolisten ohjelmien osittaisen käytön varsinaisen ohjelman sisällä. Rajapinnasta riippuen kommunikointi tapahtuu esimerkiksi funktioiden, luokkien tai protokollien välityksellä. [8]

Ohjelmassa käytettiin PDMS ja Excel -ohjelmistojen rajapintoja tiedon välitykseen. Excel-rajapintaa käytettiin ohjelmassa komponenttien tiedon lukemiseen ja tietojen kirjoittamiseen Excelistä. PDMS -rajapinnan kautta käytiin hakemassa tietoa PDMS:n tietokannoista ja tarvittaessa vietiin tietoa PDMS:ään. Kaikki mallintaminen ja osien luonti PDMS -projektipuuhun hoidettiin PDMS-rajanpinnan kautta osittain monimutkaisilla funktioilla. PDMS -rajapinnan kautta kutsuttiin myös PDMS:n sisäisiä makrofunktioita.

5.3 Ohjelmointityyli

Ohjelmakoodi pyrittiin luomaan selkeästi ymmärrettäväksi ja helposti muokattavaksi uusien elementtien lisäystä varten. Muuttujat, funktiot ja luokat pyrittiin nimeämään samalla tavalla ja kuvaavasti joka paikassa, jolloin saadaan helpommin selville minkälainen muuttuja, funktio tai luokka on kyseessä ja mihin se liittyy ohjelmassa. Esimerkiksi `static` muuttujat kirjoitettin kokonaan isoilla kirjaimilla ja kaikkiin Convection Cageen liittyviin oli lisätty "CC" (`buildCC.cs`). Myös ohjelmakoodin kommentointi pyrittiin luomaan tarpeeksi yksinkertaiseksi, mutta kuitenkin riittävän tarkaksi, jolloin muut pääsisivät helpommin ohjelmakoodiin sisälle.

5.3.1 Olio-ohjelmointi

Olio-ohjelmoinnin tavoitteena on käyttää reaalimaailmaa mallina ohjelmoidessa. Tämä mahdollistaa esimerkiksi UML -mallinnuksen hyväksikäyttämisen olio-ohjelmoinnissa. Aikaisemmassa suositussa ohjelmointilähestymistavassa, proseduaalisessa ohjelmoinnissa, ohjelma sisälsi listan ohjeita, miten ohjelma toimi. Olio-ohjelmoinnissa ohjelma muodostuu olioista (object), jotka toimivat yhteistyössä toisten olioiden kanssa. Yksittäinen olio on käytännössä pieni kone, joka sisältää siihen liittyvää tietoa ja toiminnallisuutta. Olio -käsite ohjelmoitaessa helpottaa ohjelmakoodin käsittelyä ja luontia, kun asiat voidaan ajatella kokonaisuuksina, joissa kaikki niihin liittyvä on samassa paikassa. [9]

Olio-ohjelmoinnin periaatteiden mukaan tehty ohjelmakoodi on selkeämmin ymmärrettävää ja helpommin laajennettavissa kuin esimerkiksi perinteinen prosedulaalinen ohjelmakoodi. Pienissä yksinkertaisissa ohjelmissa olio-ohjelmoinnin käyttö on yleensä työlästä, joten sitä ei välttämättä kannata käyttää silloin. Olio-ohjelmointiin perustuvan ohjelman rakenne on työlästä rakentaa puhtaalta pyödältä, mutta kun runko on valmis, nopeutuu ohjelmointi paljon.

5.3.2 Periytyminen

Periytyminen jaetaa kahteen osaan, yliluokkaan (super class tai baseclass) ja aliluokkaan (sub class tai derived class). Aliluokka on yliluokasta johdettu uusi luokka ja aliluokka voi periä monia yliluokkia. Aliluokka perii kaikki yliluokan attribuutit ja rajapinnat. Yliluokan ominaisuuksia on mahdollista ylikirjoittaa ja aliluokkaan voi lisätä uusia attribuutteja ja omia rajapintoja. Periytyminen voidaan tehdä kahdella eri tapaa joko yksinperiytymisellä tai moniperiytymisellä. Yksinperiytymisellä tarkoitetaan, että yhdestä yliluokasta periytyy vain yksi aliluokka. Moniperiytymisessä yliluokalla voi taas olla monta aliluokkaa. Molemmilla tavoilla saadaan aikaan luokkahierarkia. Periyttämisellä vähennetään turhan ohjelmakoodin toistoa ja periytymisen avulla on helppo laajentaa jo olemassa olevaa ohjelmaa.

5.3.3 Kapselointi

Kapseloinnilla voidaan tarkoittaa olio-ohjelmoinnissa kahta asiaa: tiedon ja funktioiden yhteensitomista oliossa tai tiedon piilotusta. Kapseloinnilla voidaan estää suora viittaaminen olion muuttujiin ja funktioihin. Tämän avulla voidaan vaikuttaa tapaan, millä olio käsittelee ja tallentaa tietoa olion rajapintaan vaikuttamatta.

Tiedon piilotukseen käytetään suojaumääreitä. Käytetyimpiä suojamääreitä ovat public (julkinen), private (piilotettu) ja protected (suojattu). Public näkyy koko ohjelmassa, private näkyy vain luokan sisällä ja protected näkyy luokan sisällä ja aliluokissa. Piilotetun tiedon lukeminen ja tallentaminen olion ulkopuolelta tehdään yleensä niin sanottuja Get ja Set -funktioita käyttäen. Yleensä piilotetaan funktioita ja muuttujia, joita ei käytetä tai saa käyttää luokan ulkopuolella. Julkisilla funktioilla muodostetaan yhteys, jonka avulla oliota voidaan käyttää suunnitellulla tavalla. Suojattuja muuttujia ja funktioita käytetään, kun niitä tarvitaan olion ulkopuolella ja niitä ei saa muuttaa.

5.3.4 Monimuotoisuus

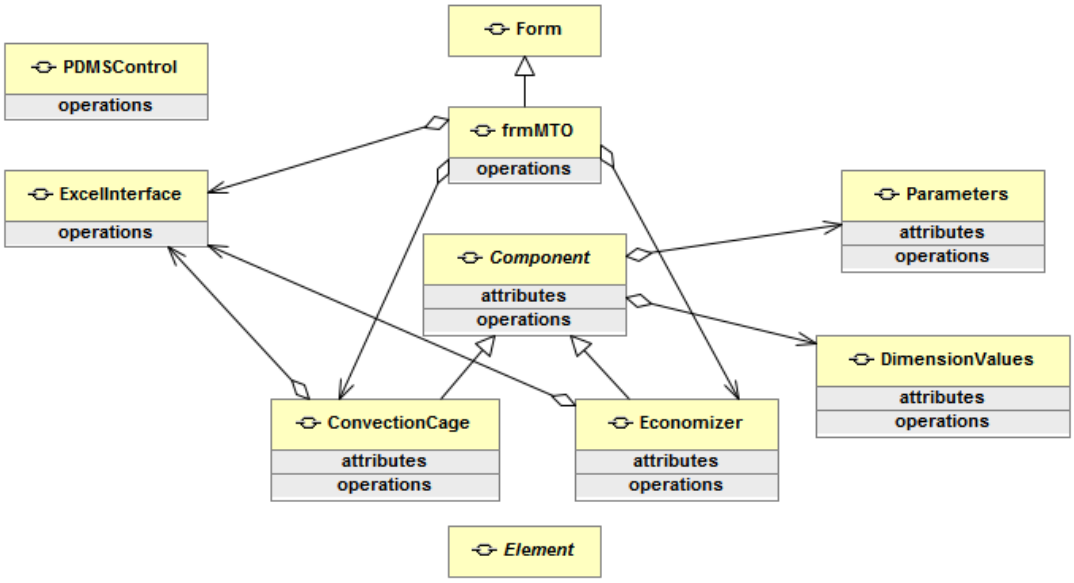
Monimuotoisuudella eli polymorfismilla tarkoitetaan olion tilanteesta riippuvaa käyttäytymistä ja se on luonnollista seurausta olioden periytymisestä, viestinvälityksestä ja korvaavuudesta. Se siis tarkoittaa, että yksi funktio voi suorittaa erinlaisia tehtäviä. Tärkeimpiä asioita monimuotoisuudessa ovat funktioiden kuormitus ja uudelleen määrittely eli ylikirjoitus.

