Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Pekka Hellsten

Toimituskohtaisen dokumenttikokoelman hallintatyökalu ACS800-tuoteperheelle

Insinöörityö 20.2.2010

Ohjaaja: projektipäällikkö Sami Mäki-Korte
Ohjaava opettaja: yliopettaja Hannu Laine

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Pekka Hellsten Toimituskohtaisen dokumenttikokoelman hallintatyökalu ACS800-tuoteperheelle 90 sivua 20.2.2010
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	projektipäällikkö Sami Mäki-Korte yliopettaja Hannu Laine

Insinöörityössä suunniteltiin ja toteutettiin ABB Oy:lle hallintatyökalu, jolla koostetaan ACS800-taajuusmuuttajien toimituskohtainen dokumenttikokoelma. Työkalun keskeisin tehtävä on kerätä tuotteiden manuaalit ja toimintaa täydentävät loppudokumentit yhteen kokoelmaan, joka voidaan lähettää asiakkaalle sähköisessä muodossa tai fyysisesti paperimapeissa. Tärkeä vaatimus oli parantaa käytössä olevaa dokumenttikokoelman muodostusprosessia automatisoimalla eri toimintoja. Manuaalien keräämisessä tuli hyödyntää ulkoisen ACS800 Product Configurator -tuotekonfiguraattorin projektitietoa.

Ohjelma toteutettiin C++-ohjelmointikielellä käyttäen Qt-ohjelmistokehyksen Open Source Edition 4.5 -laitosta. Windows API -ohjelmointirajapintaa ja Visual Basic for Applications -ohjelmointikieltä käytettiin paperimappien päätylehtien ja sisällysluetteloiden luomisessa. VBA:lla toteutettiin myös manuaalien keräämiseen tarvittavan manuaalisääntötiedoston XML-muunnos. jQuery-kirjastoa hyödynnettiin dokumenttikokoelmasta muodostettavan HTML-indeksisivun teossa.

Työn lähtötavoitteena oli korvata käytössä oleva CDmacro-työkalu, joka kerää pelkästään tuotteiden manuaalit. Tähän tavoitteeseen ei vielä päästy. Työ laajeni koko dokumenttikokoelman hallintatyökaluksi ja keskittyi vain ACS800 Product Configuratorin kattamiin tuotteisiin. Uudella työkalulla voi luoda dokumenttikokoelmia kaikille multidrive-tuotteille ja nestejäähdytteisille single drive -tuotteille. CDmacroa käytetään vielä ilmajäähdytteisten single drive -tuotteiden ja tiettyjen tuotekokonaisuuksien manuaalien keräämiseen. Hallintatyökalussa nämä ominaisuudet rajattiin jatkokehitykseen, jotta työssä voitiin kunnolla keskittyä multidrive-tuotteisiin.

Insinöörityön tuloksena syntyi ACS800 MPCompiler -nimen saanut tuotedokumentaation keräävä itsenäinen työpöytäsovellus. Ohjelma otettiin käyttöön sovellussuunnitteluosastolla. Ohjelmalle toteutettiin käyttöliittymä, jonka avulla dokumenttikokoelman muodostusprosessia voi seurata hallitusti ja yksityiskohtaisesti. Dokumenttikokoelman rakennetta voi muokata useilla toiminnoilla ja asetuksilla vastaamaan lopullista käyttötarkoitusta. Uuden työkalun ansiosta dokumenttikokoelmien muodostaminen multidrive-tuotteille tulee olemaan nopeampaa ja helpompaa.

Hakusanat	dokumenttien hallinta, ohjelma, Qt, C++, käyttöliittymä

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Pekka Hellsten
Title	Delivery specific document collection management tool for the ACS800 product family
Number of Pages	90 pages
Date	20 February 2010
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor	Sami Mäki-Korte, Project Manager
Supervisor	Hannu Laine, Principal Lecturer

The purpose of this thesis was to design and implement a management tool for ABB, which composes a delivery specific document collection for ACS800 frequency converters. The application's main task is to collect the products' manuals and supplementary documents. This collection can afterwards be delivered to the customer in either electrical form or physically in paper binders. A key requirement was to improve the existing document collection process by automating various operations. The project data of an external ACS800 Product Configurator system had to be utilized for the manual collecting process.

The application was implemented in C++ under Qt framework's 4.5 Open Source Edition license. Windows API and Visual Basic for Applications were used in the document collection process to form the paper binders' covers, spines and table of contents'. VBA was also used to create an XML conversion of a particular manual status list file. The animations and navigation of the document collection to be featured on the HTML page was implemented using the jQuery library.

The initial goal of the thesis was to replace the existing CDmacro tool, which only collects the products' manuals. This goal was not yet accomplished. The thesis expanded into an entire document collection management tool and focused only on products covered by the ACS800 Product Configurator. The management tool can compile document collections for all multidrive products and for liquid-cooled single drive products. CDmacro is still used to collect manuals for air-cooled single drives and for a few certain smaller product families. These features were post-poned for further development so the new management tool could properly focus on multidrive products.

This thesis resulted in an independent desktop application and product documentation collector named ACS800 MPCompiler. The program was deployed to the application engineering department, where it stays in active use. The program has a graphical user interface, which allows the user to monitor the document collection process under versatile control and in detail. The structure of the document collection can be modified with several functions and settings to accommodate the intended use. With the new management tool, the document collection process for multidrive products will be faster and easier.

Keywords	document management, program, Qt, C++, user interface

Sisällys

Lyhenne- ja käsiteluettelo

.NET — .NET Framework; Microsoftin kehittämä useita ohjelmointikieliä tukeva *ohjelmistokehys*, joka on saatavilla Windows-käyttöjärjestelmille.

ACS800 Product Configurator — ABB Oy:n työkalu ACS800-tuotteiden konfigurointiin. Käytetään myös projektin aikaista nimitystä Juice.

ActiveX — Uudempi nimitys OLE (Object Linking and Embedding) -tekniikasta, jonka avulla sovellus voidaan sulauttaa ja yhdistää Microsoftin tukemiin dokumentteihin ja muihin ohjelmistokomponentteihin.

CDmacro — ABB Oy:n nykyinen työkalu tuotteiden manuaalikokoelmien muodostamiseen.

COM — Component Object Model; Microsoftin kehittämä alustariippumaton rajapinta, joka määrittelee ohjelmistokomponenttien keskinäisen kommunikaation. Käytännössä sisältää useita osa-alueita, joihin lukeutuu muun muassa *ActiveX*-tekniikka.

CSS — Cascading Style Sheets; erityisesti WWW-dokumenttien tyyliohjeiden kirjoittamiseen käytetty kieli, jolla ilmaistaan dokumentin ulkoasu.

CSV — Comma-separated Values; tiedostoformaatti, joka taulukoi varastoidun tiedon pilkuin ja rivinvaihdoin eroteltuna.

Vianjäljitysohjelma — Debugger; tietokone-ohjelma, jota käytetään testaamaan muita ohjelmia.

Valintaikkuna — Dialog; ohjelmistoteknologiassa käytetty termi kuvaamaan käyttöliittymän väliaikaista ikkunaa, jonka tarkoituksena on esittää käyttäjälle valintoja tai tietoa ja vastaanottaa saadut syötteet takaisin prosessoitavaksi.

GPL — GNU General Public License; vapaa ohjelmistolisenssi, joka antaa saajalle oikeudet muokata ja jakaa uudelleen lisensoitua ohjelmaa. GPL edellyttää muunnellun ohjelman noudattavan samoja vapauksia.

HTML — HyperText Markup Language; kuvauskieli verkkosivujen tekoon.

JavaScript — Dynaamiset toiminnallisuudet mahdollistava pääasiassa verkossa käytettävä oliopohjainen komentosarjakieli.

LGPL — GNU Lesser General Public License; *GPL*-lisenssiä sallivampi ohjelmistolisenssi. LGPL-ohjelmisto voidaan linkittää yhteen ei-GPL-lisensoidun ohjelmiston kanssa, jolloin lähdekoodia ei tarvitse julkaista.

MFC Microsoft Foundation Classes; Microsoftin Windows API -ohjelmointirajapinnan päälle rakennettu C++-luokkakirjasto.

Meta-Object System

Qt:n perusmekanismi ja samalla laajennus C++-kieleen, joka mahdollistaa itsenäisten ohjelmistokomponenttien luomisen ja yhdistämisen.

ODBC Open Database Connectivity; alunperin Microsoftin kehittämä rajapinta tietokannoissa sijaitsevaan dataan yhdistämisessä.

Ohjelmistokehys

Application Framework; tietokoneohjelmien toteuttamiseen käytetty ohjelmisto, joka sisältää ohjelmointia helpottavia apuvälineitä, kuten omia luokkakirjastoja.

Plussakoodi Universal Type Code; plus code; ABB Oy:n tunnuskäytäntö tuoteominaisuuksien erotteluun ja hallintaan.

Qt Monialustainen ohjelmistokehys, joka on kehitetty graafisten käyttöliittymäsovellusten luomiseen.

SAP ERP Saksalaisen SAP AG -yrityksen kehittämä toiminnanohjausjärjestelmä, jolla voidaan hallita tuotantoa, jakelua, varastonhallintaa, laskutusta ja kirjanpitoa.

SAX Simple API for XML; sarjamuotoinen *XML*-tiedon käsittelyrajapinta.

SQL Structured Query Language; relaatiotietokantojen hallinnoinnissa käytetty standardoitu kyselykieli.

UML Unified Modeling Language; standardoitu graafinen mallinnuskieli.

Käyttöliittymäelementti

Widget; ohjelmistoteknologiassa käytetty termi kuvaamaan käyttöliittymän visuaalista elementtiä.

WTL Windows Template Library; ilmainen oliopohjainen C++-kirjasto Windows-ohjelmointiin.

wxWidgets Monialustainen graafisten käyttöliittymien luomiseen erikoistunut C++-luokkakirjasto.

XML eXtensible Markup Language; alustariippumaton tekstimuotoinen merkintäkieli, joka tarjoaa rakenteellisen esitystavan tiedon yhdistämiseen, vaihtamiseen ja julkaisemiseen.

1 Johdanto

Insinöörityön tavoitteena oli suunnitella ja toteuttaa ABB Oy:lle ohjelma, jolla ACS800-tuotteille voidaan laatia toimituskohtainen tuotedokumentaation sisältävä kokoelma. ABB on monikansallinen yli sadassa maassa toimiva teollisuuskonserni, jonka toimialaan kuuluvat automaatiotekniikka ja sähkövoimatekniikka. Yhtiö on perustettu vuonna 1988 ruotsalaisen ASEAn ja sveitsiläisen BBC Brown Boverin yhdistyessä, mutta juuret ulottuvat myös Suomeen asti, kun suomalainen Oy Strömberg Ab siirtyi ASEAn omistukseen vuotta aiemmin 1987 [1; 2]. Insinöörityö tehtiin Drives-yksikölle, joka on erikoistunut taajuusmuuttajien ja tasavirtakäyttöjen kehittämiseen ja valmistukseen teollisuusalan käyttötarpeisiin.

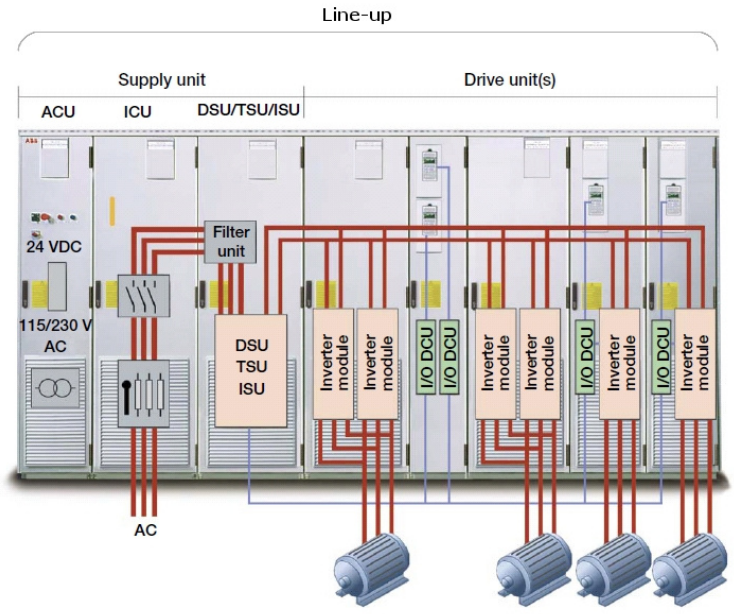
Ohjelman on tarkoitus korvata käytössä oleva Microsoft Excel -pohjainen *CDmacro*-työkalu, jolla dokumenttikokoelmaan kuuluvat manuaalit kerätään tuotevalintojen perusteella. Työntekijän kannalta manuaalien valitseminen tuotekohtaisesti on hidas ja virhealtis tehtävä. Lähtökohdaksi työlle otettiin olemassa olevan *ACS800 Product Configurator* -tuotekonfiguraattorin projektitiedon käyttäminen, jolla manuaalien valintaprosessi saadaan automatisoitua. Lisäksi uudella työkalulla haluttiin nopeuttaa ja helpottaa lopullisen dokumenttikokoelman muodostusprosessia. Henkilökohtaisena tavoitteenani oli kehittää yleisiä ohjelmointitaitojani. Minulla ei aikaisemmin ollut kokemusta laajamittaisen työpöytäsovelluksen kehittämisestä ja halusin insinöörityölläni korjata tämän puutteen. Hallintatyökalun kehittäminen antoi myös oivan tilaisuuden olla mukana ohjelmiston koko kehityskaaren toteuttamisessa.

Kirjallinen raportti esittelee insinöörityön tuloksena saadun ACS800 MPCompiler -sovelluksen ja sen kehitysprosessin vaiheet. Raportissa keskitytään työn ratkaisuihin, mutta myös alkuperäiset suunnitelmat ja syyt muutoksiin mainitaan. ACS800-taajuusmuuttajien tuoteperhe ja dokumenttikokoelman yleisperiaate esitellään ohjelman toiminnallisuuden hahmottamiseksi. Toteutettujen toimintojen lisäksi on tarpeen käsitellä myös erilliset jatkokehityssuunnitelmat, jotka rajattiin pois työn julkaisuversiosta.

2 ACS800-tuoteperhe

2.1 Yksi- ja monikäyttöiset taajuusmuuttajat

ACS800-tuotteet ovat sähkökäyttöisiä taajuusmuuttajia, joiden tehot vaihtelevat välillä 0.55–5600 kW. Tuoteperhe on laajin ABB:n Drives-yksikön taajuusmuuttajatuoteperheistä. Käyttökohteiden pienintä päätä edustavat kutomakoneet, ja tehokkaimpia tuotteita voi löytää paperitehtaista ohjaamassa paperimassan pumppaamiseen käytettyjä moottoreita. ACS800-taajuusmuuttajat jakautuvat single drive- ja multidrive-tuoteperheisiin käyttötarkoituksien ja ominaisuuksien mukaan. Single drive -tuotteet ohjaavat yhtä moottoria, jota pyöritetään omalla muista mahdollisista moottoreista poikkeavalla nopeudella. Näitä käytetään esimerkiksi ilmastointilaitteissa, ydinvoimalan jäähdytyspumpuissa tai laivan potkureissa. Multidrive-taajuusmuuttajalla ohjataan useita moottoreita samanaikaisesti ja tuote suunnitellaan aina tapauskohtaisesti asiakkaan toiveiden mukaisesti. Käyttökohteena voi olla esimerkiksi yksittäinen paperikone, jossa kaikkien moottoreiden on pyörittävä samalla nopeudella linjan toimivuuden takaamiseksi. Sekä single drive- että multidrive-tuotteita valmistetaan ilma- ja nestejäähdytysratkaisuilla. Multidrive-tuote ja ABB:n taajuusmuuttajatuoteperheisiin liittyviä yleisiä termejä nähdään kuvassa 1. [3, 4.]



Kuva 1. Esimerkki yhden linjakokoonpanon ACS800 multidrive -taajuusmuuttajasta [4, s. 6].

Single drive- ja multidrive-taajuusmuuttajat koostuvat moduuleista, jotka vastaavat kaikesta toiminnallisuudesta. Moduulit kuuluvat aina niiden toimintoja ohjaavien ja keskinäisestä kommunikoinnista huolehtivien laajempien ryhmien sisälle. Tuotteisiin voi kuulua useita ryhmiä, kuten tasajännitettä järjestelmään syöttävä syöttöryhmä (supply unit) tai käyttöryhmä (drive unit), joka huolehtii tasajännitteen muuntamisesta vaihtojännitteeksi ja moottorin ohjauksesta. Muita ryhmiä ovat muun muassa ylemmän tason ohjausryhmä (control unit), dynaaminen jarruryhmä (brake unit) ja nestejäähdytteisten tuotteiden jäähdytysryhmä (cooling unit). Multidrive-tuotteissa ryhmistä koostetaan linjakokoonpano (line-up) yleensä tehtaan yhtä tuotantolinjaa tai sen osaa kohti. Linjakokoonpano koostuu usein vähintään yhdestä syöttöryhmästä ja jokaista ohjattavaa moottoria kohden yhdestä käyttöryhmästä. Lopullisessa kokoonpanossa multidrive-tuotteet ovat aina kaapitettuja, mutta single drive -tuotteet voivat olla myös lattialle sijoitettuja tai seinään asennettuja. Suurimmat kaapitetut single drive -tuotteet sisältävät vain yhden käyttö- ja syöttöryhmän sekä vaihtoehtoisen määrän muita ryhmiä.

Multidrive- ja single drive -tuotteiden toiminnallisuuksiin vaikutetaan standardiominaisuuksien lisäksi optioilla. Ne ovat tuotteesta riippuvaisia vaihtoehtoisia lisäominaisuuksia, kuten multidrive-taajuusmuuttajilla linjakokoonpanon väri tai käyttöryhmän parametrit tulostava käytönohjaispaneeli. Optioita on useita satoja ja kaikkien mahdollisten yhdistelmien konfigurointi käsin on mahdotonta, joten tätä varten ABB:llä on kehitetty ominaisuuksien *plussakoodeja* hyödyntävä ACS800 Product Configurator -tuotekonfiguraattori.

2.2 Universal Type Code

ABB:llä on käytössä Universal Type Code – tai tunnetummin plussakoodiksi – kutsuttu tunnuskäytäntö, jossa tärkeimmille tuotteen ominaisuuksille on määritelty lyhyet tunnukset. Esimerkiksi F266 kuvaa käyttöryhmän DC-katkaisinta. Plussakoodeja käytetään helpottamaan tuotekokonaisuuksien hallintaa, sillä ominaisuudet voidaan esittää yksilöllisillä tunnuksilla. Yhdistelemällä plussakoodeja saadaan tuotteen tekniset tiedot ilmaistua yhdellä merkkijonolla. Tällaisesta yhdistelmästä käytetään samaa plussakoo-

di-nimitystä, mutta käytännössä se jakautuu tuotteen tyyppikoodiin, joka sisältää tuoteperheen, tuotetunnuksen, tehon ja jännitteen sekä ominaisuuksien plussakoodeihin.

	Ordering Code	107 inverters	207 (ISU) regenerative supply unit	307 and 507 (6 p & 12 p DSU supply units)	407 & 807 (6 p & 12 p TSU supply units)	107LC (inverters)	207LC (ISU)	307LC - 1207LC (6p - 24 p DSU supply units)	607 / 607LC (3-phase brake units)
		Frame sizes R2i - 12xR8i	Frame sizes R7i - 12xR8i	Frame sizes D3 - 5xD4	Frame sizes B4-B5	Frame sizes R2i-10xR8i	Frame sizes R8i-10xR8i	Frame sizes D3 - 3xD4	Frame sizes R7i - 5xR8i
Braking (see braking unit table)									
Incoming unit apparatus									
Disconnector and contactor for single supply units	F253 F250	-	• 10)	• 10)	-	-	• 11)	• 11)	-
Air circuit breaker	F255	-	• 12)	• 12)	•	-	• 11)	• 11)	-
Drive units									
DC switch	F266	•	-	-	-	•	-	-	-
Safety options		-							
Prevention of unexpected start-up	Q950	□	-	-	-	□	-	-	-
Earth fault monitoring, earthed network	Q953	•	•	•	□	•	•	•	•
Earth fault monitoring, unearthed mains	Q954	-	□	□	-	-	□	□	-

• Standard
□ Option with ordering code
- Not available

Syöttöryhmä 407

Kuva 2. Katkelma ACS800 Multidrives -katalogin listaamista tuoteominaisuuksista [4, s. 45].

Tarkastellaan syöttöryhmää 407, jonka ominaisuuksia on listattu kuvassa 2. Tuotteen standardiominaisuuksiin kuuluu muun muassa ilmakatkaisija (F255) ja vaihtoehtoisiin optioihin maasulun valvonta (Q953). Yksinkertainen plussakoodi tässä tapauksessa voisi olla ACS800-407-0680-5+F255+Q953, jossa merkinnät 0680 ja 5 vastaavat tehoa 680 W ja jännitettä 500 V.

2.3 ACS800 Product Configurator

ACS800 Product Configurator on ACS800-tuoteperheelle räätälöity verkkopohjainen asiakastilaukseen kuuluvien tuotteiden konfigurointityökalu. Se on kehitetty pääasiassa multidrive-tuotteille, mutta sillä voidaan myös suunnitella nestejäähdytteisiä single drive -tuotteita. Työkalulla luodut projektit listataan Microsoft Access -tietokantaan ja yksityiskohtaisempi tieto tallennetaan XML-dokumentteihin. Toimituskohteesta riippuen yksi projekti voi sisältää monta linjakokoonpanoa, joiden rakenteet, optiot ja ryhmien tekniset tiedot ovat kaikki määriteltävissä. Linjakokoonpanon ja ryhmien ominaisuuksien hallintaan käytetään plussakoodeja. Tuotekonfiguraattoria varten on jouduttu mää-

rittelemään myös sisäisiä tunnuksia ominaisuuksille, joilta plussakoodit puuttuvat. Liitteessä 1 on esitetty kuvakaappaus työkalun linjakokoonpanonäkymästä.

3 Dokumenttikokoelma

3.1 Tuotedokumentaation kattava manuaalipaketti

Tuotteiden manuaalit lähetetään asiakkaille kokoelmissa yhdessä muiden loppudokumenttien kanssa. Manuaalit kuvaavat myytävien tuotteiden ja niihin sisältyvien moduulien asennuksen, käytön ja ominaisuudet. Vaikka tietyt manuaalit kuvaavat esiasennettuja tuotteita (moduulit), huoltotöiden takia kaikkien manuaalien toimittaminen asiakkaalle on tärkeää. Manuaalien laajan lukumäärän hallitsemiseksi dokumenteille on määritelty yksikäsitteinen manuaalikoodiksi kutsuttu tunnus. Loppudokumentit ovat tuotedokumentaatiota täydentäviä dokumentteja, joilla tarkennetaan asiakasprojektikohtaisesti tilattujen tuotteiden tai niiden osien toimintaa ja ominaisuuksia. Näitä ovat muun muassa tuotteen mittakuvat, osaluettelot, piirikaaviot, riviliitintaulukot, moduulien vakiopiirikaaviot ja testausraportit.

Tietyissä tapauksissa asiakas haluaa tuotedokumentaation paperilla tai dokumentaatio halutaan säilyttää fyysisesti myöhempää käyttöä varten. Käytännön mukaisesti manuaalit ja loppudokumentit tulostetaan kahteen eri paperimappiin. Työntekijän vastuulle jää sisällön keräämisen ohella molempien mappien päätylehtien ja loppudokumenttimapin sisällysluettelon kirjoittaminen asiakkaan ja tuotteen tiedoilla. Useimmiten tällainen kahden mapin kokoelma luodaan jokaista projektin linjakokoonpanoa kohden, jolloin sisällysluettelo laaditaan linjakokoonpanon ja ryhmien loppudokumenteille. On kuitenkin joitakin poikkeustapauksia, joissa asiakas haluaa koko projektin kattavan mapin. Molempien paperimappityyppien kansilehdille, päätylehdille ja yhteiselle sisällysluettelolle on olemassa valmiiksi muotoillut ja vakioarvoilla täytetyt Microsoft Word -mallidokumentit.

3.2 Nykyinen muodostusprosessi

Toistaiseksi manuaalipaketin muodostamiseen on käytetty Microsoft Excel -makroa nimeltä CDmacro. Makro sisältää yksinkertaisen käyttöliittymän, jonka avulla käyttäjä valitsee tuotteet, joiden dokumentaatio on tarkoitus koota. Työkalu kerää tuotteita koskevat *PDF*-muotoiset manuaalidokumentit ja kopioi ne uuteen hakemistoon. Hakemistorakenteeseen luodaan myös valmiit hakemistot myöhemmin lisättäville loppudokumenteille. Lisäksi CDmacro muodostaa kopioiduille manuaaleille *HTML*-pohjaisen indeksisivun, josta dokumentteja pääsee tarkastelemaan. Toimitusta varten dokumenttikokoelma voidaan jälkeenpäin joko polttaa CD-levylle tai pakata zip-tiedostoksi.

Manuaalidokumenttien keräämiseen makro käyttää erillistä manuaalisääntötiedostoa. Manuaalisääntötiedosto on ABB:n dokumentointiosaston ylläpitämä Microsoft Excel -dokumentti, jossa on listattu kaikki ABB:n Drives -yksikössä käytössä olevat manuaalit eri taajuusmuuttajien tuoteperheille. Manuaaleista kerrotaan muun muassa nimet, tiedostopolut, manuaalikoodit, kielet ja kategoriat. Kategorioilla erotellaan, onko kyseessä esimerkiksi käyttöryhmämanuaali vai ohjelmistomanuaali. Tiedosto listaa myös kaikki manuaaleihin liittyvät ominaisuudet plussakoodein varusteltuna omissa sarakkeissaan. Esimerkiksi jos ominaisuus *RMBA-01 Modbus Adapter* (K458) on kuvattu manuaalissa, kyseisen rivin saraketta vastaavaan soluun on merkitty ruksi. CDmacron kannalta tämä on olennaista, koska tuotteiden manuaalit kerätään vertaamalla manuaalien ja valittujen tuotteiden plussakoodeja keskenään. Kun makro on kopioinut kerätyt manuaalit ja luonut valmiin hakemistorakenteen, käyttäjä etsii asiakasprojektia koskevat loppudokumentit verkkolevyiltä ja lisää ne dokumenttikokoelmaan.

CDmacrolla voi kerätä manuaaleja sovellussuunniteltaville multidrive- ja single drive -tuotteille. Hintalistojen mukaan myytäville single drive -standardituotteille manuaalit kerätään erillisten ohjeiden mukaan. Sovellussuunniteltavien tuotteiden dokumentaatio laaditaan ja kootaan sovellussuunnitteluosastolla, mutta standardituotteiden dokumentaatio kerätään tehtaassa ja toimitetaan usein fyysisesti tuotteen mukana. CDmacron piiriin kuuluvat myös erikoismanuaalikokoelmat, jotka ovat yksittäiselle tuotekokonaisuudelle tai -perheelle määriteltyjä valmiita manuaalipaketteja. Nämä moduuleille,

ACS850-04-taajuusmuuttajalle ja ACSM1- sekä ACQ810-tuoteperheille laaditut kokoelmat sisältävät vain single drive -tuotteisiin liittyviä manuaaleja.

4 Hallintatyökalun vaatimukset

4.1 Olemassa olevien resurssien hyödyntäminen

Tärkeä lähtökohta uuden työkalun kehittämiselle oli parantaa dokumenttikokoelman muodostusprosessia hyödyntämällä ABB:n sisäverkossa olevia resursseja. Käyttämällä ACS800 Product Configuratorin projektitietoa ensisijaisena syötteenä voidaan tuotteiden valitseminen hoitaa ohjelmallisesti, mikä nopeuttaa kokoelman valmistusta. Dokumenttipakettien luominen projektikohtaisesti ei aiheuttaisi rajoitteita, koska yksi projekti vastaa lähes aina yhtä asiakastilausta. Ohjelma käyttäisi seuraavia ABB:n sisäverkosta löytyviä resursseja syötteinään:

- ACS800 Product Configuratorin projektitietokanta
- projektien XML-dokumentit
- manuaalisääntötiedosto
- manuaalidokumentit
- paperimapin mallidokumentit.

