TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Jarkko Vainio

Kontekstisensitiivisen ohjejärjestelmän suunnittelu ja toteutus Qt-sovellukseen

Diplomityö

Tarkastaja: professori Kari Systä
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
8. huhtikuuta 2015

TIIVISTELMÄ

JARKKO VAINIO: Kontekstisensitiivisen ohjejärjestelmän suunnittelu ja toteutus Qt-sovellukseen
Tampereen teknillinen yliopisto
Diplomityö, 47 sivua, 5 liitesivua
Toukokuu 2015
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Kari Systä

Avainsanat: Qt-ohjelmistokehys, DITA XML, Qt Help -ohjelmistokehys, kontekstisensitiivinen ohje

Työn tavoitteena on suunnitella kontekstisensitiivinen ohjejärjestelmä automaatiojärjestelmän konfigurointiin ja ohjaamiseen käytettävään PC-työpöytäsovellukseen. Kontekstisensitiivinen ohjejärjestelmä tarjoaa ohjeita käyttäjälle sovelluksen tilaan perustuen. Ohjejärjestelmän perimmäisenä tavoitteena on auttaa sovelluksen käyttäjiä suoriutumaan työstään mahdollisimman helposti ja tehokkaasti. Kohdesovellus on toteutettu Qt-ohjelmistokehystä käyttäen ja samoin toteutettava ohjejärjestelmä käyttää Qt-ohjelmistokehystä. Työhön liittyy oleelliselta osalta myös kohdeympäristöön määritelty ohjeprosessi, jossa ohjemateriaali lähdemuodostaan prosessoidaan Qt-ohjelmistokehyksen tukemaksi ohjepaketiksi. Ohjeprosessissa ohjemateriaalin lähdemuotona toimii DITA XML -merkkauskieli, jota käytetään teknisen dokumentaation tuottamiseen.

Aluksi työssä tarkastellaan olemassa olevia ohjejärjestelmän toteutuksia ja niiden määrittelyjä yleisesti. Taustatutkimuksen jälkeen eritellään ohjejärjestelmän vaatimukset kohdeympäristössään ja suunnitellaan niiden mukainen ohjejärjestelmä. Suunnittelutyössä käytetään hyödyksi olio-ohjelmoinnin menetelmiä sekä UML-kaavioita.

Lopputuloksena on vaatimusten mukainen ohjejärjestelmä, jonka työn tilaaja hyväksyi ja on ottanut sisäisesti koekäyttöön. Ohjejärjestelmän perustoteutus on valmis. Perustoteutuksen lisäksi työssä esitellään myös jatkokehitysideoita. Ohjejärjestelmän, ja siihen liittyvän ohjeprosessin, laaja käyttöönotto kohdesovelluksen todellisessa ympäristössä vaatii ohjejärjestelmän jatkokehitystä.

ABSTRACT

JARKKO VAINIO: Design and Implementation of a Context-Sensitive Help System for a Qt Application
Tampere University of Technology
Master of Science Thesis, 47 pages, 5 Appendix pages
May 2015
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: professor Kari Systä

Keywords: Qt Framework, DITA XML, Qt Help Framework, Context-Sensitive Help

Purpose of this thesis is to design a context sensitive help system. The help system is designed for a Qt framework based PC desktop application that is used to configure and monitor an automation system. The context sensitive help system will provide help for users based on the application's current state and assist users in doing their work easily and efficiently. There is also a related help process defined for the automation system that will largely affect the design of the help system. The help process defines how source material is processed into a Qt help package. The used source format is DITA XML which is a markup language used to produce technical documentation.

The thesis firstly takes a look at some existing implementations and defines help systems in general terms. After the background study, requirements are specified and the help system is designed based on those requirements. Object oriented programming methods and UML diagrams are used as design tools. Finally the design process will lead to a completed help system that fulfils its requirements. In the end of the thesis some further development options are presented.