Kuormituksella tarkoitetaan samannimisten funktioiden luomista olioon, mutta funktiolle menee eri argumentit, joiden avulla pystyy erottamaan funktioiden käytön. Ylikirjoitus viittaa taas ali- ja yliluokassa oleviin samannimisiin funktioihin. Aliluokassa voidaan tehdä ylikirjoitus samannimiselle funktiolle, jolloin se korvaa yliluokassa olevan toteutuksen, mutta argumenttien on oltava samoja molemmissa. Ylikirjoituksella voidaan myös tarkentaa yliluokan funktioita, jolloin aliluokassa oleva funktio ei korvaa vaan täydentää yliluokan funktioita. Ylikirjoittamisen ongelmana voidaan tosin nähdä ylikirjoituksen

vapaus ohjelmoitaessa, jolloin uusi funktio ei välttämättä vastaa toiminnaltaan täysin ylikirjoitettua funktiota.

5.4 Rakenne

Ohjelmointia on pyritty ajattelemaan moduulisesti eli tekemään yhtenevistä luokista samantyyliisiä. Esimerkiksi kaikki PDMS -toimintaan liittyvien luokkien nimessä on "PDMS" -liite. Alla olevassa kuvassa on esitelty ohjelman rakenne yksinkertaistettuna yleisellä tasolla. Nuolet kuvassa 6 kuvaavat, kuinka ohjelman pääluokat liittyvät toisiinsa.



Kuva 6. Ohjelman päärakennekuva

Ohjelma jakautuu kolmeen päämoduulitasoon. Form on näkymätaso eli käyttöliittymään liittyvät ohjelmakoodit. Component -taso taas pitää sisällään varsinaiset kattilakomponentit ja sen alle myös lisätään projektin edetessä muut komponentit. Alin taso eli Element taso on yksittäisten elementtien rakentamista varten, sisältää esimerkiksi putkien, kammioiden ja teräsrakenteiden rakennusta vastaavat ohjelmakoodin osat. Muita tasoja ohjelmakoodissa on rajapinnat, kuvassa vasemmalla puolella olevat Excel- ja PDMS -luokat ja kokoelmat sekä oikealla puolella kuvassa 6 olevat Parameters ja DimensionValues. Jokainen päätaso käyttää rajapintoja suoraan jollain tavalla ohjelmaa ajaessa, mutta vain Form ja Component -tasot käyttävät kokoelmia suoraan.

5.5 Testaus ja dokumentointi

Dokumentointi hoidettiin kahdella tavalla: lisäämällä suoraan koodiin sopivan tarkat selitykset jokaisesta tarpeellisesta funktiosta, oliosta ja muuttujasta ja kirjoittamalla yleisesti toiminnasta ja rakenteesta Word -dokumentti. Myös käyttöohje luotiin ohjelmaa varten. Käyttöohjeen piti olla sopivan tarkka, että ohjelmaa aikaisemmin käyttämätön ja ohjelmoinnista tietämätön pystyi käyttämään ohjelmaa. Käyttöohjeeseen tuli myös jokainen eri variaatio komponenteista, joita ohjelmalla pystyi mallintamaan.

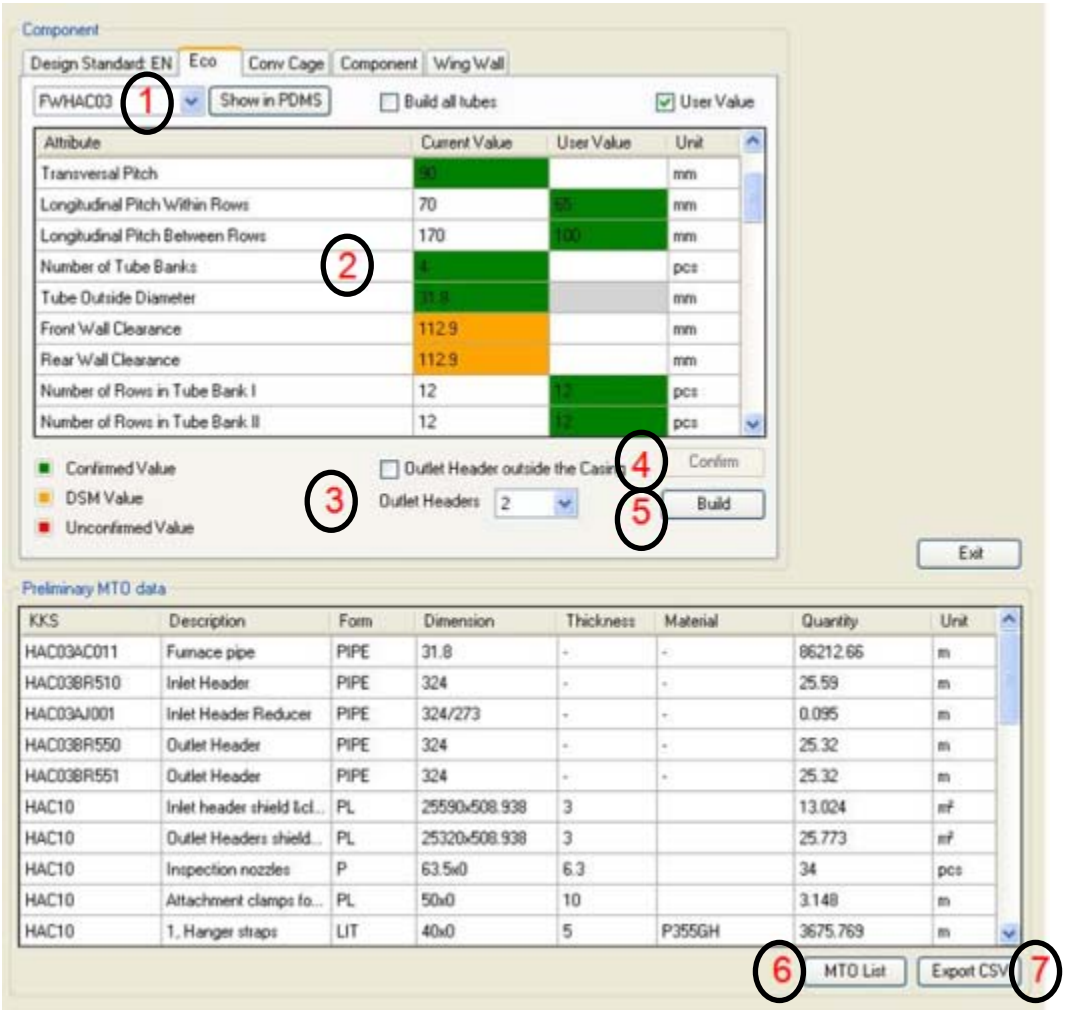
Ohjelman testaus hoidettiin pääasiallisesti manuaalisesti kokeilemalla eri arvoja ja toimintoja normaaleissa käyttötapauksissa ja myös tarkoituksellisesti vääränlaisesti, jotta ohjelma saataisiin kaatumaan. Ohjelmakoodin funktioille tehtiin myös erillisiä testiluokkia, joiden avulla pystyi pelkästään testaamaan ohjelman sisäisten funktioiden toimintaa. Funktion toiminta ei kuitenkaan tarkoittanut suoraan, että ohjelma toimisi oikein PDMS:n alla, joten niiden käyttö jäi vähemmälle.