CDmacron käyttämien syötteiden lisäksi uusi työkalu hyödyntäisi tuotekonfiguraattorin projektitietokantaa, projektien XML-dokumentteja ja paperimappien päätylehti- ja sisällysluettelomalleja. Projektikanta listaa kaikki tuotekonfiguraattorilla luodut projektit, mutta ei tallenna kuin projektia koskevat yleistiedot. Näitä ovat muun muassa projektin nimi, tunnus, kauppanumero ja tila. Yksityiskohtaisempi tieto projektista ja sen tuotteista tallennetaan XML-dokumenttiin. Dokumentissa on listattu kaikki konfiguroidut tiedot linjakokoonpanoista ja ryhmistä, kuten rakenteet ja ominaisuudet. Projektitiedon avulla saadaan manuaalit kerättyä ja kopioitua samalla menetelmällä kuin CDmacrossa, vertaamalla tuotteiden ja manuaalisääntötiedostossa listattujen manuaalien plussakoodeja keskenään. Projektitietoa voidaan myös hyödyntää paperimapin päätylehtien ja sisällysluetteloiden luomisessa. Dokumentit saadaan luotua ohjelmallisesti, kun pohjana käytetään

näiden dokumenttien valmiiksi muotoiltuja Microsoft Word -mallidokumentteja. Työn aikana ohjelmaan lisättiin asetukset, joilla manuaalien kopiointi ja paperimappien luonti voidaan kytkeä pois päältä. Dokumenttikokoelman muodostus jäi riippuvaiseksi projektikannasta, projektin XML-tiedostosta ja manuaalisääntötiedostosta.

4.2 Toiminnallisuus

Hallintatyökalun toiminnalliset vaatimukset määriteltiin ohjausryhmän kanssa. Ryhmään kuului minun lisäksi insinöörityön ohjaaja, dokumentointiosaston edustaja, sovellussuunnittelun edustaja sekä ohjelman tuleva käyttäjä sovellussuunnittelusta. Työkalusta toteutettaisiin itsenäinen työpöytäsovellus Windows-ympäristöön. Ilmajäähdytteisten single drive -taajuusmuuttajien ja erikoismanuaalikokoelmien käsittely rajattiin pois sovelluksen toiminnallisista vaatimuksista, jotta työn aikataulu ja laajuus pysyisivät kohtuullisena. Tuotekonfiguraattorilla voi luoda vain multidrive- ja nestejäähdytteisiä single drive -projekteja, joten edellä mainittujen toimintojen lisääminen olisi vaatinut liikaa resursseja. Ensimmäisen julkaisuversion haluttiin keskittyvän multidrive-projekteihin ja ylimääräiset osa-alueet siirrettiin jatkokehityssuunnitelmiksi.

Projektitiedon hyödyntäminen manuaalien keräyksessä

Uuden hallintatyökalun päävaatimus oli, että manuaalien keräämisessä hyödynnettäisiin ACS800 Product Configurator -tuotekonfiguraattorin projektitietoa. Tällä saadaan automatisoitua dokumenttikokoelman muodostusprosessia, koska manuaalien keräämiseen tarvittava tieto saadaan projektista.

Dokumenttikokoelman muokattavuus

Loppudokumenttien lisääminen haluttiin liittää osaksi uutta työkalua, joten tähän tehtävään määriteltiin esikatselutoiminto. Esikatselun tuli näyttää dokumenttikokoelman hakemistorakenne ja antaa käyttäjän muokata sitä haluamakseen. Hakemistorakenteeseen vaadittiin listaus kerätyistä projektimanuaaleista ja valmiiksi luodut hakemistot

loppudokumenteille. Rakenteen tuli noudattaa loogista hakemistojaottelua, jossa manuaalit ja loppudokumentit sijaitsevat erillisissä hakemistoissa.

HTML-indeksisivun luominen

Ohjelman tuli muodostaa HTML-muotoinen indeksisivu, jota asiakas voisi käyttää dokumenttikokoelman tuotedokumentaation selaamiseen. Sivulle vaadittiin tiedostolistaus kaikista dokumenteista, jotka piti pystyä tulostamaan yksittäin, hakemistokohtaisesti tai kaikki linjakokoonpanoon kuuluvat kerrallaan. Lisäksi kokoelman manuaaleista haluttiin erillinen listaus, josta selviäisi dokumentin nimi ja manuaalikoodi. Sivun ulkoasun haluttiin muistuttavan paperimappeja, eli käytännössä dokumentit tulisi olla eroteltuina ryhmittäin linjakokoonpanokohtaisesti.

Mappidokumenttien luominen

CDmacron painopiste on sähköisessä julkaisussa, sillä paperimappien koostaminen jää täysin käyttäjän vastuulle. Uudella työkalulla halutaan huomioida myös fyysinen julkaisu, joten asiakkaalle lähetettävien paperimappien laatimista täytyi helpottaa. Nykyisin kaikki mapin välilehdet muokataan käsin, mutta siirryttäessä uuteen työkaluun tulee ohjelman muokata annetuista päätylehti- ja sisällysluettelomalleista linjakokoonpanokohtaiset. Dokumenttien tulee sisältää tiedot asiakkaasta, projektista, kauppanumerosta ja ryhmistä. Paperimappien kansilehtimalleja ei tarvitse muokata, koska ne ovat vakioita sekä manuaali- että loppudokumenttimapeissa.

Hylätty suunnitteluratkaisu

Ohjelman alkuperäisenä vaatimuksena oli luoda kaksi vaihtoehtoista manuaalikokonaisuutta toimitustavasta riippuen. Paperimappiin tulostettavan kokoelman erityissisältöön kuuluivat muokatut paperimapit, ja vastaavasti asiakkaalle lähetettävään CD- tai DVD-levylle poltettuun kokoelmaan kuului HTML-indeksisivu. Tästä kahden kokoelmatyypin erottelusta kuitenkin luovuttiin tarpeettomana, ja molempien ominaisuudet yhdistettiin yhteen kokonaisuuteen.

4.3 Graafinen käyttöliittymä

Sovellukselle tuli kehittää graafinen käyttöliittymä, joka on looginen ja helposti omaksuttava. Dokumenttikokoelman muodostaminen hahmoteltiin suhteellisen suoraviivaiseksi prosessiksi: käyttäjä valitsee projektin, esikatselee dokumenttikokoelman, lisää samalla halutut loppudokumentit ja generoi kokoelman tietokoneelle. Nämä vaiheet tuli ottaa huomioon käyttöliittymän suunnittelussa. Projektit tulee esittää listauksessa, josta yksittäistä projektia voi hakea tunnuksella, nimellä tai kauppanumerolla. Esikatselun on näytettävä dokumenttikokoelman hakemistorakenne lopullisessa muodossaan ja antaa käyttäjän lisätä, poistaa ja järjestellä dokumentteja. Järjestelytoiminnon vaikutuksien tulee ulottua HTML-indeksisivulla näytettävään tiedostolistaukseen. Lisäksi käyttöliittymään tarvittiin asetussivu, jossa asetusten lisäksi pystyi määrittelemään ohjelman käyttämät ABB:n sisäverkon resurssit. Kehitysprosessin aikana resurssienmääritystoiminto kuitenkin poistettiin käyttöliittymän ominaisuuksista, kun sen todettiin olevan liian altis inhimillisille virheille. Tilalle toteutettiin resurssilistauksen esittävä näkymä.

5 Suunnittelu

5.1 Toteutusalustan valinta

Tärkeänä kriteerinä ohjelman toteutusalustalle pidin tukea C++-ohjelmointikielelle. Muita ohjelmointikielivaihtoehtoja olivat yrityksellä käytössä olevat C# ja Visual Basic *.NET*-alustalla, mutta aikaisempi kokemukseni ja mielenkiintoni C++:aan asettivat sen ensisijaiseksi valinnakseni. Tutustuin useaan ohjelmointikehykseen etsiessäni käyttötarkoituksiini sopivaa toteutusalustaa. Kehyksen oli oltava ilmainen, kattavasti tuettu ja riittävän monipuolinen käyttöliittymän omaavan sovelluksen kehittämiseen.

Microsoftin *MFC*-kirjasto olisi laajuutensa ansiosta kelvannut, mutta yrityksen siirrettyä painopistettä .NET-kehitykseen C++-ohjelmoijat ovat hiljalleen lähteneet etsimään muita ratkaisuja. Microsoftin vuonna 2004 vapaan lisenssin alaiseksi vapauttama *WTL*-kirjasto on aktiivisen käyttäjäkunnan suosittelema alusta, vaikkakin sen dokumentointia on moitittu. WTL ei kuitenkaan pärjännyt vertailussa *wxWidgets*-käyttöliittymäkirjaston

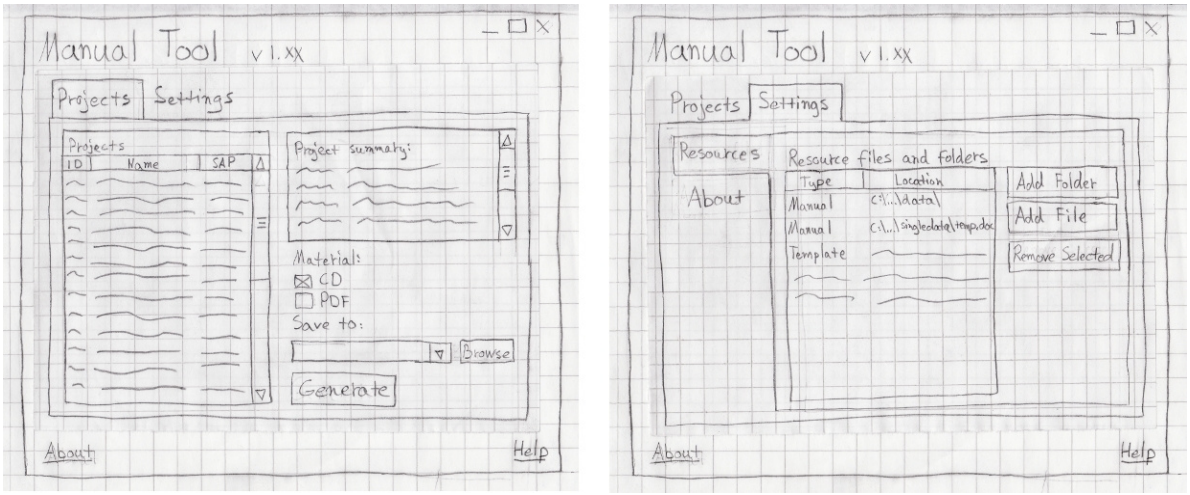
kanssa, josta tuli alkuperäinen valintani toteutusalustaksi. Kirjastossa kiehtoi sen monipuolisuus käyttöliittymien luomiseen, kattava dokumentaatio sekä ilmaiset räätölöidyt kehitysympäristöt ja visuaaliset suunnittelutyökalut [5].

Ehdin opiskella wxWidgets-kirjastoa muutaman päivän helmikuussa 2009, kunnes Nokia ilmoitti julkaisevansa aiemmin ostamansa norjalaisen TrollTech-yhtiön Qt-käyttöliittymäkirjastosta *LGPL*-lisenssin alaisen version maaliskuussa samana vuonna [6]. Toteutusalustaa valitessani olin tutustunut erittäin lupaavaan Qt:hen jo aikaisemmin, mutta ajattelin silloisen GPL-lisenssin rajoittavan liikaa ABB:n vaihtoehtoja insinöörityön suhteen. Vahvasti luokkapainotteinen, kaupallisen yrityksen tukema ja samat wxWidgets-kirjaston edut omaava Qt sai minut vaihtamaan toteutusalustaa vain muutama päivä alkuperäisen valinnan jälkeen.

5.2 Käyttöliittymä

Käyttöliittymää suunnitellessani päätin heti alussa, että sovellukseen ei tule valikkopalkkia. Halusin dokumenttikokoelman muodostamisen noudattavan johdonmukaista eri näkymiin jaettua prosessia yksinkertaisella ja selkeällä käyttöliittymällä. Toimintojen sijoittaminen niitä koskeviin näkymiin poistaa valikkopalkin tarpeen, jolloin käytettävyyttä saadaan parannettua yksinkertaisemmalla käyttöliittymällä. Kokoelman muodostus ja sovelluskohtaiset asetukset oli myös eroteltava toisistaan luomalla näille omat välilehdet. Suunnitteluvaiheessa välilehtiin kaavailtiin neljää näkymää. Projektivälilehti muodostuisi projekti- ja esikatselunäkymistä, ja asetusvälilehti jakautuisi asetus- ja resurssinäkymiin.

Lähdin hahmottelemaan käyttöliittymää kynällä ja paperilla. Piirsin pääikkunasta pelkistetyn pohjamallin ja useita luonnoksia projekti-, asetus- ja resurssinäkymistä. Sijoittelemalla näkymäpiirroksia pohjamallin päälle sain nopeasti tarkasteltua, miltä sovellus näyttäisi, ja tarkennettua käyttöliittymän asettelua. Asetusvälilehteen suunnitelluista alanäkymistä luovuttiin toteutuksen aikana, kun huomattiin yhden näkymän riittävän sekä resurssien että toiminnallisten asetuksien näyttämiseen. Kuvassa 3 on nähtävillä yhdet hahmotelmat projekti- ja resurssinäkymistä sekä suunniteltu välilehtijaottelu.



Kuva 3. Projekti- ja resurssinäkymän käyttöliittymähahmotelmat

Valmiista käyttöliittymähahmotelmista laadittiin Microsoft Paint -kuvankäsittelyohjelmalla ohjausryhmälle esiteltävät mallit. Luonnoksissa käytettiin pohjana kuvakaappauksia englantilaisesta rekisterin puhdistus- ja väliaikaistiedostojen poistotyökalusta nimeltä CCleaner, jotta hahmotelmien ulkoasua saatiin parannettua. Liitteeseen 2 on koottu projekti- ja resurssinäkymän graafiset käyttöliittymähahmotelmat sekä vaihtoehtoisia välilehtiluonnoksia.

5.3 Arkkitehtuuri

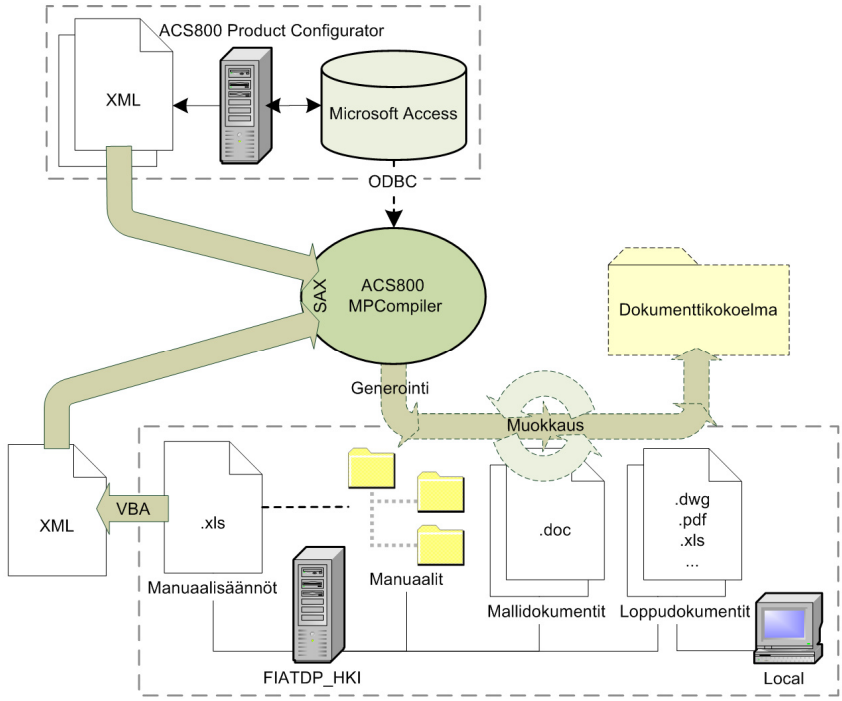
5.3.1 Ongelmat määrittelyssä

Suunnitteluvaiheessa tavoitteenani oli luoda ohjelman arkkitehtuurista tarkka määritelmä. Laadin vaatimusmäärittelyn pohjalta keskeneräiseksi jääneen toiminnallisen määrittelyn, jossa esitin ohjelman toimintalogiikan ja tärkeimpien toimintojen käyttöliittymät, kuvaukset, tarkoitukset ja käsittelyt. Määrittelyä tarkentavien *UML*-kaavioiden laatiminen osoittautui kuitenkin hankalaksi. Ohjelmistokehyksen opettelu oli aloitettu, mutta tietämys oli vielä hyvin rajallista. Kokemuksen puute käyttöliittymäsovellusten kehittämisestä yhdessä heikon Qt-tietämyksen kanssa herättivät useita kysymyksiä luokkien toteutuksista, tarkoituksista ja keskinäisistä suhteista. Hylkäsin lopulta tavoitteen kattavasta määrittelystä keskittyen ohjelmointikehyksen opetteluun ja arkkitehtuurin tarkentamiseen ohjelmoinnin aikana. Ohjelman jako täsmällisiin moduuleihin

määrittelyvaiheessa helpottaa rakenteen hahmottamista, mutta määrittelyn vaikeuksien, kehyksen vaatiman oppimisajan ja yhden henkilön kehitystyön takia en pitänyt sitä tässä tapauksessa välttämättömänä.

5.3.2 Järjestelmän yleiskuvaus

Uuden työkalun toimintaympäristö tulisi olemaan laajempi kuin CDmacron, joka kopioi dokumenttikokoelmaan vain manuaalit. Toiminnan suunnittelussa tuli ottaa huomioon monta uutta tietoresurssia ja se, kuinka päästä niihin käsiksi. Ohjelman toimintaympäristö on esitetty kuvassa 4.



Kuva 4. ACS800 MPCompiler -hallintatyökalun toimintaympäristö

Sovelluksen toimintalogiikka hahmoteltiin tietoresurssien ja ohjelmalle asetettujen vaatimuksien pohjalta ennen ohjelmoinnin aloittamista. Ohjelma hakee käynnistyksen yhteydessä kaikki ACS800 Product Configuratorin projektit ja listaa ne käyttöliittymässä. Valitun projektin XML-dokumentti luetaan verkkolevyltä projektitunnuksen avulla, kun käyttäjä siirtyy esikatseluun. Suunnitelmana oli käsitellä XML-dokumentti jo projektivalinnan jälkeen, mutta tämän todettiin hidastavan liikaa projektien selaamista.

Dokumentin tiedoilla luodaan projektin tuotteille plussakoodit, joiden avulla saadaan manuaalisääntötiedostosta projektia koskevat manuaalit ja niiden sijainnit selville. Manuaalisäännöille jouduttiin kehitysprosessin aikana toteuttamaan XML-muunnos, koska Excel-dokumenttia ei voitu suoraan lukea Qt:ssä. Lisäksi projektimanuaalien tarkistusta varten toteutettiin oma näkymä, kun huomattiin tarve yksityiskohtaisempien tietojen näyttämiseen kuin mihin esikatselun manuaalilistausta oli järkevä laajentaa. Manuaalinäkymä näytetään projektimanuaalien keräämisen jälkeen ennen varsinaista esikatselutoimintoa, ja se pohjautuu esikatselun tavoin väliaikaismuistissa olevien ilmentymien käyttöön.

Esikatselua varten luodaan tietokoneen väliaikaismuistiin hakemistorakenne, jossa (oletusasetuksilla) näytetään kerätyt projektimanuaalit ja verkkolevyltä löytyvät mallidokumentit. Esikatselussa käyttäjä vastaa myös loppudokumenttien lisäämisestä dokumenttikokoelmaan. Loppudokumenttien keräystä ei haluttu automatisoida, koska kyseisten tiedostojen hankalan saatavuuden takia käyttäjän tulee itse olla vastuussa tehtävästä. Dokumentit luodaan useilla työkaluilla tai vastaavasti haetaan *SAP ERP* -toiminnanohjausjärjestelmästä, joten niiden tallennussijainnit vaihtelevat laajalti. Esikatseluvaiheen jälkeen kaikki väliaikaismuistiin listatut tiedostot kopioidaan dokumenttikokoelmaan. Samalla luodaan HTML-indeksisivu ja muokataan mallidokumentit.

5.3.3 Toiminnallisuus

Ohjelman toiminnallisuus suunniteltiin jaettavaksi neljään näkymään toimintalogiikan pohjalta. Resurssi-, projekti-, esikatselu- ja asetusnäkymillä oli selkeät toisistaan eroteltavat toiminnot, joista laadittiin kuvaukset toteutusvaiheessa hyödynnettäviksi. Työn edetessä resurssi- ja asetusnäkymät yhdistettiin ja projektimanuaaleille toteutettiin kokonaan uusi näkymä. Lisäksi Qt-kehystuntemuksen parantuessa suunnittelun aikana laadittuja toimintoja tarkennettiin ja uudelleenarvioitiin. Muutoksista huolimatta toiminnallisuuden pääperiaatteet säilyivät kuitenkin samoina.

Resurssien määrittäminen

Resursseilla tarkoitetaan ABB:n sisäverkosta löytyviä tiedostoja, joita ohjelma käyttää syötteinä dokumenttikokoelman luomisessa. Mahdollisuus resurssimäärityksien muokkaamiseen on tärkeää, koska tiedostojen sijainnit verkkolevyillä saattavat vaihtua. Suunnitelmana oli toteuttaa käyttäjästä riippuvainen dynaaminen resurssienlisäystoiminto, jossa resursseja voisi lisätä tai poistaa yksitellen ja hakemistokohtaisesti (ks. kuvan 3 resurssinäkymä). Toiminnosta luovuttiin, kun resurssien käyttöä lähdettiin tutkimaan uudelleen ja kävi ilmi, että omia resurssimäärityksiä tarvitaan odotettua vähemmän. Määritysten lukumäärän takia toiminnon hyödyt jäivät mahdollisista inhimillisistä virheistä aiheutuvien ongelmatapauksien varjoihin. Resurssit näytetään kuitenkin asetusvälilehden alla ohjelman virhetilanteiden varalle, mutta niitä ei voi muokata.

Projektimanuaalien kerääminen

Projektimanuaalit kerätään ennen esikatseluun siirtymistä. Ohjelma jäsentää valitun projektin XML-dokumentin ja luo tietojen avulla dokumentissa listatuille tuotteille täydelliset tyyppikoodin kattavat plussakoodit. Manuaalisääntötiedosto listaa manuaalien kuvaamat ominaisuudet plussakoodein, joten vertaamalla yhden tuotteen plussakoodia jokaisen manuaalin plussakoodiin saadaan selville, mitkä manuaalit kuvaavat kyseisen tuotteen ominaisuuksia. Esimerkiksi englanninkieliselle manuaalille *Sine Filters for ACS800 User's Manual* on listattu ominaisuus *Sine filter step-up transformer application* plussakoodilla E206. Jos projektiin kuuluisi tuote ACS800-407-0680-5+E206+..., tulisi edellä mainittu sinisuodattimista kertova manuaali lisätä projektimanuaaleihin. Prosessi toistetaan kaikille projektin tuotteille, kunnes projektimanuaalit on tallennettu listaukseen, joka kertoo manuaalien nimet ja tiedostosijainnit.

Dokumenttikokoelman esikatselu

Dokumenttikokoelman muokkaukseen tarkoitettu esikatselutoiminto pohjautuu väliaikaismuistissa sijaitsevan puumaisen hakemistorakenteen luomiseen. Toiminnossa ei vielä muokata tai kopioida mitään tiedostoja. Esikatseluun siirryttäessä kaikista

kerätyistä dokumenteista (projektimanuaalit ja mallidokumentit) luodaan tiedostoja kuvaavat ilmentymät, jotka näytetään hakemistorakenteessa. Työn alussa ei tiedetty, miten nämä ilmentymät olisi toteutettu tai mikä niiden rakenne olisi. Vaatimuksena oli vain, että ne sisältävät tiedon dokumentin nimestä ja hakemistopolusta. Muodostettavan dokumenttikokoelman rakennetta voi muokata lisäämällä, poistamalla tai järjestelemällä dokumentteja. Toiminnot suunniteltiin alun perin pelkkien painikkeiden ja *valinta-ikkunan* taakse, mutta käytettävyyden parantamiseksi toteutettiin vedä ja pudota (drag and drop) -ominaisuus esikatseluun lisätyn tiedostojärjestelmänäkymän avulla. Lisäksi esikatseluun lisättiin tiedostojen uudelleennimeämistoiminto.

Suunnitteluvaiheessa esikatselun toteutusta harkittiin muutettavan edellä mainitun menetelmän rajoituksien takia. Yksittäisiä dokumentteja ei voi muokata, koska niistä on vain merkinnät muistissa. Ratkaisu heikentää esikatselun tehokkuutta, kun esimerkiksi loppudokumentteja ei voi tarkastaa tai korjata. Täydellinen esikatselu olisi vaatinut tiedostosta paikallisen kopion aina dokumenttikokoelmaan siirrettäessä, jolla oltaisiin varmistettu, että tiedosto ei ole samanaikaisesti käytössä. Jatkuva tiedostojen kopioiminen olisi hidastanut toimintoa liikaa. Kompromissina suunniteltiin tiettyjen tärkeiden dokumenttien muokkausmahdollisuutta. Molemmista vaihtoehdoista luovuttiin ja päätettiin pysyä alkuperäisessä suunnitelmassa, joka selkeimmin erottaa dokumenttikokoelman hahmottelun ja generoinnin toisistaan.

Dokumenttikokoelman muodostus

Dokumenttikokoelman generoinnissa muodostetaan määritettyyn tallennushakemistoon esikatselun mukainen hakemistorakenne. Kerätyistä projektimanuaaleista ja mallidokumenteista, sekä esikatselussa lisätyistä loppudokumenteista on tallessa merkinnät, jotka sisältävät dokumentin sijainnin tiedostojärjestelmässä. Hakemistosijaintien avulla kopioidaan tiedostot dokumenttikokoelmaan ja muokataan mallidokumentit linjakokoonpanokohtaisesti. Samalla luodaan dokumenttikokoelmalle HTML-indeksisivu. Indeksisivun luominen pohjautuu HTML-mallidokumentin muokkaamiseen, sillä dokumentin kirjoittaminen kokonaan ohjelmallisesti olisi tarpeettoman työlästä. HTML-mallia ei

kuitenkaan kopioida verkkolevyltä, vaan se tallennetaan ohjelman ajettavaan exe-tiedostoon kääntämisen yhteydessä.

Toteutuksen aikana päätettiin myös tarkentaa dokumenttikokoelman hakemistorakennetta lisäämällä jokaiselle linjakokoonpanolle oma juurihakemisto. Alkuperäinen suunniteltu rakenne erotti vain manuaalit ja loppudokumentit toisistaan, mutta uudella rakenteella haluttiin ilmaista tieto linjakokoonpanon tarkkuudella. Linjakokoonpanohakemistoihin tulisi siis lisätä omat manuaalihakemistot. Ainoa tapa muuten selvittää, mitkä manuaalit koskevat mitäkin linjakokoonpanoa, olisi tarkastaa HTML-indeksisivulta. Dokumenttikokoelman koko haluttiin pitää pienenä, joten manuaalidokumenttien kopioimisen sijasta ohjelma luo generoinnissa linjakokoonpanohakemistoihin oikotietiedostot, jotka osoittavat manuaalien päähakemistossa sijaitseviin dokumentteihin. Koko generointiprosessin kulusta näytetään latauspalkki erillisessä valintaikkunassa, josta käyttäjä voi tarkastaa yksittäiset vaiheet.

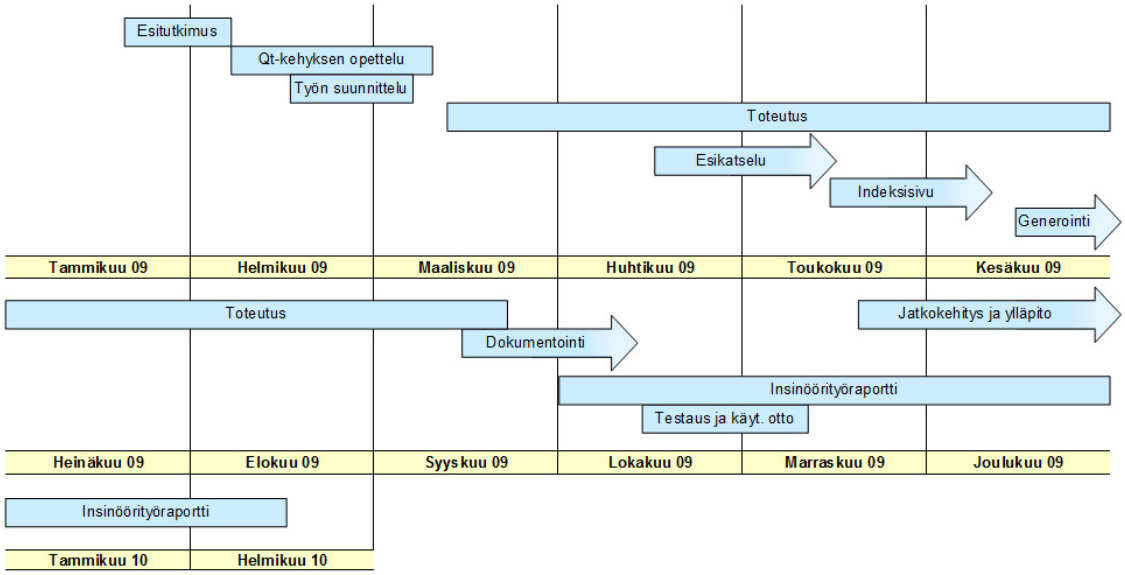
Ohjelman toiminnallisten asetusten määrittäminen

Ohjelman toimintaan vaikuttavat asetukset määriteltiin työn edetessä. Vaikka suunnitteluvaiheessa asetusten tarkkaa lukumäärää ei tiedetty, pidettiin tarpeellisena jakaa asetukset omaan näkymäänsä. Päätettiin, että ohjelma käyttää resurssimäärityksien ja toiminnallisten asetusten hallintaan pysyväismuistiin tallennettua asetustiedostoa. Suunnitelmana oli päivittää asetustiedostoa ajonaikaisesti, jolloin mahdolliset muutokset kirjoitettaisiin ja luettaisiin tiedostosta välittömästi. Tehokkuuden ja käytännöllisyyden takia päädyttiin säiliömenetelmään, jossa asetukset luetaan vain ohjelman käynnistyksen yhteydessä ja kirjoitetaan sulkemisen yhteydessä.