6 MALLINNUS- JA MATERIAALILASKENTAOHJELMA

Ohjelma on suunniteltu nopeuttamaan Foster Wheeler Energia Oy:n tarjousvaiheen mallinnusta ja materiaalilaskuja, mutta ohjelmaa voidaan myös hyödyntää projektivaiheessa. Ohjelmaa ajetaan PDMS:n kautta aliohjelmana.

6.1 Ulkoasu ja käyttöliittymä

Käyttöliittymä pyrittiin pitämään ohjelmassa selkeänä ja yksinkertaisena.



Kuva 7. Käyttöliittymä

Käyttöliittymä (Kuva 7) on jaettu kahteen osaan. Ylempi osa pitää sisällään mallinnettavat kattilakomponentit. Alempi osa taas on materiaalimäärälaskuja ja niiden ulos vientiä CSV -tiedostona.

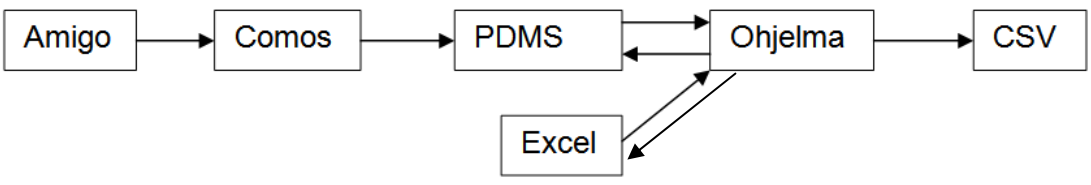
6.1.1 Toimintaperiaate

Ohjelman käyttö aloitetaan valitsemalla listasta (1) mallinnettava komponentti. Komponentin tiedot ladataan ylempään taulukkoon (2) PDMS-tietokannasta ja Excelistä. Mikäli taulukon (2) arvoja on tarpeellista muuttaa tai manuaalisesti lisätä puuttuvia arvoja, tehdään se "User Value" valinnan avulla. Taulukon (2) alapuolelta valitaan mallinnettavan komponentin variaatiot (3). "Confirm"-painikkeesta (4) ohjelma tarkistaa taulukossa (2) olevat tiedot ja värikoodaa arvot. Oranssi tarkoittaa arvon muuttamista Excelin kautta DSM-ohjeiden mukaiseksi, punainen solu kertoo arvon olevan virheellinen, ja vihreä solu, kun arvo on normaali. Komponentin mallinnus tapahtuu "Build" -painikkeesta (5), jolloin ohjelma mallintaa PDMS:ään annettujen tietojen mukaisen komponentin. (Kuva 7)

Tarpeen vaatiessa mallinnuksen jälkeen voidaan laskea materiaalimäärät ala-osan taulukkoon "MTO List" -painikkeen (6) avulla. "Export CSV" -painike (7) vie materiaalimäärät ohjelmasta CSV -tiedostona. (Kuva 7)

6.2 Tiedon kulku

Päätietolähteinä ohjelma käyttää Excelissä ja PDMS -tietokannassa olevaa tietoa. Excelissä oleva tieto on luotu DSM -ohjeiden ja yleisien mallinnus-sääntöjen mukaan manuaalisesti ja siellä olevaa tietoa käytetään harvemmin muuttumattomien tietojen lukemiseen. PDMS-tietokannassa on projekti-kohtaista tietoa, joka tuodaan tarpeen vaatiessa Comos -ohjelmasta. Amigo -laskentaohjelmisto tuottaa tietoa Comosiin kattilakomponenteille. Ohjelmasta pystyy mallinnuksen jälkeen tuottamaan materiaalimäärät CSV-tiedostona, jota voi käyttää muihin tarkoituksiin. Kuva 8 esittää tiedon kulun ohjelmaan.



Kuva 8. Tiedon kulku ohjelmaan

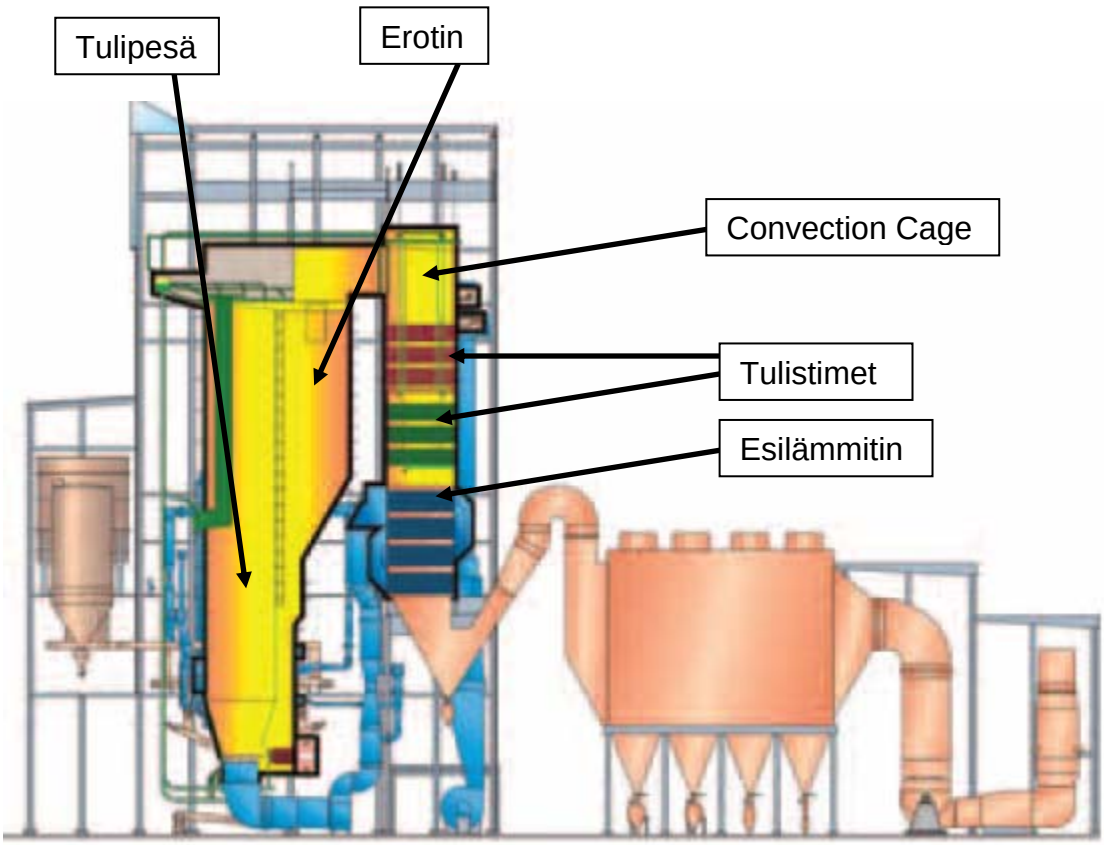
6.3 Lisäelementit ja jatkokehitys

Materiaalilaskentaohjelmaa kehitetään osissa käyttäen eri ohjelmoijaa joka elementissä. Jatkokehityksen kannalta on siis tärkeää, että omasta ohjelmakoodista tulisi sopivan selkeää ja yhdenmukaista aikaisemman ohjelmakoodin kanssa. Se tulee auttamaan seuraava ohjelmoijaa, joka pääsee helpommin sisälle koodiin ja pystyy korjaamaan ja käyttämään samoja ohjelmakoodin osia. Näin pystytään helposti vähentämään turhaa ohjelmakoodin toistamista. Osa luokista tuli myös suunnitella tavalla, jotta niitä pystyisi käyttämään monissa eri kiertopetikattilan elementtien mallinnuksessa ja tiettyjä osakokonaisuuksia pyrittiin myös suunnittelemaan yksinkertaisiksi ja monipuolisiksi laajempaa käyttöä varten eikä vain yhden komponentin kanssa.

7 CONVECTION CAGE

Convection Cage on yksi komponentti kiertopetikattilassa ja sijaitsee kattilalaitoksen takavedossa savukaasuvirrassa tulipesän ja erottimen jälkeen. Mallinuksen kannalta muita pääosia kiertopetikattilassa ovat: tulipesä eli furnace, erotin eli separator, tulistimet ja esilämmittimet.

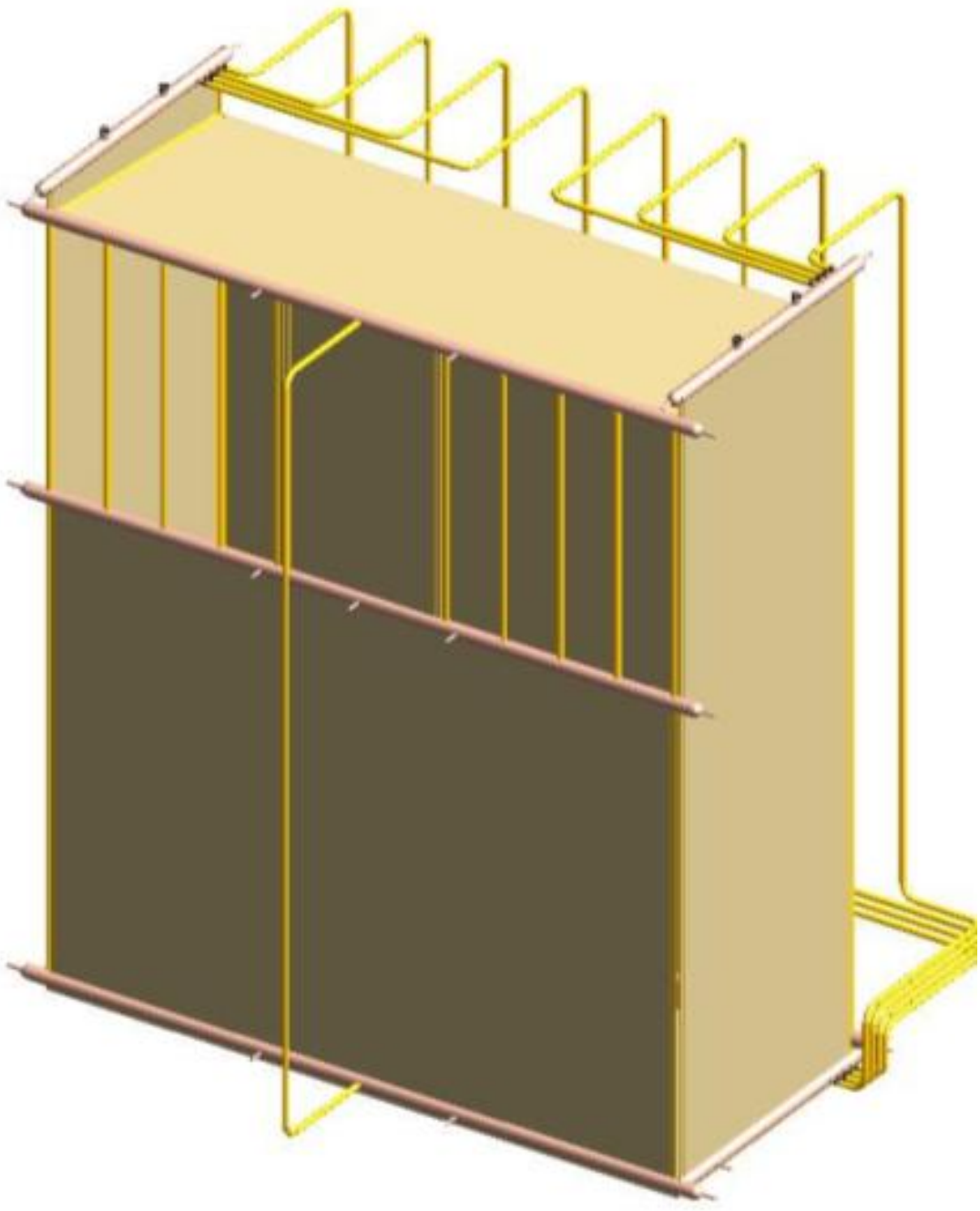
Convection Cage on käytännössä iso konvektiopinta, joka on muodoltaan suorakulmaisen häkin kaltainen. Convection Cage muodostuu eväputki-seinämistä ja sen sisällä virtaa savukaasu. Convection Cagen sisälle sijoitetaan yleensä tulistimet ja mahdollisesti reheaterit ja esilämmittimet. Näiden ja eväputkiseinämien tarkoitus on siirtää lämpöä savukaasusta, jotta mahdollisimman suuri osa lämmöstä saadaan talteen ja savukaasun lämpötilaa alennettua. (Kuva 9)



Kuva 9. Convection Cagem sijainti kattilassa

7.1 3D -Malli

Convection Cagen 3D -malli luodaan ohjelman kautta PDMS:ään. Convection Cagea ei mallinneta täydellisesti, koska materiaalilaskentaohjelma on tarkoitettu pääasiassa tarjousvaiheeseen. Tarkoituksena on mallintaa vain materiaalien puolesta riittävästi, jotta tarvittavat materiaalimäärät voidaan näin laskea tarjousta varten nopeammin kuin manuaalisesti mallintamalla. Lisäksi tulee huomioida layout suunnittelun tarpeet. (Kuva 10)



Kuva 10. Convection Cage 3D-malli

Mallinnustarkkuus ja tarvittavat variaatiot käytiin muun projektiryhmän kanssa läpi kokouksissa valiten yleisin mallipohja aikaisemmista projekteista. Projektin alussa oli tarkoitus vain mallintaa yksi variaatio ja yksinkertainen malli Convection Cagesta. Projektin edetessä mallia tarkennettiin ajan antaessa periksi lisäosilla ja variaatioita lisättiin. Eri variaatiot eivät paljon muuttaneet mallia Convection Cagen tapauksessa, koska se on rakenteeltaan kohtuullisen yksinkertainen kattilakomponentti.