5.4 Aikataulu

Insinöörityöprojekti aloitettiin tammikuun lopulla 2009. Suunnitellun ja ohjeellisen aikataulun mukaan käyttöönotto määriteltiin kesäkuun puoliväliin, jolloin sovellus olisi ollut testattu ja työn arviointi olisi voitu suorittaa käyttäjäpalautteen mukaan heinäkuun alussa. Aikataulu venyi noin kolme kuukautta sovelluksen toteutuksen kestäessä syys-

lokakuun vaihteeseen. Kuvasta 5 käy ilmi sovelluskehityksen vaiheet ja lopullinen aikataulu.



Kuva 5. Insinöörityöprojektin lopullinen aikataulu

Syy aikataulun venymiseen oli pitkälti toteutusvaiheen keston aliarvioiminen. Kehitysprosessin aikana lisättiin myös uusia toiminnallisuuksia, joilla ohjelman käytettävyyttä ja käyttötarkoitukseen soveltuvuutta saatiin parannettua. Työrytmi vaikutti myös omalta osaltaan aikatauluun. Täysipäiväisiin viikkoihin voitiin siirtyä vasta kesäkuussa lukuvuoden päätyttyä. Palavereita pidettiin ohjausryhmän kanssa suunnitellusti kuukauden välein kehityssyklin aikana. Palavereissa määriteltiin sovelluksen esivaatimukset, mutta työn edetessä niiden luonne muuttui enemmän seurantapalavereiksi, joissa tiedotettiin uusista ominaisuuksista, mietittiin jatkoaikataulua ja tarvittaessa tarkennettiin ohjelman vaatimuksia.

6 Käytetyt tekniikat ja työkalut

6.1 Qt-ohjelmistokehys

6.1.1 Monialustainen kehitysympäristö ja luokkakirjasto

ACS800 MPCompiler on kehitetty käyttäen Qt-kehyksen Open Source Edition -laitoksen versiota 4.5. Qt on alun perin norjalaisen Trolltech-yhtiön kehittämä sovelluksien ja käyttöliittymien kehittämiseen tarkoitettu ohjelmistokehys, joka on saatavilla Mac OS X-, Linux-, Windows-, S60- ja sulautettuihin Linux-pohjaisiin alustoihin [7]. Qt sisältää C++-luokkakirjaston ja kehitysympäristön, ja siitä on saatavilla kaksi eri laitosta. Ilmainen Open Source Edition on LGPL-lisenssin alainen laitos, jossa muutokset kirjaston lähdekoodiin täytyy julkistaa ja maksullisten ohjelmien kehittäminen on rajoitettua. Teknisen tuen sisältävä Commercial Edition -laitos ei sisällä edellä mainittuja rajoituksia, mutta se on saatavilla vain maksullisella lisenssillä. Kehys on erittäin hyvin dokumentoitu. Moduulien ja luokkien käyttöön löytyy kattavat ja täsmälliset ohjeet, joissa esimerkkien avulla esitetään tärkeimpien toimintojen käyttö. Qt:n kotisivuilta löytyy tutoriaaleja uusille kehittäjille, manuaaleja moduulien käyttöön ja esimerkkiohjelmia, joiden lähdekoodia voi vapaasti hyödyntää. [8; 9.]

6.1.2 Ominaisuudet

Qt:n oliomallin tuoma keskeisin ominaisuus C++:aan ja samalla myös suurin ero muihin ohjelmistokehyksiin verrattuna, on olioiden keskinäisen kommunikaation mahdollistava signal-slot-mekanismi. Muita tärkeitä ominaisuuksia ovat tapahtumat ja tapahtumasuodattimet, yhteysriippuvainen merkkijonojen kielikäännös, turvallinen QPointer-osoitinluokka, mahdollisuus määrätä olioille dynaamisia ajonaikaisia ominaisuuksia sekä olioiden keskinäistä rakennetta havainnollistava hierarkkinen puurakenne. Osa näistä on toteutettu standardeilla C++-tekniikoilla, mutta mekanismit kuten signal-slot ja dynaamiset ominaisuudet vaativat Qt:n oliomalliin pohjautuvaa metaoliojärjestelmää (*Meta-Object System*). [10.]

Signal-slot-mekanismi

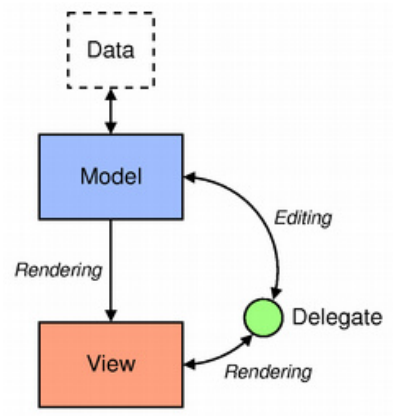
Olio lähettää signaalin tietyn tilanvaihdon seurauksena. Qt:ssä on useita esimääritettyjä signaaleita, jotka lähetetään esimerkiksi valintaruudukon arvon vaihtumisen tai napin painalluksen seurauksena, mutta ohjelmoija voi myös kirjoittaa niitä itse. Tapahtuma (signal) yhdistetään yhteen tai useampaan haluttuun käsittelijäfunktioon (slot), joka on käytännössä normaali C++-jäsenfunktio. Esimerkiksi yllämainitussa valintaruudukon tapauksessa vaihtunut arvo voitaisiin tarkastaa käsittelijässä ja suorittaa tuloksen perusteella jatkotoimenpiteet. Signal-slot-mekanismin käyttö on nopeata ja vaivatonta, ja ennen kaikkea se helpottaa olioiden kommunikoinnin hallintaa. [11, s. 6, 20–21.]

Hierarkkinen oliorakenne

Qt:n oliomallin sydämenä toimii *QObject*-kantaluokka. Kantaluokasta periytyville olioille voidaan luomisen yhteydessä määritellä vanhempi (parent), joka tuhoutuessaan huolehtii kaikkien lapsiolioidensa tuhoamisesta. Välittämällä luotu lapsi jonkun toisen olion vanhemmaksi ja jatkamalla tätä sukupuuta saadaan lopulta muodostettua hyvin laaja hierarkinen oliorakenne, jossa esimerkiksi pääikkunaa suljettaessa tuhotaan automaattisesti myös kaikkien alinäkymien *käyttöliittymäelementit*. Tämä automaattinen muistinhallintamekanismi helpottaa ohjelmoijan työtä, koska enää ei tarvitse huolehtia kaikkien dynaamisesta muistista varattujen olioiden tuhoamisesta. [11, s. 28.]

Model/view-arkkitehtuuri

Model-View-Controller (MVC) -mallia käytetään usein käyttöliittymien rakentamisessa. Siinä on eroteltu datan säilöminen (model), esittäminen (view) ja käyttöliittymän syötteet (controller) toisistaan. Qt:ssä datan esittäminen ja käyttöliittymän syötteet on yhdistetty keskenään ja näin on saatu yksinkertaisempi, mutta yhtä joustava model/view-arkkitehtuuri. Arkkitehtuuria käytetään datan ja sen esitysmuodon keskinäisten suhteiden hallinnointiin. Tällä toiminnallisuuden ja ulkoasun erottelulla voidaan tukea saman tiedon esittämistä useassa eri muodossa. Kuvassa 6 nähdään model/view-arkkitehtuurin komponentit ja niiden välinen rakenne.



Kuva 6. Qt-ohjelmistokehyksen model/view-arkkitehtuuri [12].

Model/view-arkkitehtuurissa malli (model) ylläpitää rajapintaa lähdetietoon, jotta muut arkkitehtuurin komponentit pääsevät siihen käsiksi. Qt tarjoaa useita valmiita malleja, mutta ohjelmoija voi halutessaan periä *QAbstractItemModel*-kantaluokan ja toteuttaa laajemman rajapinnan, jos räätälöidyistä vaihtoehdoista ei löydy sopivaa. Näkymä (view) huolehtii mallin välittämän lähdedatan esittämisestä. Tietoon viitataan lyhytaikaisilla *QModelIndex*-luokan ilmentymillä, jotka kuvaavat yhden tietoalkion sijainnin. Qt:n näkymät pohjautuvat *QAbstractItemView*-kantaluokkaan, josta vastaavasti on saatavilla useita eri käyttötarkoituksiin sopivia aliluokkia. Räätälöidyn näkymän toteuttaakseen ohjelmoija voi periä jonkun näistä luokista. Kolmas osapuoli model/view-arkkitehtuurissa on delegaatti (delegate), joka toteuttaa yksittäisen tietoalkion ulkoasun ja mahdollistaa käyttäjän syötteiden vastaanottamisen. [11, s. 217–219; 12.]

6.2 Windows API

Windows API on Microsoftin kehittämä Windows-käyttöjärjestelmien ohjelmointirajapinta. Sen palveluita käyttävät kaikki Windows-pohjaiset sovellukset. Ohjelmoija voi rajapinnan avulla rakentaa käyttöliittymiä, käyttää järjestelmäresursseja, piirtää grafiikkaa sekä sisällyttää videota, audiota, verkostotyöskentelyä ja tietoturvaa [13]. Qt:n kirjastot käyttävät Windows-alustalla Windows API -rajapintaa, joten ohjelmoija voi vapaasti yhdistää Windows API -komentoja Qt:n kanssa samaan lähdekooditiedostoon. Tämä voi joskus olla välttämätöntä, jos Qt ei tue jotain haluttua ominaisuutta, kuten insinöörityössä kaivattua ActiveX-tekniikkaa Microsoft Word- ja Excel -doku-

menttien muokkaamiseen. Työssä jouduttiin turvautumaan suoraan Windows API -rajapinnan komentoihin paperimappien mallidokumenttien muokkaustoiminnon toteuttamisessa. Dokumentit muokataan varsinaisesti VBA-makrojen avulla, mutta makrojen kutsumiseen käytetään *COM*-olioita.

6.3 Visual Basic for Applications

Microsoftin Visual Basic for Applications (VBA) on sulautettu kehitysympäristö, joka tarjoaa Visual Basic -ohjelmointikielen työkalut olemassa olevan sovelluksen laajentamiseksi. Microsoftin ohjelmista Access, Excel, Outlook, PowerPoint ja Word sisältävät VBA 6.5 -ympäristön [14]. VBA on tietyssä mielessä rajoitettu versio Visual Basic -ohjelmointikielestä, koska sitä voi käyttää vain ohjelmassa, johon se on sulautettu. Lisäksi VBA:n syntaksi ja ominaisuudet eroavat hieman Visual Basicista, koska se käyttää liitetyn sovelluksen oliokirjastoa [15]. Insinöörityössä VBA-ympäristöllä kirjoitettiin makrot manuaalisääntötiedostolle ja kolmelle muokattavalle mallidokumentille. Manuaalisääntötiedoston makro suorittaa listattujen manuaalien XML-muunnoksen, jotta ohjelma voi kerätä projektimanuaalit. Mallidokumenttien makrot vastaavasti huolehtivat linjakokoonpanokohtaisten päätylehtien ja sisällysluetteloiden luomisesta.

6.4 jQuery

jQuery on ilmainen vapaassa jakelussa oleva *JavaScript*-kirjasto, joka tukee kaikkia moderneja selaimia (Internet Explorer 6.0, Mozilla Firefox 2, Safari 3.0, Opera 9.0 ja Google Chrome). Kirjasto helpottaa huomattavasti JavaScriptin ominaisuuksien käyttöä, kuten HTML-dokumentin tarkastelua, tapahtumien käsittelyä ja animaatioiden tekoa [16]. Kirjastoon on lisäksi saatavilla useita ilmaisia harrastajien tekemiä liitännäisiä, jotka helpottavat entisestään verkkosivustojen tekoa. Edellä mainittuja ominaisuuksia käytettiin insinöörityössä dokumenttikokoelmalle muodostettavan HTML-indeksisivun toteuttamisessa. jQueryn työkaluilla luotiin sivun navigointi ja navigointiin liittyvät animaatiot. Navigointi perustuu HTML-elementtien näkyvyyksien vaihteluun.

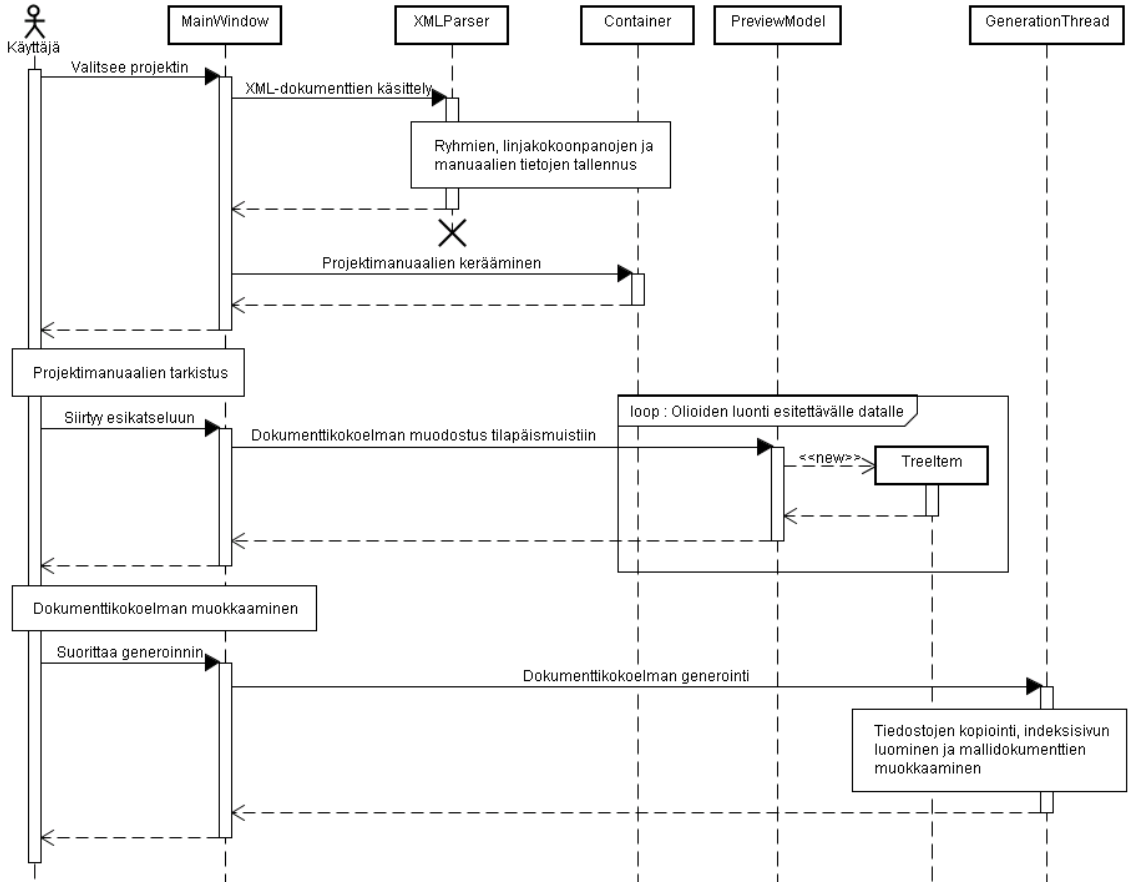
6.5 Työkalut

Qt:hen on saatavilla useita kehitystyökaluja, jotka helpottavat työn suunnittelua ja toteuttamista. Käyttöliittymän hahmotteluun ja valintaikkunoiden luomiseen käytettiin Qt Designer -ohjelmaa, joka on käyttöliittymän visuaaliseen suunnitteluun ja rakentamiseen tarkoitettu työkalu. Sillä luodut ui-tiedostot voi kääntää luokkiin liitettäviksi otsikkotiedostoiksi, jolloin käyttöliittymän toiminnot jäävät ohjelmoijan toteutettaviksi. Ohjelman kirjoittamisen apuna käytin verkossa olevan dokumentaation esittämiseen tarkoitettua Qt Assistant -työkalua. Oletusasetuksilla se sisältää Qt-kehysdokumentaation ja tarjoaa toiminnot sen nopeaan selaamiseen. Sovitettavuutensa ansiosta työkalua voi käyttää myös omissa ohjelmissa esimerkiksi käyttöohjeiden näyttämiseen. Kehitysympäristöksi valitsin uuden ja kevyen Qt:hen kuuluvan Qt Creator -sovelluksen. Työkalu osoittautui helppokäyttöiseksi, vaikkakin hieman monipuolisuuden kustannuksella. Open Source Edition -laitokselle ei löytynyt maaliskuussa 2009 tukea Qt:n ja aikaisemmin käyttämäni Microsoft Visual Studio -kehitysympäristön integraatiolle. Se löytyi vain maksullisesta Commercial Editionista. Tuki lisättiin ilmaislaitokseen vasta myöhemmin erillisen lisäosan julkaisulla, mutta Qt Creatorin parannuttua lastentaudeistaan kehitysympäristön vaihtaminen ei ollut tarpeellista [17]. Sovelluskehityksen loppupuolella ilmeni kasvava kuvankäsittelyn tarve, kun ohjelman ulkoasua aloitettiin viimeistellä. Logon luomiseen, painikkeiden ikonien muokkaamiseen sekä ohjelman graafisen yleisilmeen kehittämiseen käytin ilmaista kuvankäsittelyohjelmaa GIMP (GNU Image Manipulation Program).

7 Toteutus

7.1 Ohjelman luokkarakenne

ACS800 MPCompilerille kirjoitetut luokat ovat jaettavissa käyttöliittymään, tietovarastoihin ja tallennukseen, malleihin sekä generointiprosessiin. Ohjelman toimintojen kehittyessä useista Qt:n valmisluokista toteutettiin perityt ja monipuolisemmat luokat. Kuvassa 7 on esitetty ohjelman tärkeimmät luokat ja yksinkertainen kuvaus niiden toiminnasta, kun käyttäjä luo dokumenttikokoelman.



Kuva 7. ACS800 MPCompilerin toiminta dokumenttikokoelman luomisessa

Käyttäjän yhteydestä käyttöliittymään vastaa *MainWindow*-luokka. Se toimii käyttöliittymän perustana ja huolehtii model/view-rajapinnan näkymien sekä muiden käyttöliittymäelementtien luomisesta. Manuaali- ja esikatselunäkymän model/view-arkkitehtuurien näkymille kirjoitettiin luokat *ManualView*, *PreviewView* ja *FileSystemView*. Näillä

haluttiin laajentaa Qt:n valmisluokan ja tiedon puumaisesti esittävän *QTreeView*-kantaluokan ominaisuuksia. Muiden näkymien ja visuaalisten elementtien esittämiseen käytetään pääosin kehyksen tarjoamia käyttöliittymäelementtejä, mutta valintaikkunoita varten luotiin omat räätälöidyt *ProgressDialog*-, *AboutDialog*- ja *InsertManualDialog*-luokat. Luokat olivat tarpeellisia muodostusprosessista tiedottamisen, sovellustietojen esittämisen ja manuaalidokumenttien lisäystoiminnon aikaansaamiseksi.

XMLParser-luokka vastaa valitun projektin ja manuaalien tietojen keräämisestä XML-dokumenteista. Yhden manuaalin, linjakokoonpanon tai ryhmän tiedot tallennetaan sitä vastaavan *Manual*-, *LineUp*- tai *Unit*-luokan olioon. Projektin tiedot tallennetaan *Container*-luokan olioon, joka samalla toimii ohjelman keskeisenä tietosäiliönä mahdollistaen pääsyn lähes kaikkiin ohjelman resursseihin.

Manuaali- ja esikatselunäkymä käyttävät *ManualModel*- ja *PreviewModel*-malleja projektitietoihin pääsemiseksi. *ManualModel* huolehtii projektimanuaalien listaamisesta, ja *PreviewModel* vastaa väliaikaismuistiin luodusta dokumenttikokoelmasta. Tietovarastojen ja mallien välille toteutettiin *TreeItem*-luokka. Sitä voisi luonnehtia kääreluokaksi (wrapper class), joka sisältää tietovarastojen datan puumaista hierarkiaa noudattavan malliluokan ymmärtämässä muodossa. Yksi *TreeItem*-olio kuvaa yhtä näytettävää alkiota eli projektimanuaalia, hakemistoa, loppudokumenttia tai muuta tiedostoa. Jokaiselle oliolle voidaan myös määritellä vanhempi- ja useita lapsiolioita, joilla mahdollistetaan hierarkkisen rakenteen luominen. *TreeItem*-luokkaa lähdetietonaan käyttävät mallit perivät abstraktin *TreeModel*-kantaluokan, joka toteuttaa rajapinnan mallin ja lähdetiedon välille. *TreeModel* perii vastaavasti kaikkien Qt:n mallien kantaluokkana toimivan *QAbstractItemModel*-luokan ja toteuttaa suurimman osan model/view-rajapinnalta vaadituista toiminnoista, kuten vanhempi- ja lapsisolmujen hallinnan. Perimmäinen tarkoitus luokan kirjoittamiselle oli *ManualModel*- ja *PreviewModel*-luokkien yhteisten ominaisuuksien yhdistäminen.

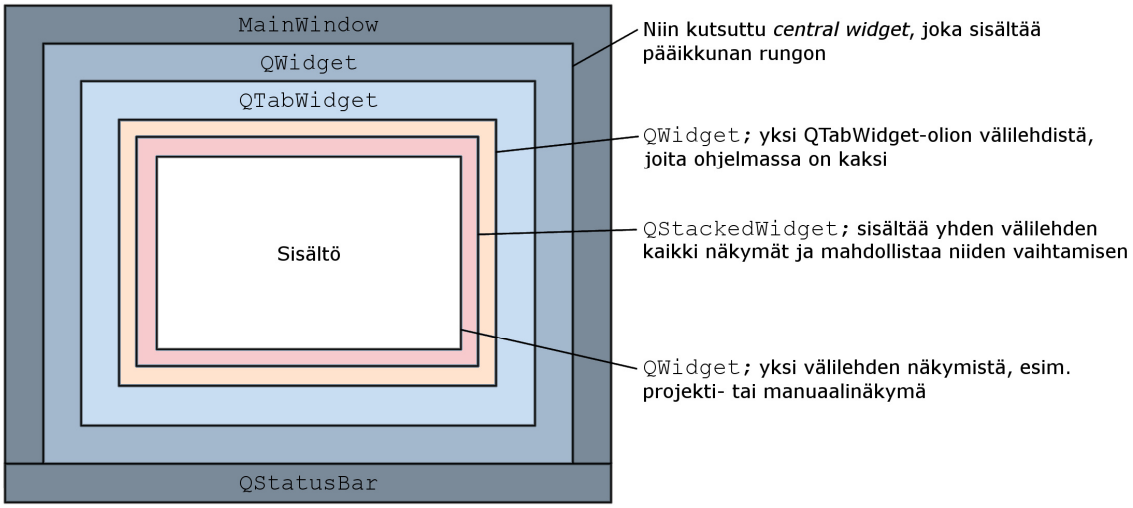
Dokumenttikokoelman muokkaamisen avuksi esikatseluun lisättiin tiedostojärjestelmänäkymä vedä ja pudota -ominaisuudelle. *FileSystemView*-näkymä käyttää *FileSystemModel*-mallia päästäkseen käsiksi siirrettävien tiedostojen tietoihin. Viimeinen toteutet-

tu malli on *ProxyModel*, joka huolehtii tietokannan projektitietojen suodattamisesta. Dokumenttikokoelman generointi suoritetaan lopulta omassa säikeessä *GenerationThread*-luokassa. Toteutus noudattaa muuten Qt-kehyksen tarjoamia ratkaisuja, mutta mallidokumenttien muokkaamisessa käytetään Windows API -komentoja. Kaikki ohjelmalle kirjoitetut luokat selviävät liitteessä 3 esitetystä typistetystä luokkakaaviosta.

7.2 Käyttöliittymän ohjelmointi

ACS800 MPCompilerin käyttöliittymä toteutettiin osittain Qt Designer -suunnittelutyökalulla, mutta suurimmaksi osaksi käsin ohjelmoimalla. Qt Designeriä käytettiin käyttöliittymän pohjan suunnitteluun käyttäen apuna käsin piirrettyjä paperiluonnoksia. Työkalun luoma ohjelmakoodi päätettiin kuitenkin kopioida kokonaisuudessaan *MainWindow*-käyttöliittymäluokkaan sen sijaan, että olisi käytetty sitä alkuperäisessä muodossaan C++-otsikkotiedostona. Muutoksella saatiin vapaat kädet vaikuttaa ohjelmakoodin rakenteeseen ja käyttöliittymän toimintaan, sekä samalla tuli opittua, kuinka käyttöliittymärajapinta syntyy. Jälkeenpäin ajateltuna järkevintä olisi ollut pysyä Qt Designerin käytössä, koska lukuisten käyttöliittymämuutosten käsin ohjelmoiminen osoittautui varsin työlääksi, vaikkakin opettavaiseksi tehtäväksi. Osasyynä Qt Designerista luopumiseen oli myös Qt-tietämyksen puute. Työn alussa en tarkkaan tiennyt, kuinka liitetyn otsikkotiedoston kautta voi vaikuttaa käyttöliittymän rakenteeseen ja toiminnallisuuteen. Vaikka käyttöliittymän runko ohjelmoitiin käsin, Qt Designer otettiin myöhemmin uudelleen käyttöön *ProgressDialog*-, *AboutDialog*- ja *InsertManualDialog* -valintaikkunoiden suunnittelussa ja toteuttamisessa.

Ohjelmaan toteutettiin seuraavat neljä näkymää: projektinäkymä, manuaalinäkymä, esikatselunäkymä ja asetusnäkymä. Käyttöliittymän perusta on rakennettu monesta päällekkäin pinotuista eri tyypin käyttöliittymäelementeistä. *MainWindow*-luokka toimii pääikkunana, mutta eri näkymien sisältö, kuten projektinäkymän projektilistaus, ei itse asiassa ole luokan olion suora jälkeläinen (hierarkista oliorakennetta ajatellen). Pääikkunan ja näkymien sisällön väliin kuuluu viisi muuta *QWidget*-, *QTabWidget*- tai *QStackedWidget*-luokan oliota. Kuvassa 8 on hahmoteltu käyttöliittymän rakennetta näiden olioiden avulla.



Kuva 8. Käyttöliittymän rakenne

Kuten kuvasta nähdään, käyttöliittymän runko pohjautuu pitkälti yhteen välilehtiä hallinnoivaan *QTabWidget*-luokan olioon. Tämä ratkaisu osoittautui toimivaksi, eikä missään vaiheessa esiintynyt tarvetta valikko- tai työkalupalkille. Käyttöliittymän ulkoasun ja asettelun viimeistelyssä käytetään tyylitiedostoa. Qt:n tyylitiedostot muistuttavat hyvin läheisesti *CSS*-tyylitiedostoja, niin ominaisuuksiltaan kuin syntaksiltaan. Tyylitiedostolla voi muokata muun muassa käyttöliittymäelementtien kokoja, marginaaleja, värejä, kirjasintyyppejä ja sijainteja. Tyylitiedostot helpottavat käyttöliittymän hallintaa huomattavasti, kun yksinkertaisilla komennoilla voi suorittaa lähes kaiken tarvittavan hienosäädön.

7.3 Tietoperusta

7.3.1 Resurssijärjestelmä

Ohjelma hyödyntää Qt:n resurssijärjestelmää (resource system), jolla voidaan tallentaa ulkoisia tiedostoja binäärimuotoisena ohjelman ajettavan exe-tiedoston yhteyteen. Näitä resursseiksi kutsuttuja tiedostoja ei pidä sekoittaa saman nimisiin, vain ACS800 MPCompilerissa käytettyihin ABB:n sisäverkon resursseihin, jotka eivät liity tässä luvussa käsiteltyyn järjestelmään millään tavalla.