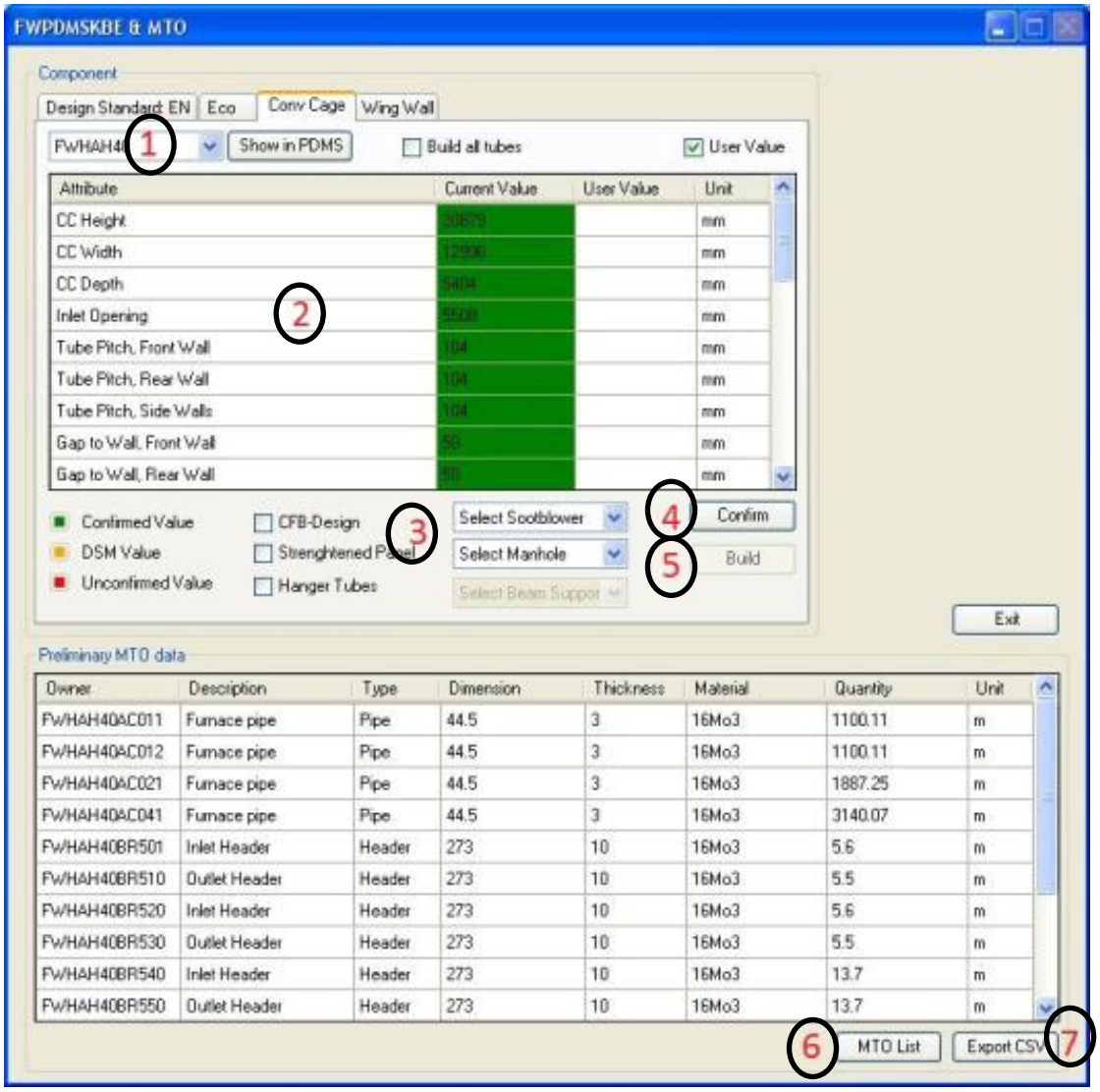
3D-malli muodostuu neljästä eri pääosasta ja lisäosista tarpeen mukaan. Neljä pääosaa ovat etuseinämä ja siinä oleva aukko, identtiset sivuseinämät ja takaseinämä joka käsittää myös katto-osan. Lisäosiana yllä olevassa kuvassa on Convection Cagen takana ja edessä olevat kannatusputket. Jokainen seinämä puolestaan rakentuu kolmesta elementissä, joiden avulla ne mallinnetaan. Nämä osat ovat putket, kammiot ja putkien välissä sijaitsevat evät, jotka mallinnettiin yhtenäisenä levynä mallin keventämisen vuoksi. Kammioihin mallinnettiin myös tarkastus- ja yhdysputkiyhteet tarpeen mukaan. Seinämiin mallinnetaan yleensä vain kaksi putkea reinoihin kuvaamaan kuinka putket tulisivat malliin, koska kaikkien putkien mallintaminen olisi hidasta ja raskasta käsitellä eikä varsinaisesti tuo lisähyötyä.

7.2 Käyttöliittymä ohjelmassa

Convection Cage välilehti näyttää hyvin samanlaiselta kuin muidenkin komponenttien välilehdet. Pääasialliset erot löytyvät kohdassa 3 olevista valinnoista.

Convection Cagen mallintaminen aloitetaan valitsemalla kohdasta yksi sopiva projekti. Ohjelma lataa kohdassa kaksi olevaan taulukkoon tarpeelliset tiedot Convection Cagen mallintamiseen. Taulukossa näkyy tietojen nimet, mahdolliset haetut arvot, käsinsyötetty arvo ja yksikkö arvolle. Kohdassa kolme voi valita lisäosia ja variaatioita mitä malliin haluaa, esimerkiksi kantoputket, vahvistettu paneeliosa, nuohoinluukku ja miesluukku. Confirm-painike käy läpi arvot ja tarkistaa että arvot ovat oikeinlaisia mallintamista varten. Build-painike mallintaa Convection Cagen PDMS:ään annettujen

tietojen perusteella. Alempaan taulukkoon saa materiaalimäärät Convection Cage -komponentille MTO List-painikkeella. Taulukossa näkyy tämän jälkeen kaikki tarvittava tieto jokaisesta materiaalimäärän suhteen merkkittävästä osasta komponentissa. Export CSV-painike vie taulukon arvot ulos ohjelmasta. (Kuva 11)



Kuva 11. Convection Cage välilehti

7.3 Ohjelmokoodin luomisesta

Ohjelmakoodi on pyritty tekemään yhdenmukaisesti edellisen komponentin kanssa, mutta kuitenkin yritetty kiinnittää huomioita kuinka parantaa ohjelmakoodin toimivuutta ja selkeyttä. Tarkoitus oli luoda ohjelmakoodi sopivan väljästi mahdollisten myöhempien lisäyksien vuoksi. Komponentin ohjelmointi aloitettiin hyvin karkeasti perusrungosta, jotta saatiin testattua

kaikkien uusien luokkien toimivuus ohjelmassa. Eli painiketta painamalla ohjelmassa mallintui yksinkertainen elementti PDMS:ään. Uudet luokat luotiin pääasiassa ottaen mallia edellisen komponentin vastaavista pyrkien yhdenmukaisuuteen. Hyötynä tässä oli, että huomasi helpommin, mitkä osat ohjelmakoodista mahdolisesti pystyisi yhdistämään, jolloin turhan ohjelmakoodin määrä vähenisi.

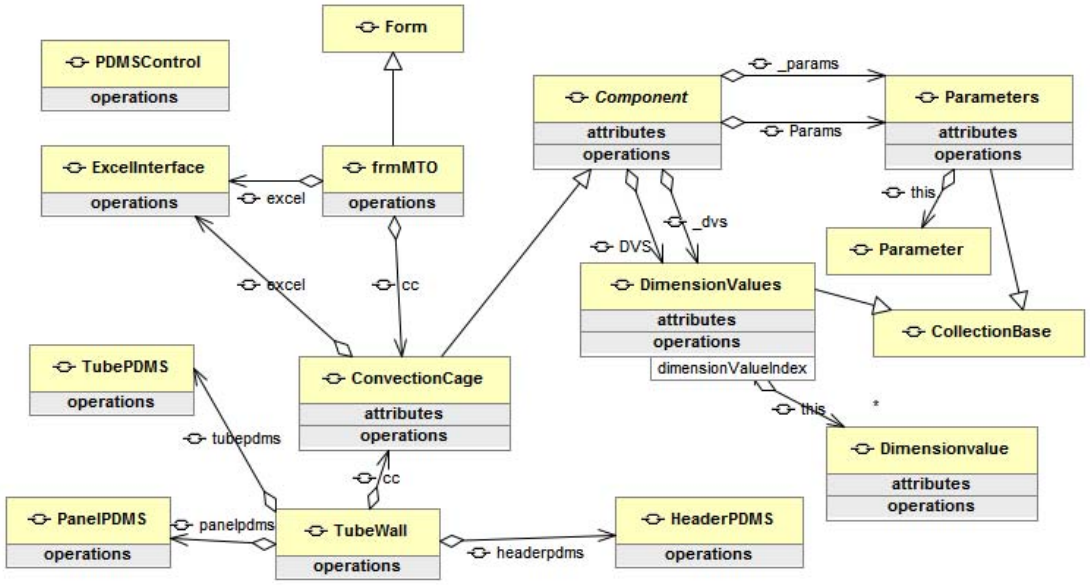
Ohjelmakoodia lähdettiin laajentamaan käytännössä keskeltä eli mallin-rakennuksesta. Mallinrakennus irroitettiin ohjelmakokonaisuudesta ja käsiteltiin omana yksinäisenä asiana, jolloin sitä oli helpompi jaotella ja ohjelmoida. Mallinrakennuksen ohjelmakoodin luontiin käytettiin iterointia. Ensimmäisellä iterointikierroksella luotiin ohjelmakoodi, joka mallinti perusversion komponentista ohjelmakoodin siisteyden ja toimivuuden jäädessä osittain taka-alalle. Seuraavilla iterointikierroksilla pääkohteena oli ohjelmakoodin siistiminen ja yhdistäminen paremmaksi kokonaisuudeksi. Convection Cagen ohjelmakoodi kävi läpi yhteensä noin kolme täydellistä muodollista muutosta projektin aikana. Loppuvaiheessa kuitenkaan rakenteellisia muutoksia ei tarvinnut paljoa tehdä, koska runko oli jo hyvin tehty. Näin uudet lisäykset ja muutokset oli helpompi ja yksinkertaisempi toteuttaa.

Tähän ohjelmointitapaan päädyttiin, koska kaikki informaatio kuinka malli tulisi luoda kokonaisuudessaan ei ollut suoraan projektin alussa käsillä ja malliin tuli muutoksia ja lisäyksiä projektin edetessä useasti. Ohjelmointitapa toimi mallinrakennusohjelmakoodissa hyvin, mutta muut lisäykset ja muutokset tehtiin kuitenkin normaalisti kerralla valmiiksi ja harvemmin palattiin tekemään rakenteellisia muutoksia. Projektin edetessä taka-alalla oli kuitenkin aina ajatus kuinka parantaa muuta ohjelmakoodia, vaikka ei suoraan olisi muokkaamassa juuri käytettävää ohjelmakoodia.

7.3.1 Rakenne ja toiminta

Ensimmäinen suunnitelma ohjelmakoodin rungon rakenteesta otettiin suoraan edellisestä komponentista. Samantien tuli kuitenkin ilmi rakenteita, joita pystyi yhdistämään komponenttien kesken yksinkertaistaen rakennetta. Jokaista

rakennetta ei kuitenkaan pystynyt täydellisesti yhdistämään pienten erojen tai muun rakenteellisen seikan vuoksi. Alla olevasta UML -luokkarakenne kuvasta näkyy kuinka ohjelma rakentuu Convection Cage -komponentin osalta ja mitenkä eri luokat riippuvat ja periytyvät toisistaan. Normaalit nuolet kuvastavat periytymistä ja erinlaiset nuolet assosiaatioita luokkien välillä. (Kuva 12)



Kuva 12. Convection Cage UML luokkarakenne

Form- ja frmMTO-luokka hoitavat kaiken ohjelman näkymään liittyvän tiedonkäsittelyn. Convection Cage-luokka on Convection Cage komponentin pääluokka jossa keskitetysti tapahtuu kaikki siihen liittyvä varsinainen tiedonkäsittely. Convection Cage-luokka periytyy Component-luokasta, jonka kautta se saa käyttöönsä Parameters- ja DimensionValues-kokoelmat. DimensionValues-kokoelma pitää sisällään Convection Cagen mallintamiseen liittyvät staattiset pääarvot, jotka tulevat suoraan PDMS:stä. Parameters-kokoelma taas pitää sisällään kaikki DimensionValues-kokoelmassa esiintyvät ja muut yleiset arvot, joita ohjelma käyttää suoraan tai välillisesti komponentin mallintamiseen.

ExcelInterface-luokka toimii rajapintana Exceliin, josta Convection Cage-luokka ja frmMTO-luokka käyvät lukemassa mallinnuksen sääntöjä ja tarkennus tietoja. Convection Cage-luokka myös vie tietoa Exceliin

mallinnuksen yhteydessä materiaalilaskuja varten. `PDMSControl`-luokka on taas varsinainen rajapintaluokka PDMS-ohjelmaa varten. Sitä ei käytetä ohjelmassa suoraan vaan pelkästään kutsutaan sen funktioita.

Convection Cagen päämallinnus luokka on `TubeWall`-luokka. Se pitää sisällään kaiken tarvittavan ohjelmakoodin Convection Cagen mallintamiseen eli käytännössä muutamia funktioita, joita yhdistämällä saadaan kokonainen komponentti kattilaan. Luokkaa on myös mahdollista jatkokäyttää muiden vastaavien komponenttien mallintamiseen mitkä ovat rakenteeltaan samanlaisia kuin Convection Cage. TubeWall-luokassa yksittäiset PDMS elementit yhdistetään sopivaksi kokonaisuudeksi, joka mallintaa ohjelmassa Convection Cagen. Luokka käyttää tähän PDMS -lisänimen omaavia luokkia, jotka vastaavat samoja elementtejä kuin mitä ne ovat varsinaisessa PDMS-ohjelmassa.

7.3.2 PDMS ohjelmakoodissa

PDMS:ää käytettiin ohjelmakoodissa melkein samalla tavalla kuin normaali PDMS -käyttäjä käyttäisi sitä "command line" toiminnon kautta PDMS:ssä. Eli käytännössä tarkoittaa sitä, että jokainen elementti ja attribuutin sijoitukset pitää tehdä yksittäisillä käskyillä. Ohjelmakoodissa tämä hoidettiin kokoamalla tarvittavat elementit ja niiden käyttämiseen tarvittavat attribuutit omiin luokkiinsa. Nämä luokat toimivat jokaisen komponentin mallinnuksen pohjana.

PDMS hierarkia rakentuu World -tason alle. World -tason alla ensimmäisenä on Site -taso, joka käytännössä tarkoittaa erillistä komponenttia tai muuta laitetta, joka on mallinnettu. Site -tason sisällä komponentti tai laite jaetaan sopiviin mallinnettaviin osiin eli Zone -tasoihin. Zone -tasojen alla on varsinaiset mallinnus elementit, jotka jakautuvat pipeihin, equipmentteihin ja structureihin. Pipe -taso pitää sisällään kaikki putkien mallintamiseen liittyvät osat. Equipment -tason alla on taas normaalit geometriset muodot, kuten lieriöt, suorakulmiot, kartiot ja puolipallot. Näiden avulla voidaan mallintaa esimerkiksi koteloita, kammioita, säiliöitä ja suuttimia. Structure -tason alle sisältyy kaikki teräsrakenteet esimerkiksi paneelit ja palkit.