Qt:n resurssijärjestelmän avulla varmistetaan ohjelman tarvitsemien ulkoisten tiedostojen olemassaolot ilman pelkoa niiden tuhoutumisesta ja menettämisestä. Järjestelmän käyttämät resurssit listataan XML-muotoiseen qrc-tiedostoon, joka luetaan ohjelman kääntämisen yhteydessä. Ohjelmakoodissa resursseihin viitataan hakemistopoluilla tai vaihtoehtoisilla aliaksilla. Resurssiviittaukset erotetaan tavallisista merkkijonoista kaksoispisteellä ja kenoviivalla, jolloin käytettävä luokka ymmärtää pyytää datan resursseista. Esimerkiksi resursseihin tallennettu ohjelman tyylitiedosto luetaan käynnistyksen yhteydessä suoraan **`QFile`**-luokan avulla (koodilistaus 1).

```
QFile file(":/stylesheet.qss");
file.open(QFile::ReadOnly);
QString styleSheet = QString(file.readAll());
qApp->setStyleSheet(styleSheet);
```

Koodilistaus 1. Tyylitiedoston lukeminen resursseista ja käyttöönotto

Oheisessa esimerkissä tyylitiedostolle on annettu yksikäsitteinen *stylesheet.qss*-alias, joten hakemistopolku on voitu typistää. Resurssijärjestelmää käytettiin kaikkien ulkoisten tiedostojen tallentamiseen, pois lukien ohjelman käyttöohjeet ja niiden esittämisessä käytetty Qt Assistant -työkalu. Resurssitiedostoon listattiin tyylitiedoston lisäksi muun muassa ohjelman ikonit ja kuvat, indeksisivun HTML-mallitiedosto, sivun CSS ja JavaScript-lähdetiedostot sekä jQuery-kirjasto.

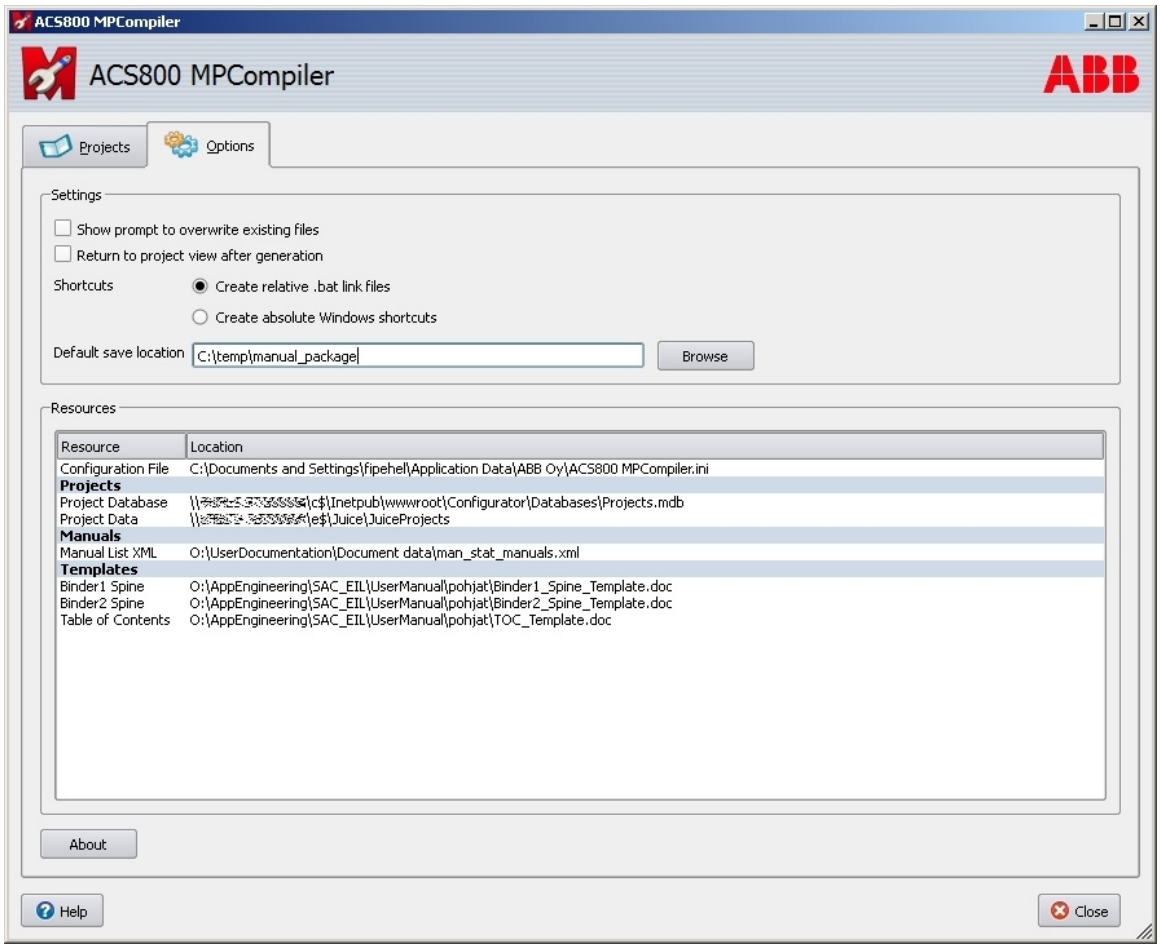
7.3.2 Tietokantayhteys

Yhteys projektitietokantaan luodaan ohjelman käynnistyksen yhteydessä. Tietokanta ja projektien XML-tiedostot sijaitsevat ABB:n verkkolevyillä, joihin käyttäjän tietokoneesta riippuen ei välttämättä ole luotu yhteyksiä. Ennen tietokantayhteyden muodostamista täytyy siis varmistaa yhteydet verkkolevyn molempiin levyasemiin. Tähän tarkoitukseen käytetään Windows-käyttöjärjestelmän *net.exe*-komentoa, jolla voi päivittää, korjata tai tulostaa tietokoneen lähiverkkoyhteyksiä tai -asetuksia [18]. Komento ajetaan toisessa prosessissa käyttäen *QProcess*-luokkaa, jolle välitetään parametreina kytkettävän verkkolevyn levyasema sekä valtuutetun käyttäjän tunnus ja salasana. Avatut lähiverkkoyhteydet katkaistaan ohjelman sulkeutuessa samaa komentoa käyttäen. Tietokan-

tayhteyden muodostamiseen käytetään Qt:n *ODBC*-tietokanta-ajuria. Virhetilanteissa käyttäjälle huomautetaan viestilaatikolla, mutta ohjelman käynnistystä ei kuitenkaan estetä. Kyseessä saattaa olla vain virheellinen tietokantapolku, jonka käyttäjä voi tarkastaa asetusnäkymästä ja korjata asetustiedostoon.

7.3.3 Sovellusasetukset

Tallennetut sovellusasetukset on sijoitettu asetusnäkymään. Näkymä on jaettu kahteen osa-alueeseen. Ohjelman toiminnallisuutta koskevat asetukset on sijoitettu ylälaitaan ja ABB:n sisäverkon resurssiviittaukset näiden alle. Resurssiviittauksia ei ole mahdollista muokata käyttöliittymän kautta. Kuvakaappaus asetusnäkymästä nähdään kuvassa 9.



Kuva 9. Ohjelman asetusnäkymä

Toiminnallisten asetusten ja ABB:n resurssiviittauksien tallennukseen käytetään tekstipohjaista ini-tiedostoa. Qt tarjoaa asetuksien käsittelyyn *QSettings*-luokkaa, joka Windows-alustalla oletuksena tallentaa asetukset rekistereihin ohjelmassa määritettyjen organisaatio- ja sovellusnimien perusteella. Tallennusmuoto päätettiin kuitenkin vaihtaa, koska ulkoisesta tiedostosta voidaan tarkastaa ja muokata resurssimäärityksiä silloinkin, kun ohjelmaa ei käytetä. Ohjelman poistamisen jälkeen ei myöskään ole vaaraa, että rekisteriin jäisi ylimääräisiä kuormittavia merkintöjä. Jatkuva asetustiedoston lukeminen kuitenkin hidastaisi ohjelman toimintaa. Päädyttin ratkaisuun, jossa kaikki asetukset luetaan *Container*-säiliöluokkaan ohjelman käynnistyksen yhteydessä. Vastaavasti asetukset tallennetaan vasta ohjelman sulkeutuessa. Tällöin asetustiedosto myös luodaan automaattisesti, mikäli sitä ei ole olemassa, kuten ohjelman ensimmäisellä ajokerralla. Sovellusasetukset luetaan ajon aikana suoraan säiliöstä käyttäen funktiokutsun parametrina asetuksen roolia kuvaavaa enum-tyyppiä (koodilistaus 2).

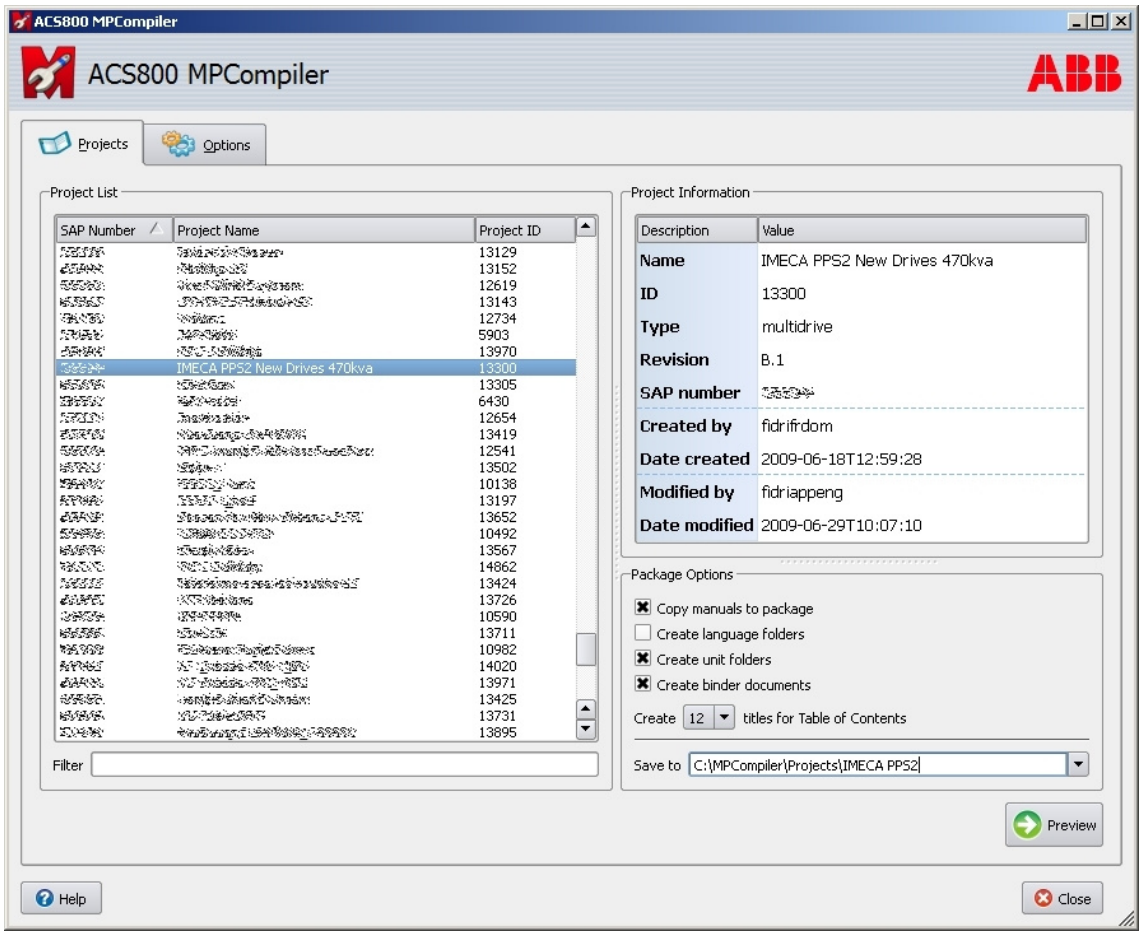
```
QString TOCPath =
m_container->setting(Settings::TableOfContents).toString();
QString binder1SpinePath =
m_container->setting(Settings::Binder1Spine).toString();
QString binder2SpinePath =
m_container->setting(Settings::Binder2Spine).toString();
```

Koodilistaus 2. Asetuksien luku Container-oliosta ohjelman suorituksen aikana

Enum-arvoja käytetään laajalti ohjelmakoodissa helpottamaan säiliöiden sisältöjen ja *TreeItem*-olioiden hallintaa. Jokaiselle *TreeItem*-oliolle määritellään alustamisen yhteydessä myös tyyppi, joka kertoo, minkätyyppistä tietoa sillä kuvataan ja miten sitä kuuluu käsitellä. Esimerkiksi enumilla *ManualItem::Category* kerrotaan, että olio kuvaa yhtä manuaalinäkymän kategoriaa. Kaikkien luokkien keskeisimmät enum-tyypit koottiin yhteen *roles.h*-otsikkotiedostoon. *Container*-säiliön kautta useilla luokilla on pääsy yhteiseen dataan ja samojen määrityksien toistaminen jokaisessa otsikkotiedostossa erikseen on tarpeetonta.

7.4 Projektien listaaminen ja valitseminen

Ohjelman aloitusnäkymänä toimii projektinäkymä. Näkymässä käyttäjä voi valita projektin, tarkastaa sen tiedot ja määrittää dokumenttikokoelman rakenteeseen ja sisältöön vaikuttavat pakettiasetukset. Kuvakaappaus projektinäkymästä on esitetty kuvassa 10.

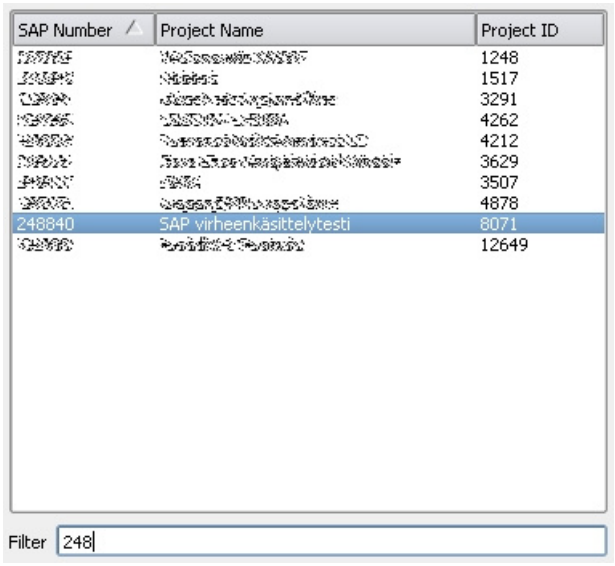


Kuva 10. Ohjelman projektinäkymä

Projektit haetaan *SQL*-kyselyllä kerran ohjelman käynnistymisen yhteydessä, jonka jälkeen tietokantayhteys suljetaan. Projektilistaan haetaan kaikki ACS800 Product Configurator -järjestelmällä Suomeen tehdyt projektit, joiden kauppanumero on kelvollinen ja tila on edennyt tilausvaiheesta eteenpäin.

Projektilista

Projektilistassa ei näytetä kaikkia SQL-kyselyn tuloksia, vaan jokaisesta projektista näytetään vain kauppanumero, nimi ja tunnus. Tulosten suodattamiseen käytetään edustajamallia (proxy model), jonka tarkoituksena on hallinnoida, mitä osia ja missä järjestyksessä alkuperäisen mallin dataa näytetään käyttäjälle. Käytäntö on hyödyllinen varsinkin tietokantojen suodattamisessa, kun halutaan näyttää vain osa alkuperäisestä sisällöstä tai antaa käyttäjän itse suodattaa tietoa. Edustajamalli tarvitsee aina kumppanikseen lähdemallin, josta tietoa suodatetaan. Projektilistan tapauksessa lähdemalli on *QStandardItemModel*-valmisluokan olio, jolle *ProxyModel*-edustajamalli on toteutettu siten, että lähdemallista näytetään vain halutut tietokannan sarakkeet. Yksittäisten projektien suodattaminen onnistuu *QRegExp*-tyypin jäsenmuuttujan avulla, jota voi käyttää säännöllisten lausekkeiden (regular expression), korvausmerkkikaavojen (wildcard pattern) tai normaalien merkkijonojen tunnistamiseen. Ominaisuutta käytettiin kuvassa 11 esitetyn projektien hakutoiminnon toteuttamiseen.



Kuva 11. Projektilistan hakutoiminto

ProxyModel-mallissa *QRegExp*-muuttujaa käsitellään normaalina merkkijonona määrittämään, kuuluuko rivi näytettävien joukkoon. Projektilistan hakuehtojen mukainen automaattinen päivitys saadaan helposti aikaiseksi muodostamalla kuvassa nähdyn tekstilaatikon ja edustajamallin välille signal-slot-yhteys (koodilistaus 3).

```
connect(m_filterLineEdit, SIGNAL(textChanged(QString)),
        m_proxyModel, SLOT(setFilterFixedString(QString)));
```

Koodilistaus 3. Signal-slot-yhteyden luominen kahden QObject-kantaluokasta periytyvän olion välille.

Projektitiedot

Valitusta projektista näytetään yksityiskohtaisemmat tiedot näkymän oikeassa yläreunassa. Yhteenveto sisältää projektin nimen, tunnuksen, tyypin (multidrive vai single drive), revision, kauppanumeron, luojan ja viimeisen muokkaajan käyttäjätunnukset sekä näiden tapahtumien päivämäärät. Sisältö koostuu vain tietokannasta löytyvästä perustiedosta, koska projektien XML-dokumenttien jäsentäminen tässä vaiheessa hidastaisi liikaa valintaprosessia. Alun perin suunnitelmana oli näyttää dokumenteista saatavat linjakokoonpanojen ja ryhmien lukumäärät, mutta päädyttiin tähän käytännöllisempään ratkaisuun.

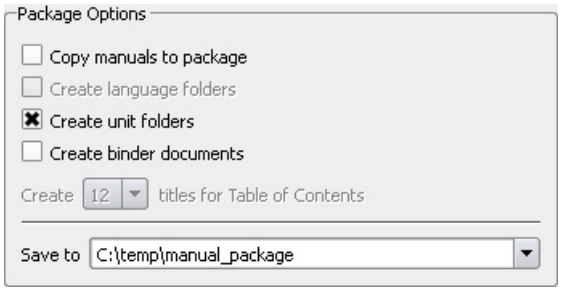
Ohjelman ja käyttöliittymäelementtien yleiseen ulkoasuun pystyy vaikuttamaan monipuolisesti tyylitiedostolla, mutta näkymäluokkien yksittäisissä tietoalkioissa joudutaan turvautumaan muihin ratkaisuihin. Projektin yhteenvetoon kaivattiin selkeyttä, joten tälle *QTreeWidget*-valmisluokan oliolle päätettiin toteuttaa oma delegaatti. *ProjectInfoDelegate*-luokassa muun muassa piirretään kuvassa 12 nähtävät erotinviivat ja maalataan *Description*-sarakkeelle tausta, jotta projektitiedot erottautuvat selkeämmin. Delegaatin toteuttaminen on paljon työläämpi ratkaisu ulkoasun muokkaamiseen kuin tyylitiedosto, mutta sillä saadaan huomattavasti yksityiskohtaisempaa jälkeä aikaan.

Description	Value
Name	SAP virheenkäsittelytesti
ID	8071
Type	multidrive
Revision	A.7
SAP number	248840
Created by	fidriappeng
Date created	2008-02-06T12:33:04
Modified by	fidriappeng
Date modified	2008-02-07T11:43:10

Kuva 12. Valitusta projektista näytettävät tiedot

Pakettiasetukset

Projektinäkymän viimeinen osa-alue on oikeassa alareunassa sijaitsevat pakettiasetukset. Asetuksien tarkoitus on antaa käyttäjän vaikuttaa dokumenttikokoelman sisältöön ja rakenteeseen toimitus- ja käyttökohteiden mukaisesti. Ne poikkeavat sovellusasetuksista vaikuttamalla suoraan dokumenttikokoelmaan, joten ne on asetettava ennen esikatselunäkymään siirtymistä. Pakettiasetuksilla voidaan määrittää, kopioidaanko kerättävät projektimanuaalit kokoelmaan, laajennetaanko hakemistorakennetta manuaalien ja linjakokoonpanojen kieli- ja ryhmäkohtaisilla alihakemistoilla, luodaanko paperimapeille päätylehdet ja sisällysluettelo sekä sisällysluettelossa listattujen otsikkojen lukumäärä. Lisäksi voidaan määrittää oletuksesta poikkeava dokumenttikokoelman tallennushakemisto. Pakettiasetukset ovat nähtävillä kuvassa 13. Tallennushakemistoa lukuun ottamatta kuvassa esitetyt arvot eivät ole samoja kuin oletusarvot (ks. kuva 10), jotka asetetaan käynnistyksen yhteydessä.



Kuva 13. Projektinäkymän pakettiasetukset

Manuaalien kopioiminen (ja tiedostojen ylipäänsä) vaikuttaa paljon dokumenttikokoelman muodostusprosessin nopeuteen, ja joskus saatetaan vain haluta tietää mitä manuaaleja projektille kuuluu. Toiminto päätettiinkin määritellä valinnaksi ja asettaa se oletusarvoisesti päälle. Asetuksilla voidaan myös erotella monikieliset manuaalit eri hakemistoihin, jolloin manuaalikansion alle luodaan jokaiselle kielelle oma alihakemisto. Muutaman ryhmän projekteissa saatetaan taas kaivata selkeämpää hakemistorakennetta, jolloin linjakokoonpanohakemistoihin muodostettavat ryhmäkohtaiset alihakemistot voidaan jättää muodostamatta. Vaikutus on suurempi kuin ensi tuntumalta luulisi, koska myös ryhmähakemistoihin luodaan omat alihakemistot loppudokumenteille. Toinen merkittävästi muodostusprosessin nopeuteen vaikuttava asetus on mappidokumenttien muokkaustoiminto. Asetuksen ollessa valittuna tulee käyttäjän valita lisäksi, riippuen projektin koosta käytetäänkö sisällysluettelossa 12, 20 vai 31 otsikon listausta. Tämä jako perustuu ABB:llä käytössä oleviin sisällysluettelovälilehtiin.

Ohjelma pitää kirjaa kolmesta viimeisestä syötetystä tallennushakemistosta alasvetovalikossa, johon sijainti syötetään. Valikon sisältö päivitetään käyttäjän siirtyessä manuaalinäkymään projektinäkymän *Preview*-painikkeesta, mutta hakemisto luodaan tiedostojärjestelmään vasta generoinnin yhteydessä. Tallennushakemiston voi määrittää joko kirjoittamalla suoraan alasvetovalikkoon tai käyttämällä valikon *Browse...*-valintaa, joka avaa tiedostojärjestelmävalintaikkunan. Kaikki pakettiasetukset tallennetaan *Container*-säiliöluokkaan, kun siirrytään seuraavaan näkymään, jolloin ne ovat muiden luokkien käytettävissä.

7.5 XML-dokumenttien käsittely

7.5.1 Manuaalisääntötiedoston XML-muunnos

Työn aikana manuaalisääntötiedoston käsittelyssä seurasi ikävä yllätys: Excel-tiedoston lukemiseen vaadittava ActiveX-tekniikka ei ollut tuettuna Qt:n silloisessa Open Source Edition -laitoksessa. Dataan pääsemiseksi oli siis löydettävä uusi ratkaisu. Vaihtoehtoina olivat manuaalisääntötiedoston tallennus *CSV*-muotoon, XML-muunnos tai Windows API -komentojen käyttäminen. CSV-tekstitiedoston pilkuin ja rivinvaihdoin eroteltu taulukkorakenne vaikutti liian työläältä käsitellä ja Windows API -rajapinnan käyttö kyseiseen tarkoitukseen olisi vaatinut syvällisempää opiskelua. Ratkaisuna päädyttiin XML-muunnoksen tekevään VBA-makroon, joka suoritetaan aina tiedoston tallennuksen yhteydessä. XML-tiedoston käsittely ohjelmointikehyksen puolella oli helpoin ratkaisu, ja menetelmät olivat jo osittain tuttuja, sillä projektien XML-dokumenttien jäsentäjän toteutus oli aloitettu aikaisemmin.

Manuaalisääntötiedosto vaati muutamia lisäyksiä XML-muunnoksen helpottamiseksi. Sarakkeiden kuvauksia ei voitu suoraan käyttää elementteinä, koska osassa oli XML-standardin kieltämiä merkkejä. Elementeille ja ominaisuuksien plussakoodeille päätettiin luoda kokonaan uusi rivi puuttumatta dokumentointitiimin nimeämismenetelmiin. Entuudestaan plussakoodit sijaitsivat samassa solussa kuvaustensa kanssa, ja niiden erottelu ohjelmallisesti olisi teettänyt turhaa työtä. XML-muunnoksen tekevä VBA-makro on esitelty liitteessä 4.

7.5.2 Käsittelyprosessi

Valitun projektin ja manuaalisääntöjen XML-dokumentit käsitellään käyttäjän siirryttyä projektinäkymästä manuaalinäkymään. Käsittelijänä käytetään SAX-rajapinnan toteuttavaa *XMLParser*-luokkaa. Tapahtumapohjaisen SAX-tekniikan edut XML-dokumentin puurakenteen väliaikaismuistiin tallentavaan DOM-tekniikkaan käy ilmi isoissa tiedostoissa: SAX-käsittelijän muistivaatimukset ovat usein huomattavasti kevyemmät kuin vastaavan DOM-käsittelijän. Projektien XML-tiedostojen koot vaihtelevat sadoista

kilotavuista moniin megatavuihin, joten työn kannalta SAXin käyttö oli turvallinen vaihtoehto.

XMLParser-luokkaa käytetään sekä projektin että manuaalisääntöjen XML-dokumenttien jäsentämiseen. Kahden eri luokan kirjoittaminen tuntui tarpeettomalta, joten projektitiedoille alun perin kaavailtua jäsentäjäluokkaa päädyttiin vain laajentamaan kattamaan myös manuaalisääntöjen jäsennys. Juurielementin tarkistamisella selvitetään, mitä XML-dokumenttia kullakin hetkellä käsitellään (koodilistaus 4).

```
if (m_rootElement == "Project") // Project XML
{
    if (m_parentElement == "LineUps")
    {
        if (qName == "LineUpID")
        {
            // Create a new LineUp and insert it into the Container object
            LineUp* newLineUp = new LineUp(m_currentText);
            m_currentLineUp = newLineUp;
            m_container->insertLineUp(m_currentLineUp);
        }
        else
            m_currentLineUp->insertLineUpData(qName, m_currentText);
    }
```

Koodilistaus 4. Projektitietojen tallennus XMLParser-luokassa

SAX jäsentelee XML-dokumentin yksisuuntaisesti, eli aiemmin käsiteltyä dataa ei voida lukea uudelleen ilman koko jäsennysoperaation aloittamista alusta. Tästä ja XML-dokumenttien rakenteista johtuen osa käsitellystä tiedosta tallennetaan väliaikaisesti jäsenmuuttujiin, jotta niihin voidaan viitata myöhemmin. Projektin XML-tiedosto sisältää myös paljon ylimääräistä informaatiota, jota ei haluta tallentaa, joten on tärkeää, että elementtitarkistuksien avulla voidaan ohjata mitä tietoa tallennetaan.

Kaikki ohjelman linjakokoonpanoja, ryhmiä ja manuaaleja kuvaavat *LineUp*-, *Unit*- ja *Manual* -oliot luodaan ja määritellään käsittelyprosessin yhteydessä. Ne toimivat *Container*-olion tavoin tietovarastoina, joita käytetään hyväksi manuaali- ja esikatselunäkymien muodostamiseen. *Manual*- ja *LineUp*-oliot tallennetaan omiin säiliöihinsä *Container*-olion jäsenmuuttujiin. Vastaavasti *Unit*-oliot tallennetaan kyseisen ryhmän linja-

kokoonpanoa kuvaavan *LineUp*-olion säiliöön. Näin kaikilla luokilla on pääsy resursseihin *Container*-olion kautta.