Pääelementti luokkia ovat TubePDMS, HeaderPDMS ja PanelPDMS. TubePDMS -luokkaa käytetään pipe elementtien tekemiseen. Pipe elementti koostuu kolmesta tasosta PDMS hierarkiassa. Pipe -tasosta, joka on ylin taso, Bran -tasosta ja alimmasta Bend -tasosta. Pipe pitää sisällään kasan Branch nimisiä elementtejä, jotka voidaan mieltää yksittäisiksi putkiksi, joilla on alku- ja loppupisteet. Jokaisella Branch -elementillä voi olla yksi tai useampia Bend -elementtejä, jotka mallintavat kyseiseen Branch -elementtiin mutkia halutuilla kulmilla ja suunnilla. Pipe -elementin luonti ohjelmassa on yksinkertaista, se tarvitsee vain Zone -elementin minkä alle se luodaan, nimen ja specificationin.

```
zone = FindElement(Aveva.Pdms.Database.DbType.Design, Name);
pipe = zone.CreateLast(DbElementType.GetElementType("PIPE"));
pipe.SetAttribute(DbAttributeInstance.PSPE,GetElement(specification));
pipe.SetAttribute(DbAttributeInstance.PURP, "PIPE");
```

Yllä olevalla ohjelmakoodin pätkällä luodaan siis uusi Pipe -elementti halutun Zonen alle. Branch -elementin luominen on monimutkaisempaa, mutta se on ohjelmassa luotu niin että sen käyttö on yksinkertaista.

```
bran = pipe.Create(branCounter, DbElementType.GetElementType("BRANCH"));
bran.SetAttribute(DbAttributeInstance.NAME,pipeName);
bran.SetAttribute(DbAttributeInstance.PSPE, DbElment.GetElement(Spec));
bran.SetAttribute(DbAttributeInstance.HSTU, DbElment.GetElement(Spec));
bran.SetAttribute(DbAttributeInstance.HPOS, Position (Xpos, Ypos, Zpos));
bran.SetAttribute(DbAttributeInstance.HDIR, Direction.Create(branHeadDir));
bran.SetAttribute(DbAttributeInstance.HBOR,Units.Bore.CreateBore().
bran.SetAttribute(DbAttributeInstance.LHEA, true);
bran.SetAttribute(DbAttributeInstance.HCON, "OPEN");
bran.SetAttribute(DbAttributeInstance.TPOS, , Position (Xpos, Ypos, Zpos));
bran.SetAttribute(DbAttributeInstance.TDIR, Direction.(direction));
bran.SetAttribute(DbAttributeInstance.TBOR, Units.Bore.CreateBore()
bran.SetAttribute(DbAttributeInstance.LTAI, true);
bran.SetAttribute(DbAttributeInstance.TCON, "OPEN");
```

Branch -elementti tarvitsee paljon enemmän attribuutteja, jotta se mallintuu PDMS:ään. Muutamia mainittavia ovat nimi, alku- ja loppupisteet, koko ja specificationit. Bend -elementit tarvitsevat melkein yhtä paljon attribuutteja kuin Branch -elementit.

Equipment -tason alla on vielä SubEquipment -taso, joka vastaa Equipment -tasoa, mutta yksi taso syvemmällä hierarkiassa. Equipment -tason alla käytettiin esimerkiksi Dish-, Cone-, Box- ja Cylinder -elementtejä, jotka vastaavat mallissa nimiä kuvaavia geometrisiä muotoja. Näiden elementtien luominen ohjelmakoodissa oli yksinkertaisempaa kuin putkien. Esimerkiksi Cylinder -elementti luotiin seuraavanlaisella ohjelmakoodilla:

```
cyl = equi.Create(position, ElementType("CYLINDER"));
cyl.SetAttribute(DbAttributeInstance.POS,.Position(xpos, ypos, zpos));
cyl.SetAttribute(DbAttributeInstance.HEIG, height)
cyl.SetAttribute(DbAttributeInstance.DIAM, diameter);
cyl.SetAttribute(DbAttributeInstance.ORI, Orientation(orientation));
```

Cylinder -elementti tarvitsee vain paikan, korkeuden, paksuuden ja suunnan. Muut Equipment -tason alla olevat elementit rakentuvat ohjelmakoodissa vastaavalla tavalla.

Structure -tasoa käytettiin ohjelmassa vain Panel -elementtien tekoon, jotka muodostuivat hieman eri tavalla muihin verrattuna. Panel -elementin mallintamiseen tarvittiin PlotOutline -elementtiä ja Pavement -elementtejä. PlotOutline -elementille annettiin vain paneelin yksi nurkka kohta, josta sitä lähdetään mallintamaan ja paksuus. Tämän elementin alle luotiin tarvittava määrä Pavement -elementtejä, joita käytettiin kulma kohtien määrittämiseen. Esimerkiksi suorakulmion muotoisen paneelin mallintamiseen tarvittiin neljä Pavement -elementtiä.

Jokaiselle elementille ohjelmassa tehtiin omat funktiot ja ne jaettiin sopiviin luokkiin niiden päätasojen mukaan. Näitä funktioita ja luokkia käyttämällä oli helppoa mallintaa ohjelman kautta PDMS:ään haluttuja komponentteja. Vaikein asia PDMS:n käyttämisessä ohjelmakoodin kautta oli orientaation oikein määrittäminen. Orientaatio muodostuu ilmansuunnista ja ylös ja alas suunnista ja näiden välisistä kulmista, joten sen hahmottaminen suoraan oikein oli hankalaa.

8 ONGELMAT

Ongelmia oli monenlaisia projektin edetessä, mutta kaikista päästiin kuitenkin ylitse tai sivulta kiertämällä ohitse. Projektin alussa ensimmäisenä ongelmana oli ohjelmakoodin sekavuus uutena käyttäjänä. Kesti muutama päivä päästä kunnolla sisälle miten ohjelmakoodi oli rakennettu ja kuinka se toimi. Tämä johtui siitä, että en ollut aikaisemmin tehnyt muita kuin pieniä yksinkertaisia ohjelmia, ja nyt edessä oli ensimmäinen isompi ohjelmointiprojekti. Suurin ongelma tässä ehkä oli ymmärtää, kuinka mitkäkin luokat keskustelivat ja riippuivat toisistaan. Kerralla koko ohjelmakoodin ymmärtäminen kunnolla olisi ollut liian työlästä ja hidasta, joten alussa tutkiskelin vain ohjelmakoodin osia joita käytin suoraan luodessa itse uutta ohjelmakoodia. Ohjelmakoodiin kunnolla sisälle päästyä vaikutti se suurimmalta osin helposti ymmärrettävältä.

Osittaisena ongelmana koko projektin aikana oli tiedon saanti mallinrakennusta varten, mikä johtui ymmärrettävästä syystä. Henkilöiltä, joilta tietoa piti saada, olivat osan ajasta kiinni kiireellisissä ja tärkeissä kattilaprojekteissa. Osittain myös itse syynä tähän, koska loin ohjelmakoodia innostuneena nopeammin kuin uutta tietoa kerkesi saada. Kiersin ongelman tosin useimmiten tekemällä ohjelmassa muuta kuin varsinaista omakohtaista komponenttia. Loin ohjelmaan uusia osia ja muokkasin ohjelmakoodia selkeästi ymmärrettävämmäksi.

Ohjelmakoodin luonnissa oli myös ongelmia. PDMS:n rajapinta .NET-ympäristöön oli uusi yrityksen käytössä, joten kaikki ohjelmakoodin osat eivät olleet täysin hallinnassa. Ongelmana siinä oli kuinka saada käskyt välitettyä oikein ohjelmasta PDMS:ään. AVEVA:n tuki kuitenkin antoi neuvoja tarpeen mukaan ja loput yritys-erehdys-tekniikan avulla itse. Ohjelmakoodin yhtenevien osien yhdistämisessä oli ongelmia pienten eroavaisuuksien vuoksi, joihin joutui kehittämään omat ratkaisut. Yleisin ongelma, joka tuli vastaan oli kuitenkin normaali virheidenkäsittely. Ohjelma koottiin dll -tiedostoon normaalin exe -tiedoston sijasta, jolloin Visual Studion omat virheidenkäsittely työkalut eivät paljoa auttaneet. Virheitä yritettiin ratkaista "`try-catch`" -funktioiden, viestilaatikoiden ja perinteisen yritys-erehdys -tekniikan avulla.