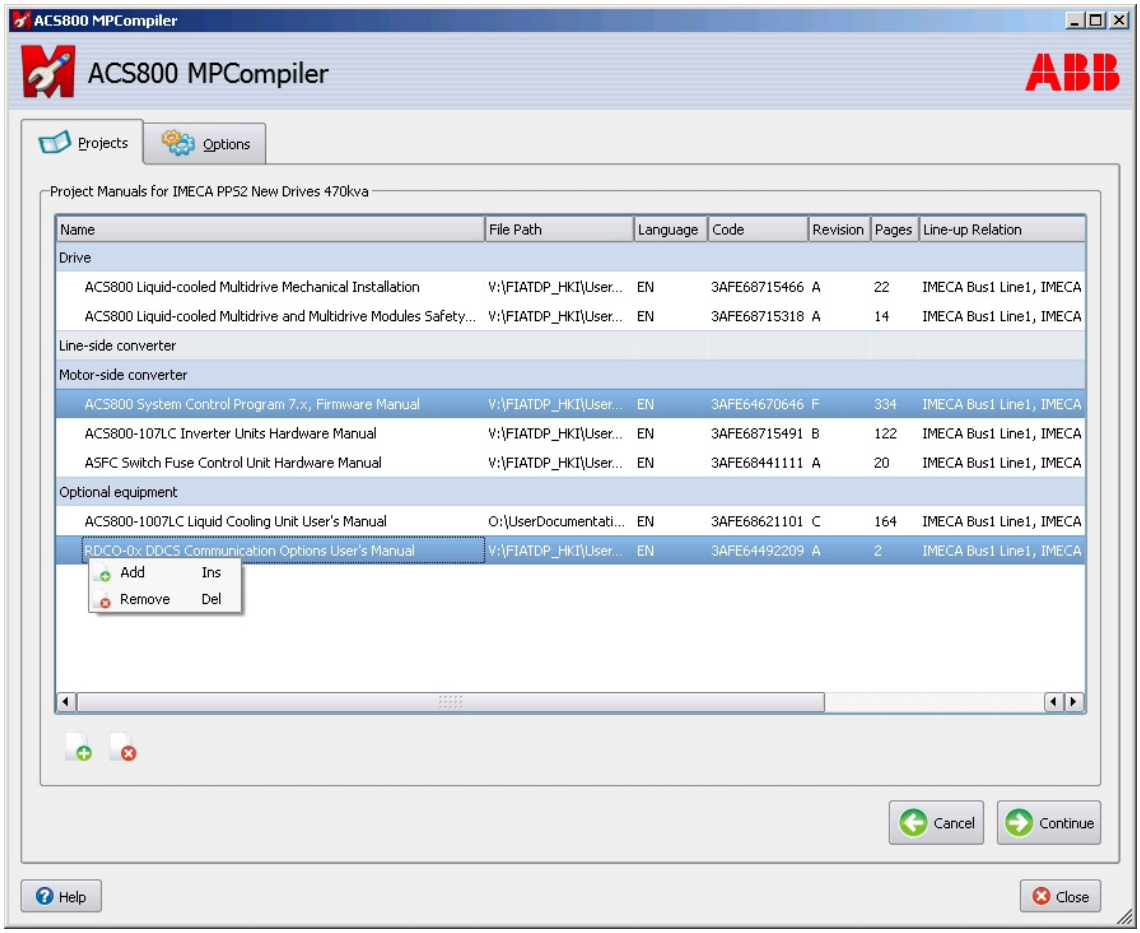
7.6 Projektimanuaalien kerääminen

Käytännössä *Container*-luokka sisältää manuaalisäiliöt sekä manuaalisääntötiedoston (kaikille manuaaleille) että projektin manuaaleille. Esikatselua varten lisättiin myös *LineUp*-luokalle manuaalisäiliö, jotta linjakokoonpanoille saatiin luotua omat manuaalihakemistot. Työn alkuvaiheessa myös *Unit*-luokalla oli oma manuaalisäiliönsä, mutta näin yksityiskohtainen tieto ei ollut esikatselun kannalta tarpeellista, joten ominaisuus poistettiin. *Unit*-luokkaa käytetään kuitenkin ryhmien plussakoodien muodostamiseen, joiden avulla kaikki projektimanuaalit kerätään.

Projektimanuaalit kerätään poimimalla *Container*-olion kaikkien manuaalien säiliöstä vain projektia koskevat manuaalit. Uusia *Manual*-olioita ei luoda erikseen projektille tai linjakokoonpanoille, vaan XML-jäsennyksessä luoduista olioista luovutetaan vain osoittimet *Container*- ja *LineUp*-olioiden säiliöihin. Prosessissa iteroidaan kaikki manuaalit jokaista linjakokoonpanon ryhmää kohti. Jos ryhmän plussakoodi sisältää manuaalin tuoteominaisuutta kuvaavan koodin, tarkastetaan vielä, löytyykö plussakoodista manuaalin kieltä kuvaava koodi. Tämä on välttämätöntä, koska tuotekonfiguraattorissa voidaan määritellä projektille vain yksi kieli, vaikka manuaalit itsessään voivat olla monikielisiä. Jos tuoteominaisuudet ja kielet täsmäävät, lisätään *Manual*-olion osoitin *Container*- ja *LineUp*-olioiden projektimanuaalisäiliöihin. Keräysprosessin suorittava *collectManuals*-funktio on esitetty liitteessä 5.

7.7 Projektimanuaalien tarkistaminen

Käyttäjän kannalta toinen vaihe dokumenttikokoelman luomisessa on projektimanuaalien tarkistaminen. Manuaalit listaavaa näkymää voidaan luonnehtia esikatselun alkuvaiheeksi, jossa käyttäjä voi lisätä uusia manuaaleja ja poistaa tarpeettomia. Kuvassa 14 on kuvakaappaus manuaalinäkymästä.



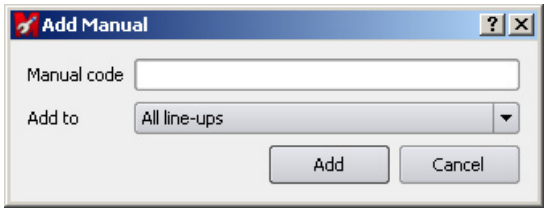
Kuva 14. Ohjelman manuaalinäkymä

Hahmottamisen helpottamiseksi manuaalit on järjestelty kategorioihin, joita voi piilottaa hiiren kaksoisnapautuksella tai eteen/taakse-nuolinäppäimillä. Manuaalilistan toteutuksessa käytettiin pohjana esikatselun model/view-arkkitehtuuria. *ManualModel*-luokka on yksinkertaistettu malli esikatselun *PreviewModel*-mallista, josta on poistettu muun muassa vedä ja pudota -toiminnallisuus. Vaikka mallien käyttötarkoitukset ja lähdetieto poikkeavat, molemmat käyttävät lähdetiedon varastoimiseen hierarkkisen tietorakenteen toteuttavan *TreeItem*-luokan olioita. *TreeItem*-luokan ohjelmakoodilistaus on koottu liitteeseen 6. Jokaista projektimanuaalia kuvaamaan luodaan uusi olio, johon tallennetaan dokumentin nimi, tiedostopolku, kieli, manuaalikoodi, revisio, sivulukumäärä, linjakokoonpanot, joiden ryhmiin manuaalit kuuluvat, ja olion käyttötarkoitusta kuvaava tyyppi. Tyyppiä käytetään vain *TreeItem*-olioiden tarkoituksien selventämiseen, joten sitä ei näytetä käyttäjälle. Manuaalidokumenttien kategorioille luodaan typistetymmät oliot, joissa vain nimi ja tyyppi ovat tallennettu.

Projektimanuaaleja voi lisätä tai poistaa käyttäen listan alapuolella sijaitsevan työkalupalkin painikkeita, hiiren oikealla painikkeella avautuvaa kontekstivalikkoa tai näppäinoikoteitä. Kategorioiden poistaminen on estetty. Tällöin kontekstivalikon toiminto sekä työkalupalkin ikoni muuttuvat harmaaksi kuvatakseen estettyä toimintoa. Kategoria poistetaan automaattisesti, jos sen alaisuuteen ei kuulu yhtään manuaalia.

Projektimanuaalin lisääminen

Manuaalidokumenttien lisäämiseen käytetään kuvassa 15 esitettyä valintaikkunaa. Valintaikkuna pyytää käyttäjältä lisättävän manuaalin manuaalikoodin, jota käytetään dokumenttia kuvaavan *Manual*-olion etsimiseen. Oletuksena manuaali lisätään kaikkiin projektin linjakokoonpanoihin, mutta tämän voi vaihtaa valitsemalla alasvetovalikosta halutun vaihtoehton.



Kuva 15. Manuaalien lisäämisessä käytetty valintaikkuna

Mikäli manuaalikoodia vastaavaa manuaalia ei löydy, näytetään käyttäjälle valintaikkunassa virheilmoitus. Virheilmoitus näytetään myös, jos manuaali on jo lisätty valittuun linjakokoonpanoon (tai kaikkiin). Kelvollisen manuaalikoodin tapauksessa luovutetaan *LineUp*- ja *Container*-olioille osoittimet koodin osoittamaan *Manual*-olioon, samalla tavoin kuin projektimanuaalien keräyksessä. Lisäksi oikean kategorian alaisuuten luodaan uusi *TreeItem*-olio kuvaamaan uutta projektimanuaalia. Mikäli kategoria puuttuu, luodaan sille ensin olio.

7.8 Dokumenttikokoelman esikatselu

7.8.1 Näkymän model/view-arkkitehtuuri

Ohjelman esikatselutoiminto oli yksi haastavimmista ja työläimmistä osa-alueista insinöörityön toteuttamisessa. Suurin syy tähän oli monipuolisen model/view-rajapinnan käyttöönotto ja arkkitehtuurin vaatima opiskelu, ja varmasti osaltaan vaikutti myös yleinen kokemuksen puute MVC-ohjelmoinnista. Rajapintatietämyksen kasvaessa toimintoon päätettiin myös lisätä uusia ominaisuuksia helpottamaan käytettävyyttä, kuten seuraavassa kappaleessa tarkemmin käsitelty vedä ja pudota -toiminnallisuus.

Esikatselunäkymän model/view-arkkitehtuurin malliluokkana toimii *PreviewModel*. Luokka käyttää lähdetietonaan *TreeItem*-olioita. Oliot luodaan vasta siirryttäessä manuaalinäkymästä esikatselunäkymään, jotta lisättyjen tai poistettujen manuaalien vaikutukset nähdään dokumenttikokoelmassa. *PreviewModel* rakentaa dokumenttikokoelman väliaikaismuistiin seuraavien perustietojen avulla: projektimanuaalit, mappidokumentit, linjakokoonpanoon kuuluvat manuaalit ja johtoryhmän kanssa dokumenttikokoelmalle määritelty hakemistorakenne. Tiedot saadaan *Container*-olion avulla haettua *Manual*-, *LineUp*- ja *Unit*-olioista. Jokaiselle yksittäiselle tietoalkiolle, kuten yhdelle manuaalille tai hakemistolle, luodaan oma *TreeItem*-olio, jolle alustamisen yhteydessä annetaan sen kuvaaman alkion tiedot. Prosessi etenee määritellyn hakemistorakenteen kannalta loogisessa järjestyksessä. Ensin luodaan päähakemisto manuaaleille ja tämän dokumentit, sitten siirrytään linjakokoonpanohakemistojen käsittelyyn. Ohjelmakoodilistaus dokumenttikokoelman väliaikaismuistiin rakentavasta *setupModelData*-funktiosta on liitteessä 7.

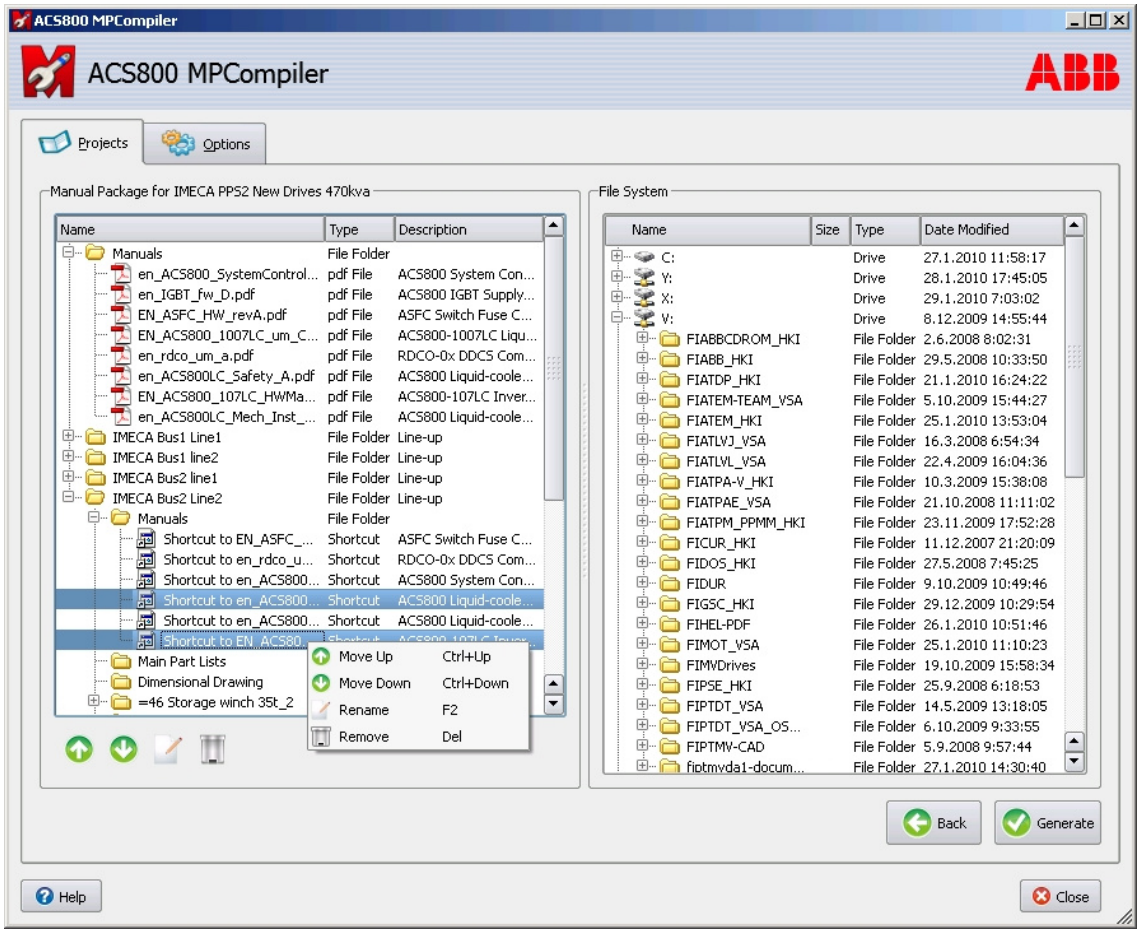
Esikatselun model/view-arkkitehtuurin näkymäluokkana toimii *PreviewView*, joka on peritty tiedon puumaisesti esittävästä *QTreeView*-luokasta. *PreviewView*-luokkaa käytetään dokumenttien portaittaisen järjestelytoiminnon toteutuksessa ja vedä ja pudota -ominaisuuden parantamisessa duplikaattien tarkistuksella. Näkymälle toteutettiin myös oma delegaatti. Qt:ssä kaikki näkymät käyttävät oletusdelegaattinaan *QStyledItemDelegate*-luokkaa, mutta *PreviewView*-näkymälle perittiin tämän delegaatin ominaisuudet monipuolisempaan *PreviewDelegate*-luokkaan. Tämä oli välttämätöntä esikatselun uu-

delleennimeämistoiminnon toteuttamiseksi. *ProjectInfoDelegate*- ja *ManualDelegate*-luokista poiketen, esikatselun delegaatilla ei muokata näkymän ulkoasua, koska tarvetta yksittäisten rivien erotteluun ei ollut. Esikatselunäkymässä tyylitiedoston tarjoamat ominaisuudet riittävät hyvin.

7.8.2 Esilaaditun kokoelman muokkaaminen

Viimeinen vaihe ennen dokumenttikokoelman generointia on paketin esikatselu ja loppudokumenttien lisääminen. Esikatselunäkymä jakautuu vasemmalla ja oikealla puolella sijaitseviin dokumenttipaketti- ja tiedostojärjestelmänäkymiin. Pakettinäkymä tulostaa kokoelman tiedostorakenteen (pois lukien indeksisivun lähdetiedostot), jota voi muokata lisäämällä, poistamalla, järjestelemällä tai uudelleen nimeämällä tiedostoja ja hakemistoja. Tiedostojärjestelmänäkymää käytetään loppudokumenttien lisäämiseen käyttäjän omalta tietokoneelta tai verkkolevyiltä. Halutut tiedostot vedetään dokumenttikokoelmaan hiirellä. Ominaisuutta voi hyödyntää myös pakettinäkymän sisällä rakenteen muokkaamiseen. Tiedostojärjestelmänäkymän kautta ei voi muokata tiedostojärjestelmää, jottei käyttäjä vahingossa poista omia tiedostojaan.

Dokumenttikokoelmalle määriteltiin oletusrakenne, joka luodaan aina esikatseluun siirryttäessä. Manuaalidokumentit sijaitsevat yhteisessä *Manuals*-juurihakemistossa ja loppudokumentteja varten jokaiselle linjakokoonpanolle on luotu oma hakemistopuunsa. Mikäli käyttäjä jättää manuaalidokumenttien kopioimisen valitsematta pakettiasetuksista, juureen tai linjakokoonpanoille ei luoda manuaalihakemistoja ollenkaan. Kuvassa 16 nähdään kuvakaappaus esikatselunäkymästä.



Kuva 16. Ohjelman esikatselunäkymä

Loppudokumenttien lisäämisen lisäksi dokumenttikokoelmaa voi muokata muullakin tavoin. Toimintoihin pääsee käsiksi valikkopalkista, kontekstivalikosta tai näppäinoikoteillä. Tiedostoja tai hakemistoja voi järjestellä dokumenttikokoelmassa joko portaittain *Move Up*- ja *Move Down*- toiminnoilla, tai hiirellä vetämällä ja pudottamalla. Portaittainen järjestely onnistuu vain saman tasoisten solmujen kesken, eli tiedostoja ei voi siirtää tällä tavoin pois hakemistosta. Uudelleen nimeämisessä tarkistetaan saman tason *TreeItem*-oliot, ja huomautetaan käyttäjälle virheilmoituksella mikäli nimi on jo varattu. Lisäksi nimi karsitaan epäkelvollisista erikoismerkeistä sekä palautetaan alkuperäinen tiedostotyyppi, jos sitä on yritetty vaihtaa. Tiedostoja voi poistaa *Remove*-toiminnolla, mutta hakemistojen täytyy olla tyhjiä. Tällä estetään harmittavien vahinkojen syntyminen.

Tiedostojen vetäminen ja pudottaminen

Esikatselun vedä ja pudota -toiminnallisuus oli osasyy näkymän haasteelliseen toteutukseen. Käytännössä hiirellä vedettyjen rivien (eli hakemistojen tai dokumenttien) tieto sarjallistetaan binäärimuotoiseksi *QDataStream*-vuohon, joka vastaavasti säilötään vedä ja pudota -toimen ajaksi *QMimeData*-luokkaan. *QMimeData* tarjoaa välineet välitettävän datan tyypin ilmaisemiseksi ja varmistaa sen turvallisen kopioimisen ohjelman sisällä sekä tarvittaessa toiseen ohjelmaan. *PreviewModel*- ja *FileSystemModel*-mallit sarjallistavat tiedon kuitenkin eri tavalla. Tämä johtuu siitä, että *PreviewModel* käsittelee suoraan *TreeItem*-olioita, kun taas *FileSystemModel* hakee lähdetietonsa tiedostojärjestelmästä. *TreeItem*-luokalle ylikuormitettiin sarjallistamisoperaattori **`operator<<`**-funktiolla (ks. liite 6), jolla yhden olion tiedot saadaan hallitusti kopioitua vuohon.

Molemmissa tapauksissa data sarjallistetaan rekursiivisesti tarkkaa järjestystä noudattaen. Järjestyksen ylläpitäminen on välttämätöntä, jotta takaisinlukuoperaatiossa datan eheys ei rikkoudu vääriin muuttujiin sijoitetuista arvoista. *QDataStream*-olioon tallennetaan ensin valitun rivin (tai monen rivin tapauksessa, ensimmäiseksi valitun rivin) lapsirivien lukumäärä, ja siirrytään näiden käsittelyyn. Vasta kun törmätään ensimmäiseen lapsettomaan riviin, tallennetaan kyseisen rivin omat tiedot vuohon (koodilistaus 5). Käytännössä tämä tarkoittaa sitä, että hakemiston tapauksessa ensin sarjallistetaan tämän alihakemistot ja tiedostot.

```
stream << (qint32)8 // Number of values the stream contains per item
<< (qint32)PreviewItem::Name << data(index, QFileSystemModel::FileNameRole)
<< (qint32)PreviewItem::FileType << QVariant::fromValue(fileType)
<< (qint32)PreviewItem::Description << QVariant::fromValue(QString(""))
<< (qint32)PreviewItem::SourcePath << QVariant::fromValue(filePath)
<< (qint32)PreviewItem::Path << QVariant::fromValue(QString(""))
<< (qint32)PreviewItem::ID << QVariant::fromValue(QString(""))
<< (qint32)PreviewItem::Icon << QVariant::fromValue(fileIcon(index))
<< (qint32)PreviewItem::ItemType << QVariant::fromValue(role);
```

Koodilistaus 5. Rivin tietojen sarjallistaminen FileSystemModel-luokassa

PreviewModel-luokalle kirjoitettiin *dropMimeData*-funktio, joka purkaa välitetyn *QMimeData*-säiliön ja luo uudet *TreeItem*-oliot vedettyjen rivien tiedoista. Lukuoperaatiossa ei haittaa kummasta näkymästä rivit on vedetty, koska *QDataStream*-vuo luetaan

yhtä järjestystä noudattaen. Lukujärjestys on yhdenmukainen tallennusjärjestyksen kanssa, eli käsiteltävän rivin tiedot luetaan rekursiivisesti vasta kaikkien sen lapsirivien jälkeen. *TreeItem*-oliot luodaan arkkitehtuuria mukailevista syistä dynaamiseen muistiin jo ennen kuin niiden data on tiedossa. Olioiden paikantaminen ja arvojen asettaminen onnistuu kuitenkin **`index`**- ja **`setData`**-funktioiden avulla. Kun tiedetään pudotushakemiston *QModelIndex*-tyypin indeksi (**`parent`**) ja tarkempi pudotuskohta hakemiston sisällä (**`row`**), voidaan kysyä, mitkä ovat hakemiston alle luodun *TreeItem*-olion indeksit sarake (**`role`**) kerrallaan. Indeksi välitetään parametrina tiedon asettavalle **`setData`**-funktiolle (koodilistaus 6).

```
qint32 dataCount, role;
QVariant itemData;
stream >> dataCount;
for (int i = 0; i < dataCount; i++)
{
    stream >> role >> itemData;
    itemIndex = index(row, role, parent);
    setData(itemIndex, itemData);
}
```

Koodilistaus 6. TreeItem-olion tietojen asettaminen vuosta PreviewModel-luokassa

Vedä ja pudota -toteutusta hyödynnettiin myös tiedostojen portaittaisessa järjestelytoiminnossa. Siirtoon valittujen rivien indeksien luovuttamisesta *PreviewModel*-luokalle vastaa *PreviewView*-näkymäluokka. *PreviewView* huolehtii myös rivien rajatarkastuksista ja valinta-alueen päivittämisestä. Varsinainen *TreeItem*-olioiden kopiointiprosessi suoritetaan kuitenkin malliluokan avulla (koodilistaus 7).

```
QMimeData* data = model()->mimeData(rowIndexes);
model()->removeRow(row, parent);
model()->dropMimeData(data, Qt::MoveAction, row + offset, 0, parent);
```

Koodilistaus 7. Yhden rivin siirtäminen PreviewView-luokassa

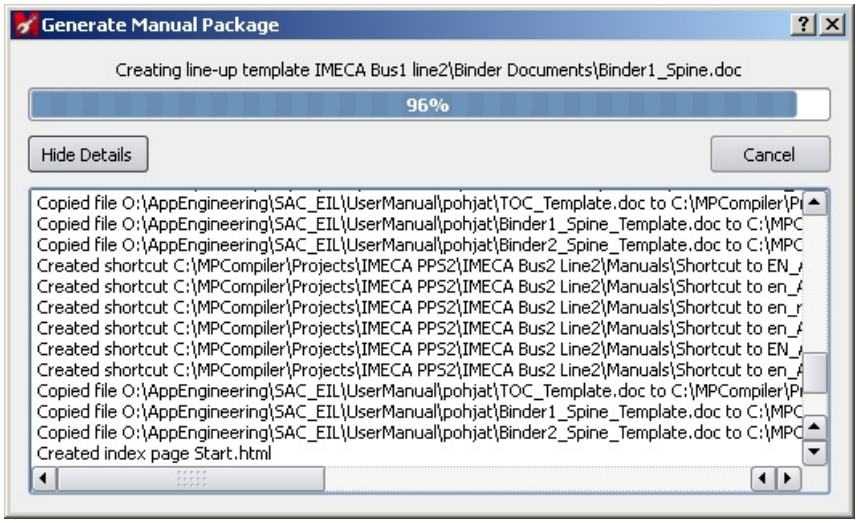
Tiedostojen järjestelytoiminnolla ei luonnollisesti pystytä vaikuttamaan käyttöjärjestelmän tiedostolistaukseen, koska käyttöjärjestelmä ylläpitää omaa järjestystä käyttäjän valinnan mukaisesti. Generoitavan HTML-indeksisivun sisältöön oli ohjelmalla kuitenkin täydet valtuudet, mikä oli vaatimusmäärittelyn kannalta riittävää. Loppudokumenttien

muutenkin epämääräisiin tiedostonimiin ei haluttu lisätä numeroituja etuliitteitä, joilla olisi voitu vaikuttaa tiedostojärjestelmän listaukseen, joten järjestelytoiminnon seuraamukset päätettiin rajata indeksisivulle.

7.9 Dokumenttikokoelman generointi

7.9.1 Säikeistetty muodostusprosessi

Dokumenttikokoelman generointi aloitetaan esikatselunäkymän *Generate*-painikkeesta. Tällöin avautuu kuvassa 17 esitetty valintaikkuna, jossa prosessin kulusta ilmoitetaan latauspalkilla. Käyttäjä voi myös halutessaan avata näkyviin yksityiskohtaisemmat tiedot muodostusprosessin vaiheista.



Kuva 17. Dokumenttikokoelman generoinnin tilasta viestittävä valintaikkuna

Virhetilanteissa generointi pyritään suorittamaan loppuun, mutta virheeseen johtanut vaihe on listauksessa eroteltu punaisella värillä. Tällä halutaan varmistaa, että yhden manuaalin puuttuessa koko prosessi ei epäonnistu. Mahdollisista virheistä ilmoitetaan myös generoinnin loputtua. Samalla valintaikkunassa näytetään painike, joka avaa käyttäjälle dokumenttikokoelman juurihakemiston resurssienhallintanäkymässä. Sovellusasetuksien mukaan, valintaikkunan suljettua siirrytään joko takaisin projektinäkymään tai jäädään esikatseluun.

Esikatsellun dokumenttipaketin generointi suoritetaan uuden säikeen käynnistävässä *GenerationThread*-luokassa. Väliaikaisen työskentelijäsäikeen (worker thread) käyttämisellä saadaan käyttöliittymä pysymään aktiivisena ja vastattua käyttäjän syötteisiin muodostusprosessin ajan. Luokka paisui toteutuksen aikana yhdeksi suurimmista luokista, jolloin ylläpitäminen osoittautui luonnollisesti hankalammaksi. Suurimmat syyt laajuuteen olivat indeksisivun ja mappidokumenttien muokkaamiseen kirjoitetut ohjelmakoodit, jotka ovat myös oleellisia generoinnin aikaansaannoksia. *GenerationThread*-luokan elinajan määrittävän *run*-funktion toteutus on liitteessä 8.

Kommunikointi käyttöliittymäsäikeen kanssa

`GenerationThread`-säie kommunikoi **`ProgressDialog`**-valintaikkunan kanssa käyttäen signal-slot-yhteyksiä. Jokaisesta käyttäjälle viestitettävästä asiasta on erikseen lähetettävä signaali, jotta käyttöliittymäsäikeen valintaikkuna osaa päivittää tietonsa vastaamaan generoinnin tilaa. Signaalien käyttötarkoitukset vaihtelevat latauspalkin maksimirajan asettamisesta ja päivittämisestä yksityiskohtaisempiin yksittäisistä vaiheista tiedottamiseen (koodilistaus 8).

```
MainWindow:
  connect(generationThread, SIGNAL(addDetail(QString)),
          progressDialog, SLOT(addDetail(QString)));
GenerationThread:
  emit addDetail(QString("Created folder %1").arg(currentPath));
ProgressDialog:
  void ProgressDialog::addDetail(const QString& detail)
  {
      m_detailsBrowser->append(QDir::toNativeSeparators(detail));
  }
```

Koodilistaus 8. Esimerkki työskentelijä- ja käyttöliittymäsäikeiden välisestä kommunikoinnista

Jos käyttäjä on valinnut sovellusasetuksista *Show prompt to overwrite existing files* -asetuksen (ks. kuva 9), *GenerationThread*-säie lähettää signaalin käyttöliittymäsäikeelle, kun kopioimisessa yritetään kirjoittaa olemassa olevan tiedoston päälle. Tällöin *MainWindow*-luokassa luodaan uusi valintaikkuna, jolla kysytään käyttäjältä mitä tehdään. Vastaus lähetetään takaisin työskentelijäsäikeelle, joka jatkaa toimintaansa ja päättelee tuloksen perusteella jatkotoimet.

Tiedostojen kopiointi ja oikotiet

Ennen dokumenttikokoelmaan lisättyjen tiedostojen kopioimisen aloittamista täytyy jokaiselle *TreeItem*-oliolle laskea hakemistopolku suhteessa juurihakemistoon. Tämä on välttämätöntä, jotta tiedetään, mihin olion kuvaama tiedosto tai hakemisto tulee tallennuskohteessa kopioida tai luoda. Toimintoa ei suoritettu aikaisemmin dokumenttikokoelman muokkaamisen yhteydessä, koska riittää, että se tehdään vain kerran. Lisäksi on järkevämpää, että kaikki laskenta suoritetaan generoinnin aikana, jolloin valintaikkuna ja latauspalkki ovat esillä. Hakemistopolkuja hyödynnetään myös muissa toiminnoissa generoinnin aikana, kuten manuaalidokumenttien oikotietiedostojen ja indeksisivun hyperlinkkien luonnissa.

Dokumenttikokoelman luominen tiedostojärjestelmään aloitetaan *PreviewModel*-mallin *TreeItem*-juurisolmusta. Koska juurisolmu kuvaa poikkeuksellisesti näkymän otsikkopalkkia, ensimmäinen käsiteltävä *TreeItem*-olio on ylimmäiseksi järjestelty rivi. Prosessi tarkastaa olion tyypin, jonka perusteella tallennuskohteeseen luodaan joko uusi tiedostokopio, oikotietiedosto tai hakemisto. Hakemiston tapauksessa käsitellään myös kaikki lapsisolmut, jotta käsittelyjärjestys säilyy johdonmukaisena. Oikotietiedostot luodaan vain kaikille manuaalidokumenteille. Oikoteiden käyttöön päädyttiin käytännöllisistä syistä, sillä manuaalien kopiointi useaan paikkaan eri linjakokoonpanoille olisi turhaan kasvattanut dokumenttipaketin kokoa.

Oikotietiedostoille toteutettiin kaksi luontimenetelmää kokoelman käyttötarkoituksia mukaillen. Menetelmiä pystyy vaihtamaan asetusnäkymästä. Yksinkertaisempi menetelmä luo normaalin ja monelle käyttäjälle tutumman Windows-käyttöjärjestelmän lnk-tyypin oikotietiedoston, jossa kohdehakemistoon viitataan absoluuttisella hakemistopolulla. Absoluuttinen hakemistopolku alkaa aina levyn juuresta ja sisältää kohteen kaikki hakemistot. Windows-oikoteiden huono puoli on se, että ne muuttuvat heti kelvottomiksi, jos kohteen hakemistopolkua muokataan. Toimitettavan dokumenttikokoelman tapauksessa absoluuttisten hakemistopolkujen käyttäminen ei luonnollisesti olisi järkevää, joten menetelmän rinnalle toteutettiin suhteellisia hakemistopolkuja käyttävät oikotietiedostot. Suhteellinen hakemistopolku ei ilmaise kohteen koko polkua, vaan ainoastaan

kohdepolun eroavaisuudet oikotiestä. Dokumenttikokoelman siirtäminen ei siis vaurioita suhteellisia oikotietiedostoja, kunhan kokoelman sisäistä rakennetta ei muokata.

```
QFile batFile(currentPath);
if (batFile.open(QIODevice::WriteOnly))
{
    QTextStream out(&batFile);
    out << "@ECHO OFF" << endl;
    out << "START " << sourcePath << endl;
    out << "EXIT" << endl;
}
```

Koodilistaus 9. Manuaalidokumentin avaavan bat-tiedoston luominen

Suhteellinen oikotietiedosto luodaan Windowsin bat-komentojonotiedostoksi (koodilistaus 9). Tiedoston merkkipohjainen sisältö on varsin yksinkertainen, manuaalidokumentti avataan *START*-komennolla suoraan suhteellista polkua käyttäen. Lopputuloksena käyttäjä ei huomaa eroa normaalista Windows-oikotiestä kuin muuttuuneena tiedostoikonina ja hetken aikaa näkyvästä MS-DOS:n komentoikkunasta.

7.9.2 Indeksisivun muodostaminen ja rakenne

Indeksisivun muodostamisessa käytetään pohjana HTML-tiedostoa, joka on esitetty liitteessä 9. Ainoa dokumentti, jota indeksisivun muodostamisessa muokataan, on tämä HTML-mallidokumentti. Mallidokumentti ja sen käyttämät JavaScript-, CSS- ja kuvatiedostot tallennetaan ohjelman kääntämisen yhteydessä qrc-resurssitiedostoon, josta ne dokumenttikokoelman generoinnin aikana kopioidaan takaisin tiedostojärjestelmään. Tällä kertaa XML-muotoisen HTML-pohjadokumentin käsittelyyn käytetään DOM-tekniikkaa noudattavaa ja yhtä XML-dokumenttia edustavaa *QDomDocument*-luokkaa. Luokan oliolle annetaan käsiteltävä tiedosto *setContent*-funktiolla, jonka jälkeen tiedoston sisältöä voidaan vapaasti muokata Qt:n XML-moduulin muita luokkia käyttämällä. Kuvakaappaukset lopullisesta muokatusta indeksisivusta nähdään liitteessä 10.

Indeksisivu koostuu yhdestä HTML-tiedostosta, mutta näkymiä on useita. Sivu sisältää animaatioita navigoinnin tehostamiseksi, muun muassa välilehdeltä toiseen siirryttäessä ja linjakokoonpanon sisältöä selattaessa. jQuery-kirjasto osoittautui erinomaiseksi työ-

kaluksi navigoinnin toteuttamisessa. Animaatioiden lisäksi jQueryä käytetään HTML-elementtien piilottamiseen välilehtiä vaihdettaessa, jotta yhden HTML-tiedoston indeksisivu oli ylipäätään mahdollista. Ratkaisu vaikeutti osaltaan toteutusta, mutta indeksisivuun liittyvien tiedostojen määrä haluttiin pitää mahdollisimman pienenä. Nyt dokumenttikokoelman juurihakemisto sisältää vain yhden *Start.html*-nimisen tiedoston, jonka käyttötarkoitus on helposti ymmärrettävissä.

Indeksisivu on eroteltu välilehdittäin projektitietoihin, manuaalilistaukseen ja linjakokoonpanohakemistoihin. *Projects*-välilehdellä kerrotaan lyhyesti projektin asiakas, nimi ja kauppanumero. *Manuals*-välilehti sisältää listauksen projektimanuaaleista, noudattaen mahdollisimman yhdenmukaista ulkoasua ohjelman manuaalinäkymän kanssa. Manuaalidokumentit on jaoteltu kategorioittain, ja ne sisältävät linkin vastaavaan dokumenttiin, mikäli pakettiasetuksista valittiin manuaalit kopioitaviksi. *Line-ups*-välilehdeltä käyttäjä voi selailla loppudokumentteja. Tarkasteltava linjakokoonpano valitaan vasemman puoleisesta listasta, jolloin tämän hakemistorakenne esitetään vierellä omassa listassaan. Listan rakenne määräytyy suoraan esikatselun linjakokoonpanojuurihakemiston mukaan, ja se toteutettiin HTML:ssä pinoamalla päällekkäin järjestelemättömän luettelon ul-tageja. Tilan säästämiseksi kaikki ensimmäisen tason listan alkiot ovat avattavia ja suljettavia. Käytännössä elementeille annettiin tunnukset, ja käyttäjän napsauttaessa alkiota (eli hakemistoa) hiirellä kyseisen alkion lapsiluettelot (eli alihakemistot ja dokumentit) joko näytetään tai piilotetaan. Kaikki tiedostot esitetään linkkeinä.

Indeksisivulle kirjoitettiin myös erillinen tulostusasettelun määrittävä CSS-tyylitiedosto, jolla projektitiedot ja manuaalilistauksen saa tulostettua. Loppudokumenttien jaottelukohtainen tulostustoiminto jäi kuitenkin toteuttamatta. Tämä määritelty vaatimus päätettiin siirtää jatkokehitykseen, jotta ohjelman julkaisu ei viivästyisi liikaa.

7.9.3 Paperimappien mallidokumenttien muokkaaminen

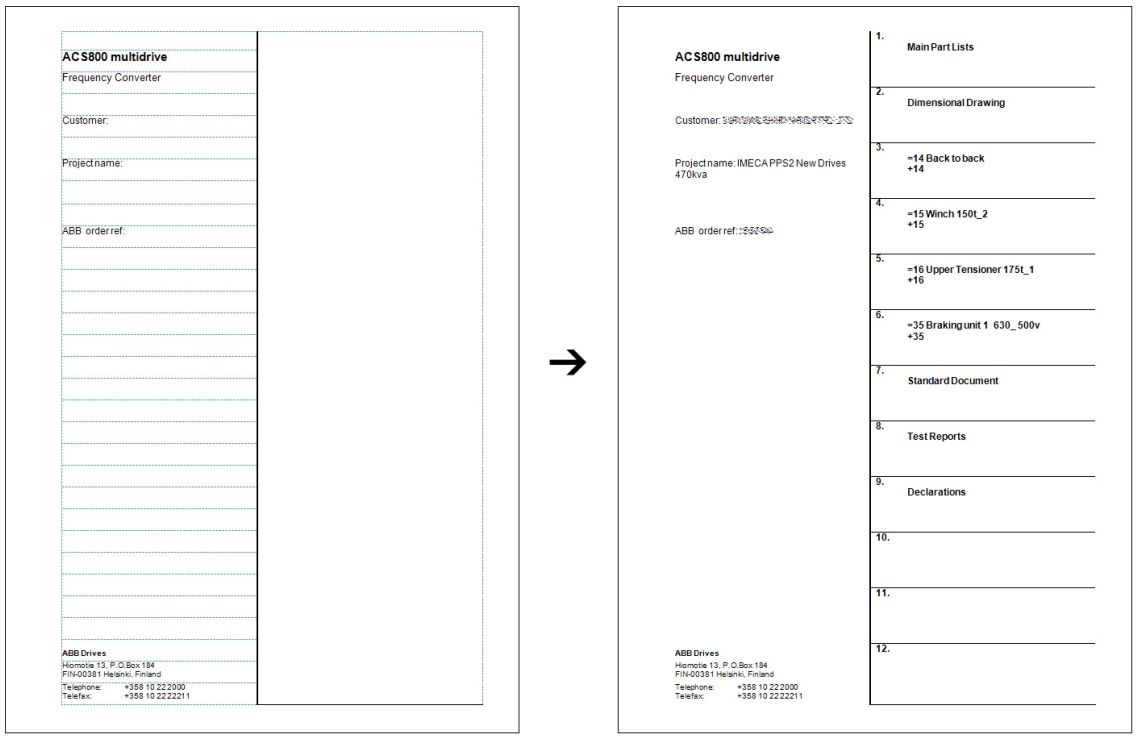
Jokaiselle linjakokoonpanolle luodaan kahta paperimappia varten päätylehdet ja yhtenäinen sisällysluettelo, mikäli pakettiasetuksissa niin on määritelty. Malleina käytetään valmiiksi muotoiltuja Microsoft Word -dokumentteja. Qt ei työn tekohetkellä tarjonnut

Open Source Edition -laitoksessa mahdollisuutta Word-dokumenttien muokkaamiseen omien kirjastoluokkiensa avulla, joten helpoin tapa toteuttaa muokkaaminen oli kirjoittaa malleille VBA-makrot, kutsua niitä Windows API -komennoilla ja välittää halutut tiedot parametreilla. Toinen vaihtoehto olisi ollut hoitaa prosessi kokonaan Windows API -rajapinnan avulla, mutta tämä olisi vaatinut huomattavasti syvällisempää opiskelua. Word-dokumenttien muokkaaminen makroilla on kuitenkin hankalampaa kuin Excel-dokumenttien, joissa tiedot sijaitsevat helposti eroteltavissa soluissa. Onneksi Word-dokumenteissakin voi luoda taulukoita, joiden soluihin makrokoodin puolella pääsee käsiksi. Tämän johdosta kaikki malleihin lisättävä projektitieto, kuten sisällysluettelossa projektin asiakas, nimi, kauppanumero ja linjakokoonpanon ryhmiä edustavat otsikot, tallennetaan taulukkoihin. Taulukot on määritelty reunattomiksi, joten paperilla niiden käyttöä ei huomaa.

VBA-makroja kutsutaan *GenerationThread*-luokasta viimeisenä vaiheena generointiprosessissa. Ennen kuin Word-dokumentteihin pääsee käsiksi, täytyy luoda yhteinen Microsoft Word -instanssi, joka ylläpitää tietoa kaikista avatuista dokumenteista. Tämän jälkeen voidaan jokainen mallidokumentti avata erikseen, konvertoida data Windows API -yhteensopivaksi ja päätylehtien tapauksissa kutsua suoraan makroja. Sisällysluettelon käsittely on monimutkaisempi prosessi otsikkolistauksen takia. Otsikkolistaukseen kuuluvat linjakokoonpanon ryhmät ja loppudokumenttihakemistot erotellaan *TreeItem*-kohtaisesti *isBinderTitle*-funktion avulla. Lisäksi täytyy laskea, kuinka monta sivukopiota tarvitaan, jos luotavien otsikoiden lukumäärä ylittää pakettiasetuksissa valitun sisällysluettelon otsikkojaon (12, 20 tai 31). *GenerationThread*-luokalle kirjoitettu mallidokumenttien muokkaamisesta huolehtiva pääfunktio on esitetty liitteessä 11.

Päätylehtimallit sisältävät vain yhden makron, joka kirjoittaa dokumenttiin asiakkaan ja projektin nimet, kauppanumeron sekä linjakokoonpanon position eli sen kauppaa varten kirjatun järjestyksen lopullisessa kokoonpanossa. Sisällysluettelomalli sisältää kolme makroa, joissa projektitietojen syöttämisen lisäksi luodaan uusi taulukko otsikkolistaukselle, lisätään yksittäinen otsikko listaukseen ja luodaan tarvittava määrä indeksointia jatkavia sivukopioita, mikäli listaus ei mahdu yhdelle sivulle. Kuvassa 18 on esitetty

sisällysluettelon mallidokumentti ja tämän pohjalta yhdelle linjakokoonpanolle luotu paperimapin sisällysluettelo.



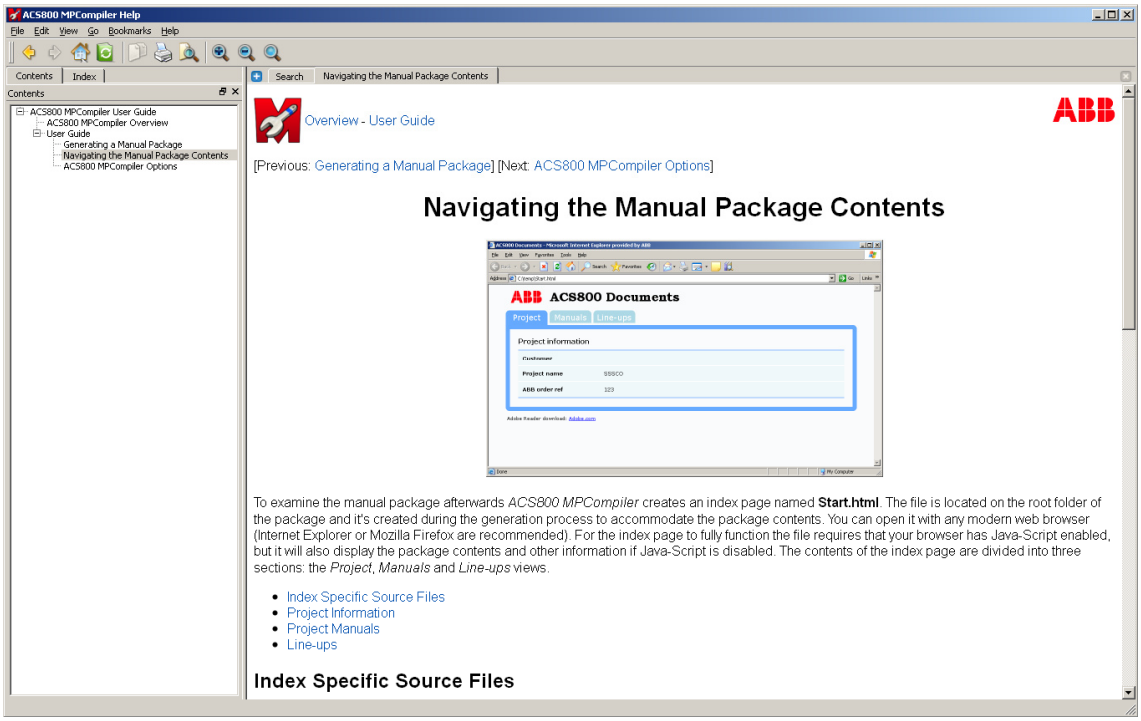
Kuva 18. Sisällysluettelon malli (taulukon solut näkyvissä) ja muokattu lopullinen dokumentti

Sisällysluettelon mallidokumentti sisältää yhden taulukon, jonka toisen sarakkeen rivit on yhdistetty yhdeksi soluksi. Tähän tyhjään soluun muodostetaan otsikkolistaus uuteen taulukkoon. Tämä rajoittaa käyttäjän mahdollisuuksia muotoilla koko mallia, mutta dynaaminen otsikkolistaus olisi muuten liian hankala toteuttaa.

7.10 Käyttöohjeet

ACS800 MPCompiler -ohjelmaan kirjoitettiin käyttöohjeet. Ne auttavat käyttäjää muodostamaan dokumenttikokoelman, selaamaan indeksisivua ja selvittämään asetuksien ja resurssien tehtävät. Käyttöohjeet saa avattua pääikkunan *Help*-painikkeesta. Tällöin käynnistetään uusi prosessi samalle Qt Assistant -selainohjelmalle, jota käytetään myös ohjelmointikehyksen dokumentaation esittämiseen. Ohjelmaa saa käyttää oman doku-

mentaation näyttämiseen, ja sovitukseen löytyy myös kattavat ohjeet [19]. Edellytyksenä on muun muassa se, että Qt-dokumentaatio on poistettu. Qt Assistantin käyttöönotto säästi huomattavasti aikaa, kun sai rauhassa keskittyä käyttöohjeiden kirjoittamiseen eikä tarvinnut toteuttaa omaa ratkaisua ohjeiden esittämiseen. Kuvassa 19 nähdään osa ACS800 MPCompilerin käyttöohjeista tulostettuna käyttäjälle.



Kuva 19. Katkelma ACS800 MPCompilerin käyttöohjeista

Qt Assistant toimii kuten WWW-selain. *Back*- ja *Forward*-painikkeilla voi vaihdella vierailtuja sivuja ja linkkiä napsauttamalla pääsee kyseiseen kohtaan dokumentaatiossa. Lisäksi ohjelma tarjoaa hakutoiminnon, ohjelmoijan määrittämän sisällysluettelon, mahdollisuuden tallentaa kirjanmerkkejä ja dokumentaation avaamisen useisiin välilehtiin. Qt Assistant käyttää syötteenään ohjekokoelmatiedostoja, jotka viittaavat dokumentaatiosta käännettyihin binäärimuotoisiin qch-ohjetiedostoihin. Tekniikka mahdollistaa usean manuaalin samanaikaisen käyttämisen, mutta ACS800 MPCompilerin tapauksessa riitti yhden ohjetiedoston ja ohjekokoelman käyttö.

Käyttöohjeet kirjoitetaan HTML-tiedostoihin. Dokumentaation esitystapaa voi lisäksi täydentää CSS-tyylitiedostoilla, minkä vuoksi jokainen käytössä oleva ohjekokoelma

voi näyttää erilaiselta. Tämä on yksi Qt Assistantin vahvuuksista, koska oman dokumentaation voi räätälöidä hyvin ohjelman tyyliin sopivaksi, eikä tarvitse välittää ristiriitaisuuksista muiden mahdollisten kokoelmien kanssa. Käytettävä ohjekokoelma voidaan vaihtaa asetuksista tai välittää komentoriviparametrina Qt Assistantin käynnistyksen yhteydessä.

7.11 Testaus ja käyttöönotto

Ohjelmaa varten ei kirjoitettu testaussuunnitelmaa. Uusia toimintoja testattiin aina toteutuksen yhteydessä ohjelmointikehyksen tarjoamilla työkaluilla. Yhdeksi tärkeimmistä osoittautui *qDebug*-funktio, joka Windows-alustalla tulostaa sille annetut parametrit konsolissa. Funktiota voi käyttää esimerkiksi osoitinmuuttujien oikeellisuuden tarkistamiseen tai säiliön sisällön tulostamiseen. Qt Creator -kehitysympäristö sisältää myös *vianjäljitysohjelman*, jolla lähdekoodin toimintaa voi tarkastella pala palalta, mutta tämän käyttö jäi vähemmälle johtuen sen ongelmista kotikoneessani olevan 64-bittisen Windows Vista -käyttöjärjestelmän kanssa.

Työn aikana testasin toimintoja ja käyttöä pääasiassa itse. Välillä ohjelmasta käännettiin testiversio sovellussuunnitteluosastolle, jossa arvioitiin, miten hallintatyökalu soveltuu käyttötarkoitukseensa. Arvioitavia asioita olivat muun muassa esikatselu ja loppudokumenttien lisääminen sekä indeksisivun tarkoituksenmukaisuus. ACS800 MPCompilerin versio 1.0, joka sisälsi kaiken määritellyn toiminnallisuuden, oli ensimmäinen versio, jota sovellussuunnittelu testasi kunnolla muodostamalla oikeista tapauksista dokumenttikokoelmia ja käyttämällä ohjelmaa työtehtävien hoidossa. Hallintatyökalu julkaistiin 15.10.2009 sovellussuunnittelun kokeiltavaksi ja siirtyi tästä vakituiseen käyttöön. Julkaisu suoritettiin exe-muotoisella asennustiedostolla, jonka muodostamiseen käytettiin ilmaista Inno Setup -asennusohjelmaa. ACS800 MPCompilerin käyttöä ohjeistettiin kuukausittaisissa seurantapalavereissa aina uusien toimintojen esittelyiden yhteydessä, joten erillisen koulutuksen järjestämiseen ei julkaisuhetkellä ollut tarvetta. Lisäksi käyttäjien vähäisyydestä johtuen jatkokehiteltävien toimintojen ohjeistaminen onnistuu myös vaivattomasti kasvokkain tai palavereissa. Tarkemmat kokemukset ja palautteet ohjelman käytöstä kerrotaan luvussa 9.

8 Jatkokehityssuunnitelma

ACS800 MPCompiler -hallintatyökalua on tarkoitus laajentaa useilla ominaisuuksilla, jotka rajattiin pois ohjelman julkaisuversiosta. Osa näistä tuli esille jo vaatimusmäärittelyssä, kun tutkittiin CDmacron ominaisuuksia, mutta niiden arvioitiin olevan liian laajoja toteutusalueita liitettäväksi julkaisuun. Loput ovat työn aikana esille tulleita ajatuksia hyödyllisistä toiminnallisuuksista. Näiden lisääminen julkaisuversioon ei kuitenkaan ollut aikataulun huomioon ottaen tarpeeksi perusteltua.

Tuki kaikille single drive -tuotteille

ACS800 MPCompiler käyttää ACS800 Product Configurator -järjestelmän projektitietoa lähdemateriaalinaan, mikä kattaa myös nestejäähdytteiset single drive -tuotteet. Ohjelmalla pystyy siis muodostamaan dokumenttikokoelmia myös näistä single drive -tuotteista, koska tuotekonfiguraattori tallentaa sekä multidrive- että single drive -projektit samassa muodossa XML-dokumentteihin. Hallintatyökalu on kuitenkin suunniteltu ja toteutettu multidrive-tuotteita ajatellen, mikä näkyy hakemistorakenteen, indeksisivun ja mappidokumenttien linjakokoonpanoajattelussa. Lisäksi laajempaa ilmajäähdytteistä single drive -tuotekategoriaa ei huomioida ollenkaan. Tuen lisääminen kaikille single drive -tuotteille on yksi tärkeistä ominaisuuksista, joilla CDmacron käyttö saadaan kokonaan korvattua.

Ilmajäähdytteisille single drive -tuotteille voitaisiin lisätä käyttöliittymään CDmacron kaltainen tuotteiden valintanäkymä, joka toimisi nykyisen projektilistauksen vierellä. Nestejäähdytteisten single drive -projektien vaikutukset generoitavaan dokumenttikokoelmaan tulisi myös tarkistaa ja korjata tuoteperhettä mukailevaksi.

Tuotekohtaisten erikoismanuaalikokoelmien muodostaminen

Toinen tarvittavista CDmacron korvaavista ominaisuuksista on tuotekohtaisten single drive -erikoismanuaalikokoelmien muodostaminen. Manuaalikokoelmia on vain muutamaa eri tyyppiä ja ne löytyvät listattuna manuaalisääntötiedostosta. Toistaiseksi näille ei

tosin ole määritelty minkäänlaisia plussakoodeja, eikä ACS800-tuotekonfiguraattori sisällä näihin liittyvää tietoa. Hallintatyökalun käyttöön voitaisiin kuitenkin määrittää omat plussakoodit, jotka sitoisivat käyttäjän valitseman kokoelman manuaaleihin.

Muut jatkokehiteltävät toiminnallisuudet

Jatkokehiteltävät ominaisuudet vaihtelevat suoraviivaisista asetuksista kokonaan uusien palveluiden hyödyntämiseen. Seuraavat ominaisuudet kuuluvat kaikki osaksi jatkokehityssuunnitelmaa:

- dokumenttikokoelman automaattinen pakkaaminen zip-tiedostoksi
- projektitiedon asettaman dokumentointikielen vaihtaminen
- loppudokumenttien PDF-muunnos
- projektimanuaalien listakohtainen lisääminen
- loppudokumenttien tulostaminen
- ulkoisten palveluiden (Microsoft SharePoint) hyödyntäminen uusien julkaisutapojen toteuttamisessa tai manuaalisääntöjen hakemisessa.

Manuaalien lisäys- ja loppudokumenttien tulostustoiminnot on raporttia kirjoitettaessa ehditty jo toteuttaa toimivina, tosin karsittuina testiversioina. Projektimanuaalien lisäämisessä näkymän valintaikkuna on korvattu uudella manuaalilistauksen sisältävällä valintaikkunalla. Manuaalit lisätään edelleen linjakokoonpanokohtaisesti, mutta nyt kaikkien tiedot ovat näkyvillä ja usean manuaalin lisääminen samanaikaisesti on mahdollista. Loppudokumenttien tulostus noudattaa linjakokoonpanolle määrättyä tiedostojärjestystä, mutta koskee vain kaikkia PDF-dokumentteja. Toiminto suoritetaan ulkoisella bat-tiedostolla, joka luodaan jokaiselle linjakokoonpanohakemistolle erikseen.

Kaikkia suunniteltuja ja esiteltyjä toiminnallisuuksia tullaan tuskin ikinä toteuttamaan. Jatkokehityssuunnitelman yhtenä tärkeänä päämääränä onkin selvittää, mitkä harkinta-asteella olevista ominaisuuksista ovat toteutuskelpoisia ja riittävän oleellisia ohjelman toiminnan kannalta.

9 Palaute ja tuloksen arviointi

Palaute ACS800 MPCompiler -hallintatyökalusta on ollut positiivista. Ohjelman käyttöliittymää on kiitetty selkeäksi ja dokumenttikokoelman muodostusprosessia johdonmukaiseksi ja vaivattomaksi. Projektitiedon käyttämisellä saavutettavaa automatiikkaa on selvästi arvostettu. Mallidokumenttien muokkaustoiminto on myös osoittautunut toimivaksi ja aikaa säästäväksi ratkaisuksi, kun informaatio on valmiina ja oikeassa muodossa. ABB:n asiakkailta kysyttiin palautetta uudesta dokumenttikokoelmasta toimittamalla sekä CDmacrolla että ACS800 MPCompilerilla luodut dokumenttikokoelmat. ACS800 MPCompilerin dokumenttikokoelmaa pidettiin parempana, koska sen linjakokoonpano- ja ryhmäkohtainen hakemistorakenne muistuttaa läheisemmin paperimappeja. Asiakkaat kertoivat myös, että heidän on uuden hakemistorakenteen pohjalta helpompi tulostaa materiaali paperimappeihin, mikäli alkuperäiset kopiot katoavat.

Positiivisesta palautteesta huolimatta esille on noussut myös muutamia ongelmakohtia liittyen ACS800-tuotekonfiguraattorin projektitiedon käyttämiseen. Projektitieto ei nimittäin aina pidä täysin paikkaansa. Joskus projekteihin on välttämätöntä tehdä muutoksia jälkeenpäin. Ne eivät näy tuotekonfiguraattorissa, jos työntekijä ei niitä sinne erikseen muista laittaa. Valitettavasti tämä usein unohtuu, koska sillä ei ole valmiin projektin kannalta merkitystä. Vanhentunut projektitieto heijastuu ACS800 MPCompileriin useimmiten puuttuvilla tai ylimääräisillä projektimanuaaleilla, jolloin käyttäjän on osattava korjata virheellinen listaus manuaalinäkymässä. Vaikka tällaiset tapaukset ovat harvinaisia, liiallinen luottaminen tuotekonfiguraattorin projektitiedon paikkansapitävyyteen on tuonut esille varjopuolia.