9 JOHTOPÄÄTÖKSET

Oman osuuteni projektissa sain työsuhteen aikana valmiiksi niiltä osin kuin oli alussa sovittu. Kerkesin myös lisäillä uusia tarvittavia toimintoja ohjelmaan ja korjailla ja kehittää ohjelmaa paremmaksi. Alustavat testauksetkin ehdin ajamaan komponentille, jolla näki että ohjelma toimii kuten oli tarkoitus.

Omasta mielestäni projekti onnistui hyvin. Alussa työ oli haasteellisempaa, mutta alun jälkeen työ oli pääosilta helpohkoa. Kuitenkin myös vaikeampia asioita tuli vastaan mikä piti työskentelyn mielenkiintoisena. Koulussa opetettuja asioita pystyi paljon käyttämään hyväksi ohjelmoinnin osalta, mitkä auttoivat pääsemään vauhtiin ohjelmakoodia luodessa.

Parantamisen varaa oikeastaan jäi vain ohjelmakoodin selkeyden ja yhdenmukaisuuden osalta. Kuitenkin seuraavat ohjelmoijat tulevat jatkamaan tekemäni ohjelman jatkokehitystä ja mahdollisesti parantavat puutteet.

Opin projektista paljon uutta työelämästä ja isossa projektissa olosta. Myös ensimmäistä isoa projektia ohjelmoidessa itsenäisesti oli uusi kokemus, mikä varmasti auttaa jatkossa paljonkin. Kaiken kaikkiaan jäi itselleni hyvin positiivinen kuva projektissa työskentelemisestä.

Kehitystyö jatkuu ohjelman osalta vielä monella komponentilla, mutta se tulee olemaan osa Foster Wheeler Energia Oy:n projekteja tulevaisuudessa.

LÄHTEET

1. Tekesin digitaalinen tuoteprosessi projekti [Viitattu 26.3.2011]
Saatavissa: http://www.tekes.fi/ohjelmat/DTP/Projektit?id=9663281

2. Energiaa työssä ja tuloksissa. Katsaus Foster Wheeler Energia OY Groupin toimintaan [Viitattu 26.3.2011]
Saatavissa: http://www.fosterwheeler.fi/Tiedostot/FW2008_Final_LOW.pdf

3. Foster Wheelerin historiaa [Viitattu 27.3.2011]
Saatavissa: http://www.fwc.com/about/history.cfm

4. Savonia esittely [Viitattu 27.3.2011]
Saatavissa: http://portal.savonia.fi/amk/esittely/

5. Pioneering CFB technology Kiertopetikattilateknologiasta [Viitattu 2.4.2011]
Saatavissa: http://www.fwc.com/publications/pdf/CFBBrochure.pdf

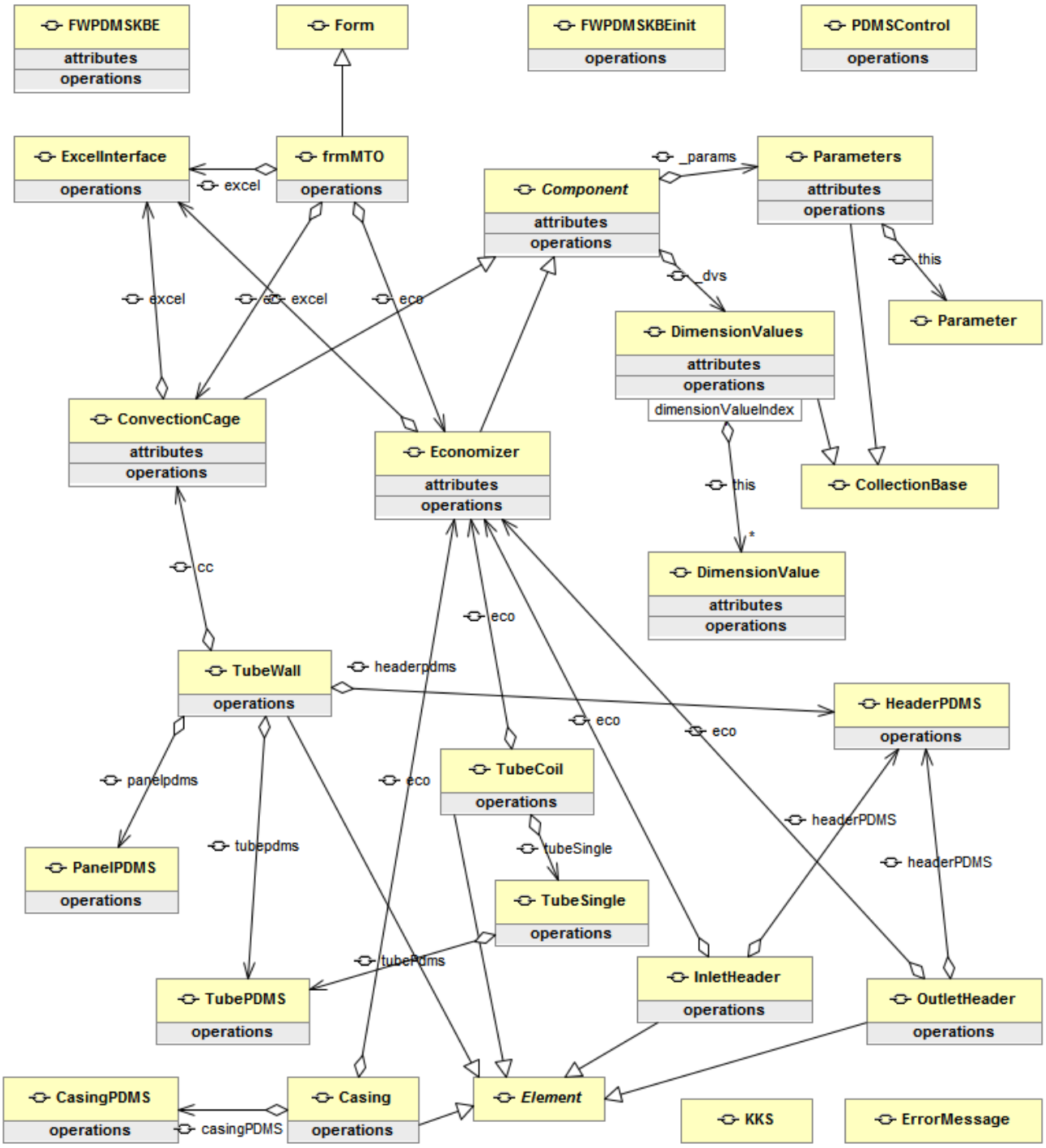
6. Aveva PDMS ohjelmisto [Viitattu 9.4.2011] Saatavissa:
http://www.aveva.com/products_services_aveva_plant_pdms.php

7. MVC arkkitehtuurista [Viitattu 10.4.2011] Saatavissa:
http://www.phpwact.org/pattern/model_view_controller

8. Rajapinnoista [Viitattu 10.4.2011] Saatavissa:
http://en.wikipedia.org/wiki/Api

9. Olio-ohjelmointi [Viitattu 13.4.2011] Saatavissa:
http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/4/Kasitteet.html

LIITE Täydellinen UML:n mukainen luokkarakennekuva ohjelmasta



www.savonia.fi