Työhön kuuluu myös joitain osa-alueita, joiden lopputulokseen en ole täysin tyytyväinen. Indeksisivu kaipaisi tietyiltä osiltaan uudelleen suunnittelua. Linjakokoonpano-välilehti tarjoaa liian vähän informaatiota pärjätäkseen käytettävyydessä käyttöliittymän tiedostojärjestelmänäkymän kanssa. Ulkoasu ja navigointi onnistuivat mielestäni hyvin, mutta sivu yleisesti kaipaisi optimointia tai vaihtoehtoisen kevytversion, koska vaarana on animaatioiden ja tämän seurauksena navigoinnin hidastuminen hitaammilla tietokoneilla. Lisäksi olen huomannut ACS800 MPCompilerin ohjelmakoodin rakenteen kai-

paavan ylläpidollisia parannuksia. Toteutuksen aikana *Container*- ja *GenerationThread*-luokat paisuivat liian suuriksi, joten näiden tehtäviä olisi syytä jakaa muualle. Projektitiedoille olen miettinyt kokonaan oman luokan toteuttamista ja Windows API -ohjelmakoodin siirtämistä *GenerationThread*-luokasta omaan tiedostoonsa.

Kaiken kaikkiaan insinöörityön tuloksena saatu toimituskohtaisen dokumenttikokoelman hallintatyökalu täytti sille asetetut vaatimukset hyvin. Tulostustoimintoa ei tosin saatu julkaisuversioon ja sen toteuttaminen alkuperäisten suunnitelmien mukaan tulee tuskin onnistumaan. Hallintatyökalu ei kuitenkaan riitä tällaisenaan korvaamaan kokonaan CDmacroa. Tähän tullaankin keskittämään enemmän huomiota jatkokehityksessä. Olisi myös hyvä jotenkin varmistaa tuotekonfiguraattorin projektitiedon paikkansapitävyys, mutta tämä ei ole välttämättä ollenkaan mahdollista nykyisillä ratkaisuilla. Epäkohdista huolimatta, hallintatyökalu on säästänyt dokumenttikokoelman muodostamiseen käytettyä aikaa ja vaivaa. Sovellussuunnittelijan arvio käsiteltävien projektien määrästä on useita satoja vuodessa, joten ohjelman vaikutus työtehokkuuden paranemiseen on konkreettista. Merkitsevistä taloudellisista hyödyistä voidaan tuskin puhua nykyisten käyttäjien vähäisyydestä johtuen.

10 Yhteenveto

Työn tavoitteena oli suunnitella ja toteuttaa hallintatyökalu ACS800 tuoteperheen dokumenttikokoelmien laatimiseen. Toteutus osoittautui paljon arvioitua laajemmaksi ja monimutkaisemmaksi. Alkutaipaleella ohjelmasta visioitiin suoraviivainen ACS800 Product Configuratoria hyödyntävä manuaalienkeräystyökalu, mutta työn edetessä huomattiin, että intoa ja rahkeita riittää muuhunkin. Aikataulun venyminen ei haitannut, koska ohjelmointi Qt-kehysympäristössä osoittautui erittäin mieluisaksi. Lisäksi uusien toimintojen ja käyttöliittymän suunnittelu ja parantaminen oli antoisaa ja hyödyllistä työtä. Määrittelyjen eläminen työn aikana ei tietenkään ole toivottavaa, mutta insinöörityön vaatimusmäärittelyssä ei osattu riittävällä tarkkuudella arvioida niitä ominaisuuksia, joilla ohjelman käyttötarkoitukseen soveltuvuutta saatiin parannettua. Samalla tuli kerrattua toiminnallisen määrittelyn ja arkkitehtuurin suunnittelun tärkeydet, joihin toivottavasti nyt osaa jatkossa paneutua tarkemmin.

Ohjelmassa riittää vielä työsarkaa, mikäli CDmacron käytöstä halutaan kokonaan luopua. Qt tarjoaa erinomaiset valmiudet sovelluksen kehitykseen. Mahdollisuus suunnitella ja toteuttaa hallintatyökalu vapain käsin vaatimuksien puitteissa aina toteutustekniikan valitsemista myöten oli erittäin mukavaa ja opettavaista. Mikäli uuden sukupolven tuotekonfiguraattorin kehityksessä aiotaan manuaalien keräys liittää osaksi järjestelmää, tulee ACS800 MPCompilerin toiminnallisuus esille tärkeänä vertailukohtana, mahdollisesti avaten jopa uusia jatkokehityssuunnitelmia.

Lähteet

1 Suomalaiset juuret. (WWW-dokumentti.) ABB Oy. <http://www.abb.fi/cawp/fiabb251/4c7fb86040626fd9c2256b2000427c68.aspx>. 5.1.2010. Luettu 17.1.2010.

2 Strömberg-Asea-BBC -yhteistyön juuret. (WWW-dokumentti.) ABB Oy. <http://www.abb.fi/cawp/fiabb251/49ec18cae8cea8b1c12575bc002a085e.aspx>. 18.6.2009. Luettu 17.1.2010.

3 ACS800 Single Drives Catalog. (WWW-dokumentti.) ABB Oy. <http://library.abb.com/global/scot/scot201.nsf/veritydisplay/f4b8337f962281bec1257603001e53ad/$File/EN_ACS800singledrivescatalog_REVK.pdf>. 12.8.2009. Luettu 11.1.2010.

4 ACS800 Multidrives Catalog. (WWW-dokumentti.) ABB Oy. <http://library.abb.com/global/scot/scot201.nsf/veritydisplay/0bd6543028bfdacfc125759000274f1b/$File/ACS800MultidrivesCatalogREVF_EN.pdf>. 20.2.2009. Luettu 11.1.2010.

5 General FAQ. (WWW-dokumentti.) wxWidgets. <http://www.wxwidgets.org/docs/faqgen.htm>. Luettu 8.11.2009.

6 LGPL License Option Added to Qt. (WWW-dokumentti.) Nokia Corporate <http://qt.nokia.com/about/news/lgpl-license-option-added-to-qt>. Luettu 28.10.2009.

7 Products - Qt. (WWW-dokumentti.) Nokia Corporation. <http://qt.nokia.com/products>. Luettu 28.10.2009.

8 Qt Licensing. (WWW-dokumentti.) Nokia Corporation. <http://qt.nokia.com/products/licensing>. Luettu 28.10.2009.

9 Qt Reference Documentation. (WWW-dokumentti.) Nokia Corporation. <http://doc.trolltech.com/4.5/index.html>. Luettu 28.10.2009.

10 Qt Object Model. (WWW-dokumentti.) Nokia Corporation. <http://doc.trolltech.com/4.5/object.html>. Luettu 28.10.2009.

11 Blanchette, Jasmin. Summerfield, Mark. C++ GUI Programming with Qt 4. Massachusetts: U.S. Corporate and Government Sales. 2006.

12 An Introduction to Model/View Programming. (WWW-dokumentti.) Nokia Corporation. <http://qt.nokia.com/doc/4.5/model-view-introduction.html>. Luettu 28.10.2009.

13 Windows API. (WWW-dokumentti.) Microsoft Corporation. <http://msdn.microsoft.com/en-us/library/cc433218(VS.85).aspx>. 12.3.2009. Luettu 10.1.2010.

14 Visual Basic for Applications Frequently Asked Questions. (WWW-dokumentti.) Microsoft Corporation. <http://msdn.microsoft.com/en-us/isv/bb190540.aspx>. Luettu 11.1.2010.

15 What does Visual Basic do? (WWW-dokumentti.) WOODWEB, Inc. <http://www.woodweb.com/knowledge_base/What_does_Visual_Basic_do.html>. 13.6.2001. Luettu 11.1.2010.

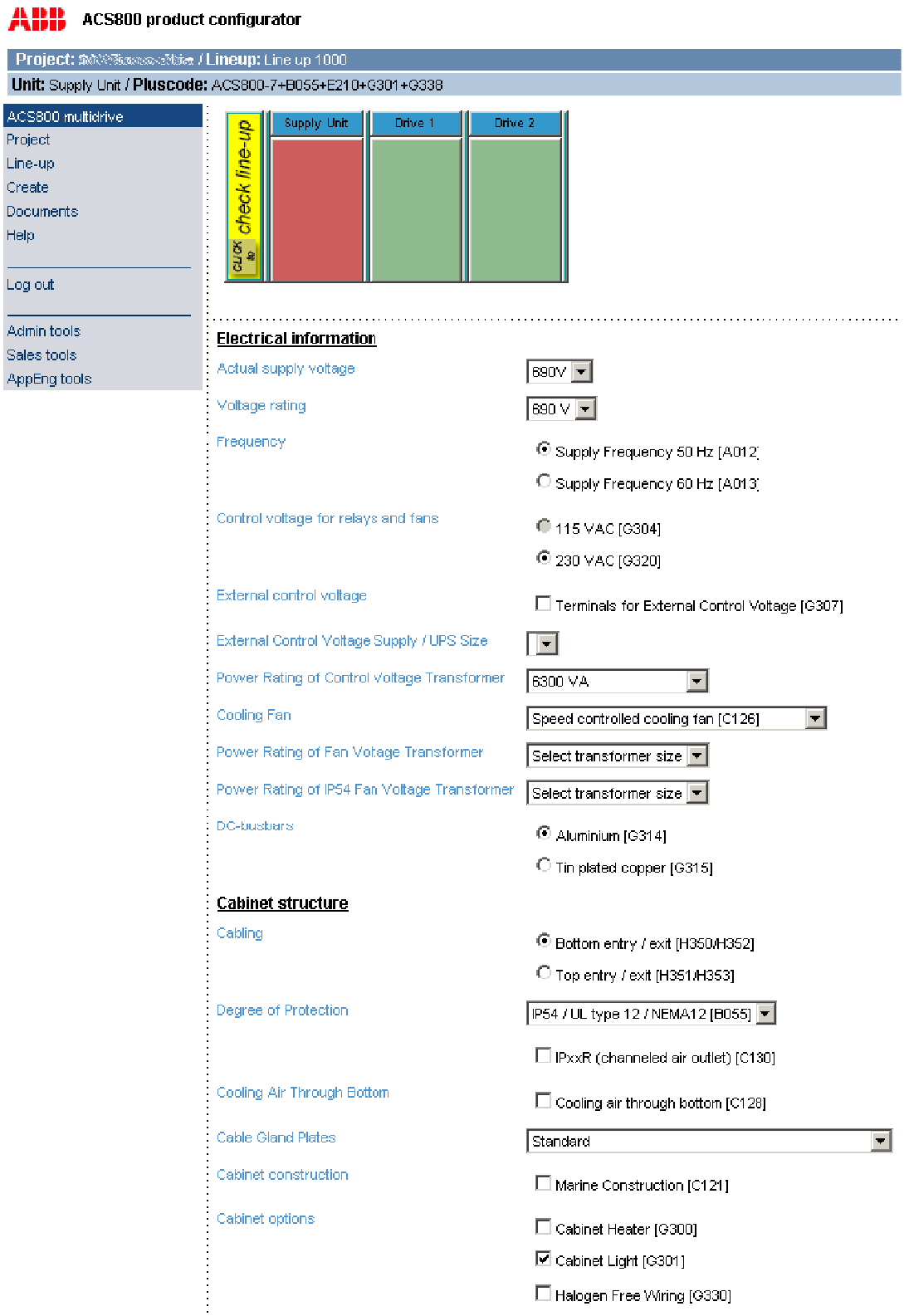
16 Resig, John. jQuery. (WWW-dokumentti.) <http://jquery.com/>. Luettu 11.1.2010.

17 What is the Visual Studio add-in? (WWW-dokumentti.) Nokia Corporation. <http://qt.nokia.com/developer/faqs/what-is-the-visual-studio-add-in/view>. Luettu 8.11.2009.

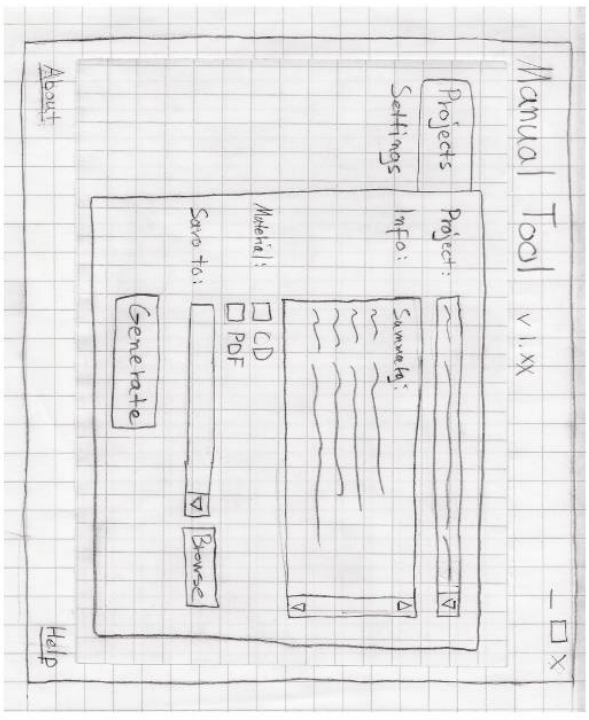
18 Microsoft DOS net command. (WWW-dokumentti.) Computer Hope. <http://www.computerhope.com/nethlp.htm>. Luettu 9.12.2009.

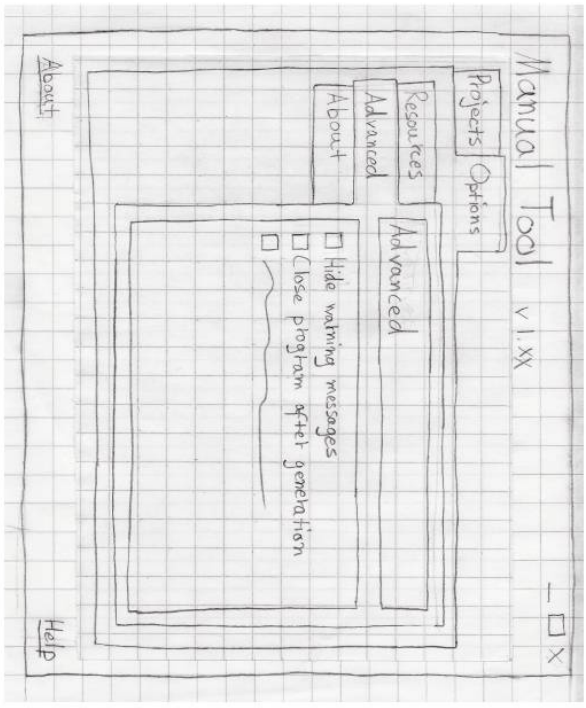
19 Using Qt Assistant as a Custom Help Viewer. (WWW-dokumentti.) Nokia Corporation. <http://doc.trolltech.com/4.5/assistant-custom-help-viewer.html>. Luettu 28.12.2009.

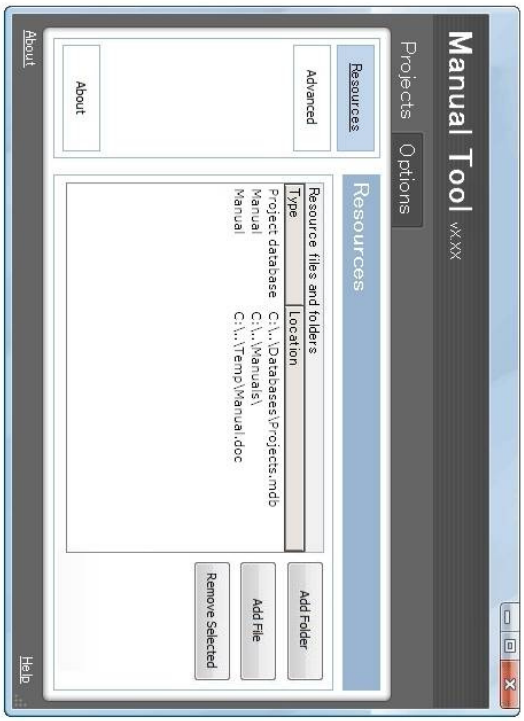
Liite 1: Kuvakaappaus ACS800 Product Configurator -järjestelmästä



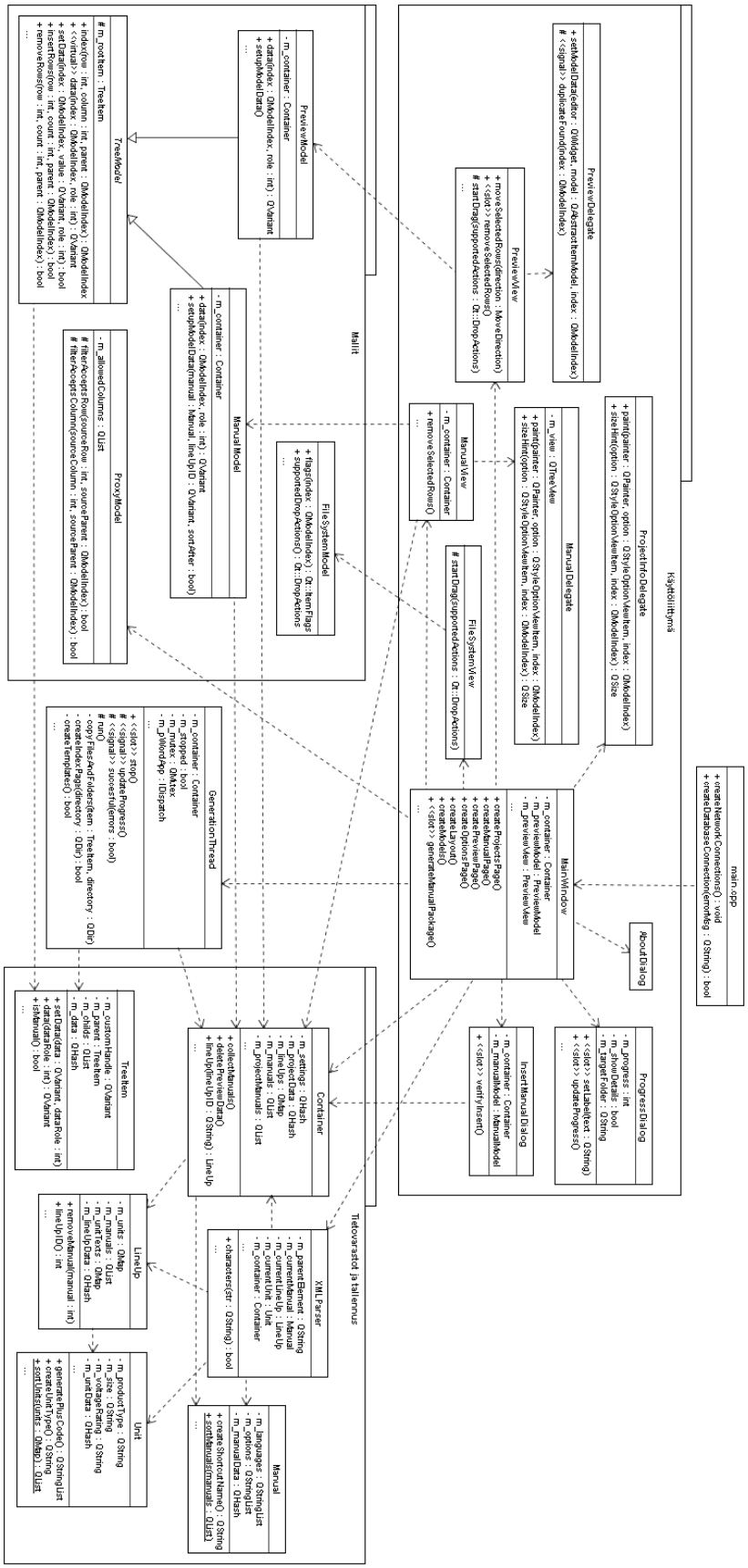
Liite 2: ACS800 MPCompiler -ohjelman käyttöliittymähahmotelmia







Liite 3: ACS800 MPCompiler -ohjelman luokkakaavio



Liite 4: Manuaalisääntötiedoston MakeXML-makro

```
Sub MakeXML()

... ' Muuttujien määritykset

On Error GoTo ErrorHandler:
XMLFileName = "man_stat_manuals.xml"

' Set the range for the XML fields
Set generalFields = Range("A3:M3")
Set languageFields = Range("Z3:AK3")
Set optionFields = Range("AQ3:EM3")
Set xmlFields = Union(generalFields, languageFields, optionFields)

' Set the row from where to start gathering data in the XML fields
firstRow = 5

Application.StatusBar = "Generating manual status list XML file: " & _
                        XMLFileName
currentPath = ThisWorkbook.Path
XMLFullPath = currentPath & "\" & XMLFileName

' Find the last row in the worksheet
If WorksheetFunction.CountA(Cells) > 0 Then
    lastRow = Cells.Find(What:="*", After:=[A1], SearchOrder:=xlByRows, _
                         SearchDirection:=xlPrevious).row
End If

' Initialize the XML field names array 1-dimensional
maxIndex = 0
ReDim FieldNames(xmlFields.Count - 1, maxIndex)

' Iterate through the given cell range to fill the XML fields' array
i = 0
For Each field In xmlFields
    ' Split each cell to check if it contains multiple options
    strArray = Split(field.Value, "+")
    index = UBound(strArray)

    ' In case a cell contains multiple options, resize the container array
    ' multidimensional to have separate dimensions for each option
    If index > maxIndex Then
        ReDim Preserve FieldNames(xmlFields.Count - 1, index)
        maxIndex = index
    End If

    ' Format the cell's options and add them to different dimensions
    For d = LBound(strArray) To UBound(strArray)
        FieldNames(i, d) = FillSpaces(CStr(strArray(d)))
    Next d
    i = i + 1
Next field

languageStartField = languageFields.Column
languageEndField = languageFields.Column + languageFields.Count - 1
optionStartField = optionFields.Column
optionEndField = optionFields.Column + optionFields.Count - 1
```

Liite 4: Manuaalisääntötiedoston MakeXML-makro

```
Open XMLFullPath For Output As #1
Print #1, "<?xml version=" & Chr(34) & "1.0" & Chr(34) & " _
          encoding=" & Chr(34) & "UTF-8" & Chr(34) & "?>" 'ISO-8859-1
Print #1, "<Manuals>"

For row = firstRow To lastRow
    Print #1, "<Manual>"
    i = 0
    For Each field In xmlFields
        If field.Column = languageStartField Then
            Print #1, "<Languages>"
        ElseIf field.Column = optionStartField Then
            Print #1, "<Options>"
        End If

        ' Iterate through all the dimensions of the XML fields' array
        ' and write an XML entry for each valid dimension
        For d = LBound(FieldNames, 2) To UBound(FieldNames, 2)
            If FieldNames(i, d) = "" Then
                Exit For
            End If

            If field.Column >= optionStartField Then
                Print #1, "<_" & FieldNames(i, d) & ">" & _
                          RemoveAmpersands(Cells(row, field.Column).Value) & _
                          "</_" & FieldNames(i, d) & ">"
            Else
                Print #1, "<" & FieldNames(i, d) & ">" & _
                          RemoveAmpersands(Cells(row, field.Column).Value) & _
                          "</" & FieldNames(i, d) & ">"
            End If
        Next d
        i = i + 1

        If field.Column = languageEndField Then
            Print #1, "</Languages>"
        ElseIf field.Column = optionEndField Then
            Print #1, "</Options>"
        End If
    Next field
    Print #1, "</Manual>"
Next row

Print #1, "</Manuals>"
Close #1

' Give control of the status bar back to the application
Application.StatusBar = False
Exit Sub

ErrorHandler:
Application.StatusBar = "Error generating Manual Tool XML reference file"
Application.Wait Now + TimeValue("00:00:01")
Application.StatusBar = False

End Sub
```

Liite 5: Projektimanuaalien keräämisen suorittava collectManuals-funktio

```
//************************************************
//! Function: Container::collectManuals
//
// Binds manuals to the project and line-ups, by handing them pointers to the
// correct Manual objects. We compare the line-ups' unit plus codes to the
// manuals' option and language codes, to resolve if a binding is required.
//
//! Return: void
//************************************************
void Container::collectManuals()
{
    foreach (LineUp* lineUp, m_lineUps.values())
    {
        foreach (Unit* unit, lineUp->units().values())
        {
            QStringList plusCode = unit->generatePlusCode();
            foreach (Manual* manual, m_manuals)
            {
                // Skip manual if it doesn't have a file name set
                if (!manual->manualData("File_name").isEmpty())
                {
                    QStringListIterator i(manual->options());
                    while (i.hasNext())
                    {
                        // Collect the manual if the unit's plus code
                        // contains the manual's option AND language
                        if (plusCode.contains(i.next()))
                        {
                            foreach (QString language, manual->languages())
                            {
                                if (plusCode.contains(language))
                                {
                                    // Existing manuals won't be inserted
                                    lineUp->insertManual(manual);
                                    insertProjectManual(manual);

                                    // Manual binding complete, no need to
                                    // check the remaining options
                                    i.toBack();
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Liite 6: TreeItem-luokan toteutus

```
//************************************************
//! Class: TreeItem
//  Author: Pekka Hellsten
//
//  TreeItem class is used as the data source for classes derived from
//  TreeModel. One instance represents one row in the corresponding tree view.
//  The structure is hierarchical, each TreeItem knows its parent and child
//  items. Data is stored in a QHash container, this is the only private data
//  member which is saved over a drag & drop operation.
//
//! Inherits: n/a
//************************************************

... // Otsikkotiedostojen lisäämiset

class TreeItem
{
   public:
      TreeItem(QHash<int, QVariant>& data, TreeItem* parent = 0);
      ~TreeItem();

      void setData(const QVariant& data, int dataRole);
      void setData(const QHash<int, QVariant>& data);
      QVariant data(int dataRole) const;

      void setChildren(const QList<TreeItem*>& children);
      void appendChild(TreeItem* child);
      void insertChild(int row, TreeItem* child);
      bool insertChildren(int row, int count);
      bool removeChildren(int row, int count);

      TreeItem* child(int row) const;
      TreeItem* parent() const;
      int childCount() const;
      int columnCount() const;
      int row() const;

      void setCustomHandle(QVariant handle);
      QVariant customHandle() const;
      QList<TreeItem*> findChildren(int role, const QVariant& value) const;

      // These functions are provided for conveniance
      bool isFolder() const;
      bool isBinderTitle() const;
      bool isLineUp() const;
      bool isShortcut() const;
      bool isManual() const;
      bool isCategory() const;

      friend QDataStream& operator>>(QDataStream& in, TreeItem* item);
      friend QDataStream& operator<<(QDataStream& out, const TreeItem* item);
      static QList<TreeItem*> findItems(int role, const QVariant& value,
                                        const TreeItem* startFrom);

   private:
      QVariant m_customHandle;
      TreeItem* m_parent;

      QList<TreeItem*> m_childs;
      QHash<int, QVariant> m_data;
};
```

Liite 6: TreeItem-luokan toteutus

```
#include "treeitem.h"

TreeItem::TreeItem(QHash<int, QVariant>& data, TreeItem* parent)
{
    m_parent = parent;
    m_data = data;
    m_customHandle = QVariant();
}

TreeItem::~TreeItem()
{
    qDeleteAll(m_childs);
}

void TreeItem::appendChild(TreeItem* child)
{
    m_childs.append(child);
}

void TreeItem::insertChild(int row, TreeItem* child)
{
    m_childs.insert(row, child);
}

int TreeItem::childCount() const
{
    return m_childs.count();
}

int TreeItem::columnCount() const
{
    return m_data.count();
}

QVariant TreeItem::data(int dataRole) const
{
    return m_data.value(dataRole);
}

int TreeItem::row() const
{
    if (m_parent)
        return m_parent->m_childs.indexOf(const_cast<TreeItem*>(this));
    else
        return 0;
}

TreeItem* TreeItem::child(int row) const
{
    return m_childs.value(row);
}

TreeItem* TreeItem::parent() const
{
    return m_parent;
}

void TreeItem::setData(const QVariant& data, int dataRole)
{
    m_data.insert(dataRole, data);
}
```

Liite 6: TreeItem-luokan toteutus

```
void TreeItem::setData(const QHash<int, QVariant>& data)
{
    m_data = data;
}

bool TreeItem::isBinderTitle() const
{
    if (m_parent->isLineUp() &&
        data(PreviewItem::ItemType) == PreviewItem::RegularFolder)
        return true;
    else
        return false;
}

bool TreeItem::isFolder() const
{
    int role = m_data.value(PreviewItem::ItemType).toInt();
    if (role == PreviewItem::RegularFolder ||
        role == PreviewItem::ManualFolder ||
        role == PreviewItem::LineUpFolder ||
        role == PreviewItem::UnitFolder ||
        role == PreviewItem::TemplateFolder)
        return true;
    else
        return false;
}

bool TreeItem::isLineUp() const
{
    if (data(PreviewItem::ItemType) == PreviewItem::LineUpFolder)
        return true;
    else
        return false;
}

bool TreeItem::isShortcut() const
{
    if (data(PreviewItem::ItemType) == PreviewItem::ShortcutFile)
        return true;
    else
        return false;
}

bool TreeItem::isManual() const
{
    int role = m_data.value(ManualItem::ItemType).toInt();
    if (role == PreviewItem::ManualFile || role == ManualItem::Manual)
        return true;
    else
        return false;
}

bool TreeItem::isCategory() const
{
    if (data(ManualItem::ItemType) == ManualItem::Category)
        return true;
    else
        return false;
}
```

Liite 6: TreeItem-luokan toteutus

```
bool TreeItem::insertChildren(int row, int count)
{
    if (row < 0 || row > m_childs.count())
        return false;

    for (int i = 0; i < count; ++i)
    {
        QHash<int, QVariant> itemData;
        TreeItem* item = new TreeItem(itemData, this);
        m_childs.insert(row, item);
    }
    return true;
}

bool TreeItem::removeChildren(int row, int count)
{
    if (row < 0 || row > m_childs.count())
        return false;

    // Delete in descending order in case we're removing multiple items
    for (int i = (row + count - 1); i >= row ; i--)
    {
        TreeItem* item = child(i);
        m_childs.removeAt(i);
        delete item;
    }
    return true;
}

void TreeItem::setChildren(const QList<TreeItem*>& children)
{
    m_childs = children;
}

void TreeItem::setCustomHandle(QVariant handle)
{
    m_customHandle = handle;
}

QVariant TreeItem::customHandle() const
{
    return m_customHandle;
}

QList<TreeItem*> TreeItem::findChildren(int role, const QVariant& value) const
{
    QList<TreeItem*> childs;
    foreach (TreeItem* child, m_childs)
    {
        if (child->data(role) == value)
            childs.append(child);
    }
    return childs;
}
```

Liite 6: TreeItem-luokan toteutus

```
QList<TreeItem*> TreeItem::findItems(int role, const QVariant& value,
                                     const TreeItem* startFrom)
{
    QList<TreeItem*> items;

    if (startFrom->data(role) == value)
        items.append(const_cast<TreeItem*>(startFrom));

    // Also check the items' childs
    for (int i = 0; i < startFrom->childCount(); i++)
        items << findItems(role, value, startFrom->child(i));

    return items;
}

QDataStream& operator<<(QDataStream& out, const TreeItem* item)
{
    // First write all the childs' data
    out << (qint32)item->childCount();
    for (int i = 0; i < item->childCount(); i++)
    {
        // Strip item out of const-ness so we can use
        // the serialization operation on it
        TreeItem* child = const_cast<TreeItem*>(item)->child(i);
        out << child; // Recursive call
    }

    // Then write the item's own data
    out << (qint32)item->m_data.count();
    QHashIterator<int, QVariant> i(item->m_data);
    while (i.hasNext())
    {
        i.next();
        out << (qint32)i.key() << i.value();
    }

    return out;
}

// This function is not used, but must be provided since the stream
// operators are registered for the class.
QDataStream& operator>>(QDataStream& in, TreeItem* item)
{
    return in;
}
```

Liite 7: Dokumenttikokoelman muodostus väliaikaismuistiin

roles.h

```
namespace PreviewItem
{
   // DataRole corresponds to the key for TreeItem's QHash data container
   enum DataRole { Name = 0x0000,
                   FileType,
                   Description,
                   SourcePath,     // Provided for shortcut items
                   Path,           // The item's location inside the package
                   ID,
                   Icon,           // The item's QIcon used by the view
                   ItemType };     // Type enum

   // Type defines the item's purpose, works as an internal ID
   enum Type { RegularFile = 0x0200,
               ManualFile,
               ShortcutFile,
               TemplateFile,
               RegularFolder,
               ManualFolder,
               LineUpFolder,
               UnitFolder,
               TemplateFolder,
               NotSpecified };
}
```

PreviewModel.cpp

```
void PreviewModel::setupModelData()
{
   // Cleanup old data
   int count = m_rootItem->childCount();
   for (int i = 0; i < count; i++)
   {
       removeRows(0, 1, QModelIndex());
       emit updateProgress();
   }

   // Initialize file icons used by the items
   QFileIconProvider iconProvider;
   QIcon pdfIcon, wordIcon;
   QIcon folderIcon = QApplication::style()->standardIcon(QStyle::SP_DirIcon);
   QIcon linkIcon =
   QApplication::style()->standardIcon(QStyle::SP_FileLinkIcon);

  /*****************************************
   * Manual folder                         *
   *****************************************/
   QHash<int, QVariant> itemData;
   bool copyManuals = m_container->copyManuals();
   bool createLanguageFolder = m_container->createLanguageFolders();

   QList<Manual*> manuals = m_container->projectManuals();
   if (copyManuals && manuals.count() > 0)
   {
      itemData = createItemData("Manuals", "File Folder", "", "", "", "",
                                folderIcon, PreviewItem::ManualFolder);

      TreeItem* manualFolder = new TreeItem(itemData, m_rootItem);
```

Liite 7: Dokumenttikokoelman muodostus väliaikaismuistiin

```
    m_rootItem->appendChild(manualFolder);
    Manual::sortManuals(manuals); // Sort by language

    // Get the pdf icon from the first manual of the list
    QFileInfo pdfInfo(manuals.first()->manualData("File_name"));
    if (pdfInfo.isFile())
       pdfIcon = iconProvider.icon(pdfInfo);

    foreach (Manual* manual, manuals)
    {
       itemData.clear();
       itemData = createItemData(manual->createFileName(),
                                 manual->createFileType(),
                                 manual->manualData("Document_name"),
                                 manual->manualData("File_name"), "",
                                 manual->manualData("Code"), pdfIcon,
                                 PreviewItem::ManualFile);

       TreeItem* destinationFolder;
       if (createLanguageFolder)
       {
          QString language = manual->manualData("Language");
          QList<TreeItem*> languageItems =
          TreeItem::findItems(PreviewItem::Name, language, manualFolder);

          TreeItem* languageFolder = 0;
          if (languageItems.count() > 0)
             languageFolder = languageItems.first();

          // If the language subfolder doesn't exist, create one
          if (!languageFolder)
             languageFolder = setupFolder(language, manualFolder);

          destinationFolder = languageFolder;
       }
       else
          destinationFolder = manualFolder;

       // Add the manual TreeItem to the primary manual folder
       TreeItem* manual = new TreeItem(itemData, destinationFolder);
       destinationFolder->appendChild(manual);

       emit updateProgress();
    }
 }

/*****************************************
 * Line-up folders                       *
 *****************************************/
 QStringList folders;
 bool createUnitFolder = m_container->createUnitFolders();
 bool createBinderMaterial = m_container->createBinderMaterial();
 foreach (LineUp* lineUp, m_container->lineUps())
 {
    itemData.clear();
    QString name = lineUp->position() + " " + lineUp->name();
    itemData = createItemData(name, "File Folder", "Line-up", "", "",
                              lineUp->lineUpData("LineUpID"), folderIcon,
                              PreviewItem::LineUpFolder);

    TreeItem* lineUpItem = new TreeItem(itemData, m_rootItem);
```

Liite 7: Dokumenttikokoelman muodostus väliaikaismuistiin

```
m_rootItem->appendChild(lineUpItem);

// Create a folder for the manual shortcuts
QList<Manual*> lineUpManuals = lineUp->manuals();
if (copyManuals && lineUpManuals.count() > 0)
{
   itemData.clear();
   itemData = createItemData("Manuals", "File Folder", "", "", "", "",
                             folderIcon, PreviewItem::ManualFolder);

   TreeItem* manualFolder = new TreeItem(itemData, lineUpItem);
   lineUpItem->appendChild(manualFolder);
   Manual::sortManuals(lineUpManuals); // Sort by language

   foreach (Manual* manual, lineUpManuals)
   {
      itemData.clear();
      itemData = createItemData(manual->createShortcutName(),
                 "Shortcut", manual->manualData("Document_name"), "",
                 "", manual->manualData("Code"), linkIcon,
                 PreviewItem::ShortcutFile);

      TreeItem* destinationFolder;
      if (createLanguageFolder)
      {
         QString language = manual->manualData("Language");
         QList<TreeItem*> languageItems =
         TreeItem::findItems(PreviewItem::Name, language, manualFolder);


         TreeItem* languageFolder = 0;
         if (languageItems.count() > 0)
            languageFolder = languageItems.first();

         // If the language subfolder doesn't exist, create one
         if (!languageFolder)
            languageFolder = setupFolder(language, manualFolder);

         destinationFolder = languageFolder;
      }
      else
         destinationFolder = manualFolder;

      // Add the manual shortcut TreeItem to line-up manual folder
      TreeItem* manual = new TreeItem(itemData, destinationFolder);
      destinationFolder->appendChild(manual);
   }
}

// Setup the first part of the required subfolders
folders.clear();
if (createUnitFolder)
   folders << "Main Part Lists" << "Dimensional Drawing";
else
   folders << "Part Lists" << "Drawings";

foreach (QString folder, folders)
   setupFolder(folder, lineUpItem);
```

Liite 7: Dokumenttikokoelman muodostus väliaikaismuistiin

```
if (createUnitFolder)
{
   QList<Unit*> sortedUnits = lineUp->units().values();
   Unit::sortUnits(sortedUnits); // Sort by index

   foreach (Unit* unit, sortedUnits)
   {
      itemData.clear();
      QString name = "=" + unit->locationDesignation() +
                     " " + unit->name();
      itemData = createItemData(name, "File Folder",
                                unit->createUnitType(), "", "",
                                unit->unitData("UnitID"), folderIcon,
                                PreviewItem::UnitFolder);

      TreeItem* unitItem = new TreeItem(itemData, lineUpItem);
      lineUpItem->appendChild(unitItem);

      // Setup unit subfolders
      folders.clear();
      folders << "Part List" << "Circuit Diagrams" << "Reports";
      foreach (QString folder, folders)
         setupFolder(folder, unitItem);
   }
}

// Setup the rest of the required subfolders
folders.clear();
folders << "Standard Document" << "Test Reports" << "Declarations";
foreach (QString folder, folders)
   setupFolder(folder, lineUpItem);

if (createBinderMaterial)
{
   itemData.clear();
   itemData = createItemData("Binder Documents", "File Folder", "", "",
              "", "", folderIcon, PreviewItem::TemplateFolder);

   TreeItem* binderFolder = new TreeItem(itemData, lineUpItem);
   lineUpItem->appendChild(binderFolder);

   QString TOCPath =
           m_container->setting(Settings::TableOfContents).toString();
   QString binder1SpinePath =
           m_container->setting(Settings::Binder1Spine).toString();
   QString binder2SpinePath =
           m_container->setting(Settings::Binder2Spine).toString();

   // Get the word icon from the first template
   QFileInfo wordInfo(TOCPath);
   if (wordInfo.isFile())
      wordIcon = iconProvider.icon(wordInfo);

   setupTemplate("Binder_TOC.doc", "Table of Contents",
                 TOCPath, "TOC", wordIcon, binderFolder);
   setupTemplate("Binder1_Spine.doc", "Binder 1 Spine",
                 binder1SpinePath, "Binder1", wordIcon, binderFolder);
   setupTemplate("Binder2_Spine.doc", "Binder 2 Spine",
                 binder2SpinePath, "Binder2", wordIcon, binderFolder);
}
emit updateProgress();
```

Liite 7: Dokumenttikokoelman muodostus väliaikaismuistiin

```
    }
    // Inform the model and its associated view that the data has been reset
    reset();
}

void PreviewModel::setupTemplate(const QString& name,
                                 const QString& description,
                                 const QString& path, const QString& id,
                                 const QIcon& icon, TreeItem* parentItem)
{
    QHash<int, QVariant> itemData = createItemData(name, "doc File",
                                    description, path, "", id, icon,
                                    PreviewItem::TemplateFile);

    TreeItem* templateItem = new TreeItem(itemData, parentItem);
    parentItem->appendChild(templateItem);
}

TreeItem* PreviewModel::setupFolder(const QString& folderName,
                                    TreeItem* parentItem)
{
    QHash<int, QVariant> itemData = createItemData(folderName, "File Folder",
                                    "", "", "", "",
                                    QApplication::style()->standardIcon(
                                    QStyle::SP_DirIcon),
                                    PreviewItem::RegularFolder);

    TreeItem* folder = new TreeItem(itemData, parentItem);
    parentItem->appendChild(folder);

    return folder;
}

QHash<int, QVariant> PreviewModel::createItemData(const QString& name,
                     const QString& fileType, const QString& description,
                     const QString& sourcePath, const QString& path,
                     const QString& id, const QIcon& icon,
                     const PreviewItem::Type type) const
{
    QHash<int, QVariant> itemData;
    itemData.insert(PreviewItem::Name, name.trimmed());
    itemData.insert(PreviewItem::FileType, fileType.trimmed());
    itemData.insert(PreviewItem::Description, description.trimmed());
    itemData.insert(PreviewItem::SourcePath, sourcePath.trimmed());
    itemData.insert(PreviewItem::Path, path.trimmed());
    itemData.insert(PreviewItem::ID, id.trimmed());
    itemData.insert(PreviewItem::Icon, icon);
    itemData.insert(PreviewItem::ItemType, type);

    return itemData;
}
```

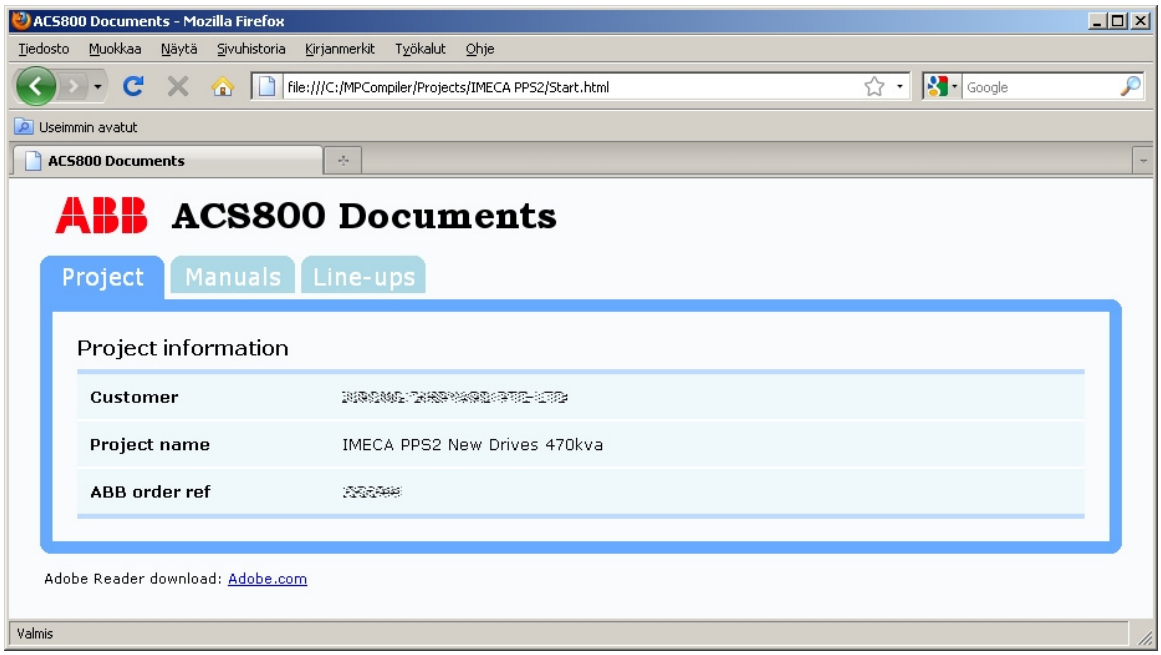
Liite 8: GenerationThread-säikeen elinajan määrittävä run-funktio

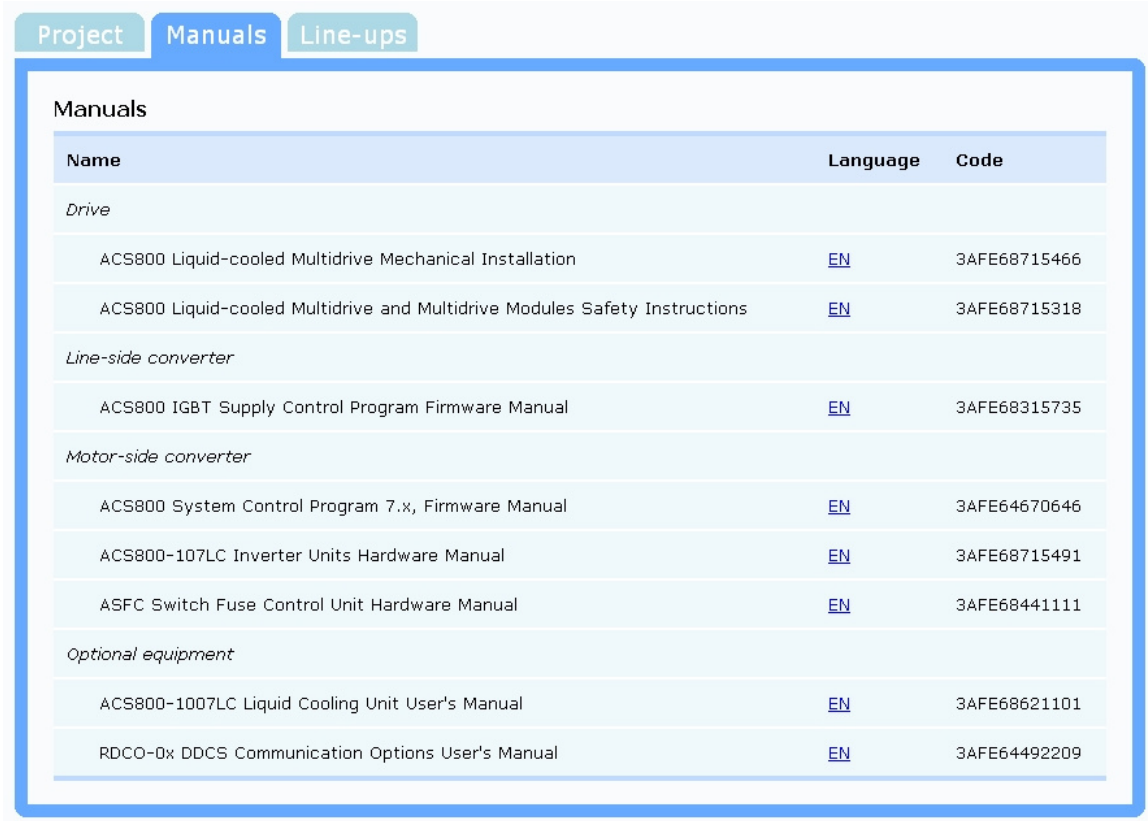
```
//************************************************
//! Function: GenerationThread::run
//
//  Thread lifeline function. When this function ends, the thread will close
//  automatically. If the user cancels the manual package generation, the
//  m_stopped boolean is set to true for a controlled exit. The thread
//  communicates with the GUI thread by using signals.
//
//! Return: void
//************************************************
void GenerationThread::run()
{
    emit addDetail(QString("Starting generation..."));
    initializeGeneration();
    emit updateProgress();

    // Copy the package content by iterating through all items recursively
    QDir rootDirectory(m_container->saveLocation());
    copyFilesAndFolders(m_rootPreviewItem, rootDirectory);

    if (!m_stopped)
    {
        if (!createIndexPage(rootDirectory))
        {
            emit addErrorDetail(QString("Could not create index page: %1")
                                        .arg(m_errorMessage));
            m_errors = true;
        }
        emit updateProgress();
    }

    if (!m_stopped && m_container->createBinderMaterial())
    {
        #ifdef Q_WS_WIN
            if (!createTemplates())
            {
                emit addErrorDetail(QString("Could not create templates: %1")
                                            .arg(m_errorMessage));
                m_errors = true;
            }
            emit updateProgress();
        #else
                emit addErrorDetail(QString("Template creation is not
                                             supported for this system!"));
        #endif // Q_WS_WIN
    }

    if (!m_stopped)
    {
        emit addDetail(QString("Generation complete"));
        emit successful(m_errors);
    }
    else
        emit addDetail(QString("Generation stopped"));
}
```

Liite 9: Indeksisivun template_Start.html-mallitiedosto

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>ACS800 Documents</title>
    <link type="text/css" rel="stylesheet" href="index/css/default.css"/>
    <link type="text/css" rel="stylesheet" media="print"
          href="index/css/print.css"/>
    <script type="text/javascript" src="index/js/jquery.js"><!-- --></script>
    <script type="text/javascript"
            src="index/js/jquery.corner.js"><!-- --></script>
    <script type="text/javascript" src="index/js/start.js"><!-- --></script>
    <noscript><h3>Your browser does not support JavaScript.</h3></noscript>
</head>
<body>
    <div id="container">
        <div id="header">
            <img src="index/img/title.png"/>
            <ul id="nav">
                <li><a href="#" name="#project">Project</a></li>
                <li><a href="#" name="#manuals">Manuals</a></li>
                <li><a href="#" name="#lineUps">Line-ups</a></li>
            </ul>
        </div>
        <div id="main">
            <div id="main_border">
                <div id="project"></div>
                <div id="manuals"></div>
                <div id="lineUps">
                    <div id="lineUpsLeft"></div>
                    <div id="lineUpsRight"></div>
                </div>
            </div>
        </div>
        <div id="footer">
            <table>
                <tr>
                    <td>Adobe Reader download:</td>
                    <td><a href="http://www.adobe.com">Adobe.com</a></td>
                </tr>
            </table>
        </div>
    </div>
</body>
</html>
```

Liite 10: Kuvakaappaukset Start.html-indeksisivusta



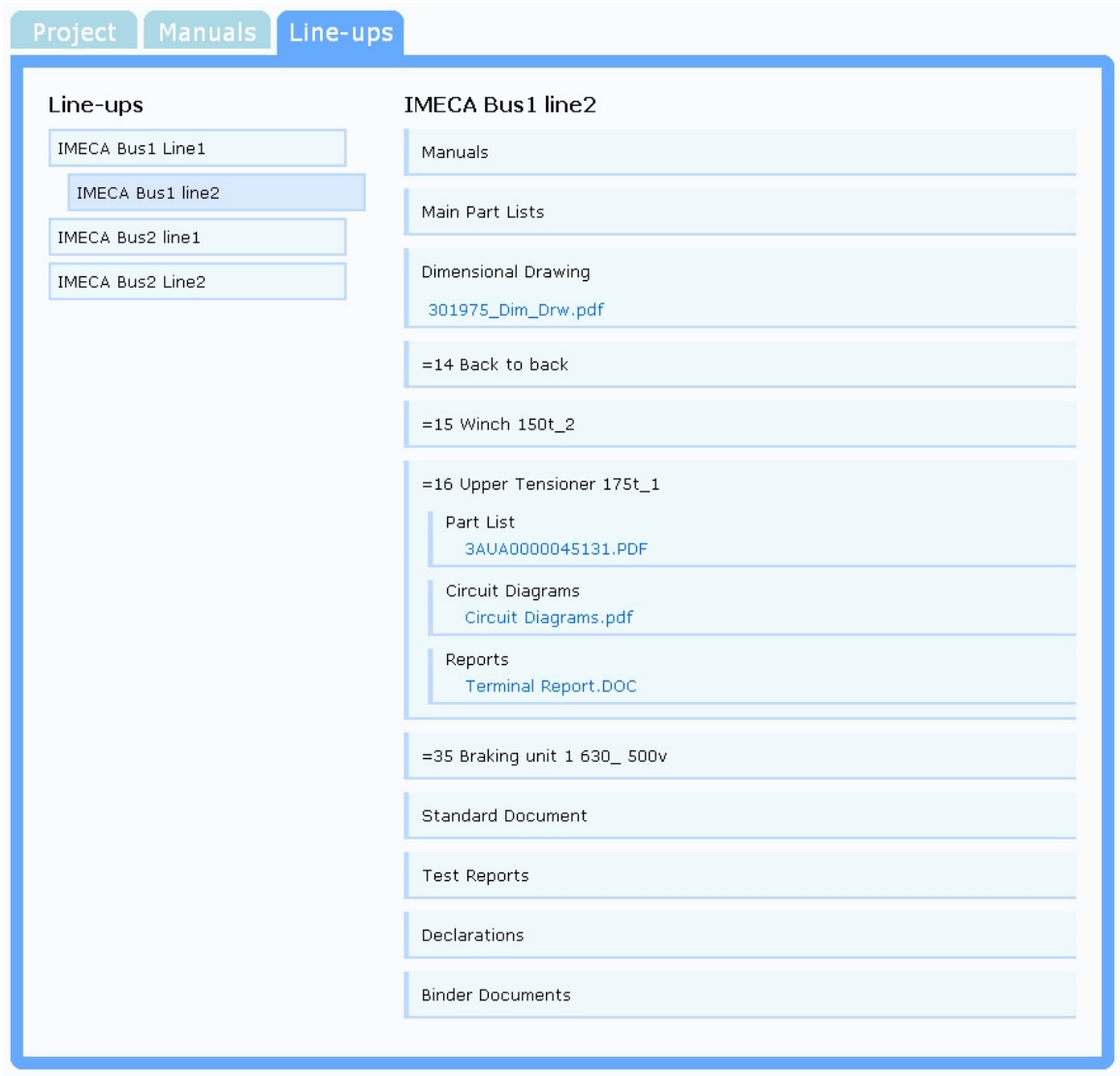


Project | Manuals | Line-ups

Manuals

Name	Language	Code
Drive		
ACS800 Liquid-cooled Multidrive Mechanical Installation	EN	3AFE68715466
ACS800 Liquid-cooled Multidrive and Multidrive Modules Safety Instructions	EN	3AFE68715318
Line-side converter		
ACS800 IGBT Supply Control Program Firmware Manual	EN	3AFE68315735
Motor-side converter		
ACS800 System Control Program 7.x, Firmware Manual	EN	3AFE64670646
ACS800-107LC Inverter Units Hardware Manual	EN	3AFE68715491
ASFC Switch Fuse Control Unit Hardware Manual	EN	3AFE68441111
Optional equipment		
ACS800-1007LC Liquid Cooling Unit User's Manual	EN	3AFE68621101
RDCO-0x DDCS Communication Options User's Manual	EN	3AFE64492209

Liite 10: Kuvakaappaukset Start.html-indeksisivusta



Liite 11: Mallidokumenttien muodostus Windows API -komennoilla

```
//************************************************
//! Function: GenerationThread::createTemplates
//
//  This is the main function for modifying binder document templates.
//  Windows API and COM objects are used through the process. Templates are
//  modified line-up-specifically in createLineUpTemplates() function. Once
//  all templates are modified, the COM library DLLs loaded by the thread are
//  unloaded by calling CoUninitialize() in the cleanUpCOM() function.
//
//! Return: Boolean value indicating the success of the total process
//************************************************
bool GenerationThread::createTemplates()
{
    HRESULT hr;
    CoInitialize(NULL);
    CLSID clsid;

    // Request the CLSID identifier for Microsoft Word
    hr = CLSIDFromProgID(L"Word.Application", &clsid);
    if (FAILED(hr))
    {
        m_errorMessage = QString("Error retrieving Microsoft Word registry
        identifier: 0x%1").arg(QString::number((ulong)hr, 16).toUpper());
        cleanUpCOM();
        return false;
    }

    // Create an instance of the Microsoft Word application
    hr = CoCreateInstance(clsid, NULL, CLSCTX_LOCAL_SERVER, IID_IDispatch,
                          (void**)&m_pWordApp);
    if (FAILED(hr))
    {
        m_errorMessage = QString("Error creating Microsoft Word application
        instance: 0x%1").arg(QString::number((ulong)hr, 16).toUpper());
        cleanUpCOM();
        return false;
    }

    // Get the documents collection object that represents all open documents
    VARIANT getResult;
    VariantInit(&getResult);
    hr = OLEMethod(DISPATCH_PROPERTYGET, &getResult, m_pWordApp,
                   L"Documents", 0);
    if (FAILED(hr))
    {
        m_errorMessage = QString("Error receiving Microsoft Word documents
        collection: 0x%1").arg(QString::number((ulong)hr, 16).toUpper());
        cleanUpCOM();
        return false;
    }
    m_pDocuments = getResult.pdispVal;
```

Liite 11: Mallidokumenttien muodostus Windows API -komennoilla

```
    // Open and modify the template files
    for (int i = 0; i < m_lineUpItems.count(); i++)
    {
        if (!m_stopped)
        {
            if (!createLineUpTemplates(m_lineUpItems.at(i)))
            {
                emit addErrorDetail(QString("Could not create templates for
                line-up %1: %2").arg(m_lineUpItems.at(i)->data(
                PreviewItem::Name).toString()).arg(m_errorMessage));
                m_errors = true;
            }
        }
    }

    emit setLabel(QString("Cleaning up..."));
    hr = OLEMethod(DISPATCH_METHOD, NULL, m_pWordApp, L"Quit", 0);
    if (FAILED(hr))
    {
        emit addErrorDetail(QString("Could not terminate Microsoft Word
        application: 0x%1").arg(QString::number((ulong)hr, 16).toUpper()));
        m_errors = true;
    }

    cleanUpCOM();
    return true;
}
```