The completed help system has been taken into internal test use. Large scale deployment of the help system, and its related help process, in a real world environment requires further development.

ALKUSANAT

Tässä diplomityössä esitelty ohjejärjestelmä on tehty Wapice Oy:n asiakasyrityksen tilauksesta. Työ on osa prosessikokonaisuutta, joka kattaa ohjemateriaalin tuottamisen ja sen käyttämisen ohjejärjestelmän kautta. Sen lisäksi tämä ohjeprosessi ja ohjejärjestelmän kohdesovellus ovat osa suurta teollisuusautomaatiojärjestelmää, joten suunnitteluympäristö on lyhyesti sanottuna todella mielenkiintoinen ja haastava.

Asko Lemmetyinen Wapicen Vaasan toimistolla kehitti työn ohjeprosessiin kuuluvaa DITA-materiaalin ohjepaketiksi prosessointia samalla kun itse Tampereen toimistolla kehitin ohjejärjestelmää kohdesovellukseen. Työstimme molemmat saman ohjeprosessin eri päätyjä ja lopputuloksena oli hyvin toimiva kokonaisuus. Askon kehittämä osuus tuottaa valmiita ohjepaketteja, joita toteuttamani ohjejärjestelmä käyttää. Työn aihe oli sopivan haastava ja hyvin opettavainen, varsinkin kun se on osa suurempaa kokonaisuutta monessa eri suhteessa. Sain reilusti vastuuta ohjejärjestelmän suunnittelussa sekä projektin eteenpäin viemisessä, ja olen hyvin kiitollinen saamastani luottamuksesta.

Haluan kiittää erityisesti Asko Lemmetyistä hyvin sujuneesta yhteistyöstä, sekä Toni Mattilaa, joka toimi työni ohjaajana. Toni on ohjannut työtäni siitä asti kun siirryin nykyiseen projektiini ja hän on jakanut valtavasti tarpeellista tietoa ja neuvoja näiden vuosien aikana, niin minulle, kuin myös muille projektissa työskenteleville. Kiitos myös työni tarkastajalle professori Kari Syställe, joka tarjosi työhön erittäin hyödyllistä näkemystä ja vinkkejä.

Tampereella, 19.04.2015

Jarkko Vainio

SISÄLLYS

LYHENTEET JA TERMIT

CHM	Microsoft Compiled HTML Help. Windows-käyttöjärjestelmän pakattu ohjepaketti.
DITA-OT	DITA Open Toolkit. Työkalupaketti DITA-dokumenttien prosessointiin.
DITA	Darwin Information Typing Architecture. XML-rakenteeseen pohjautuva rakenteinen teknisten dokumenttien tuottamiseen tarkoitettu merkkauskieli.
ditamap	DITA-muotoinen koontitiedosto, jolla määritellään DITA-aiheiden rakenne. Käytännössä määrittelee dokumentin sisällysluettelon.
DLL	Dynamically Linked Library. Dynaamisesti linkitetty kirjasto, joka voidaan ladata sovelluksen käyttöön ajonaikaisesti.
GUI	Graphical User Interface. Graafinen käyttöliittymä.
HelpID	Yksilöivä tunniste, jolla käyttöliittymän komponentti liitetään sen ohjemateriaaliin ohjeprojektissa.
HTML	HyperText Markup Language. Verkkosivujen rakenteen määrittelyyn käytetty merkkauskieli.
HTTP	HyperText Transfer Protocol. Internetissä verkkosivujen tiedonsiirtoon käytetty protokolla.
HTTPS	HTTP Secure. Salattu versio HTTP-protokollasta.
Javadoc	Java-ohjelmointikielisten sovellusten koodidokumentaatiomekanismi.
LaTeX	LaTeX on ladontajärjestelmä, jolla voidaan tuottaa teknisiä dokumentteja.
MIME	Multipurpose Internet Mail Extensions. Datan sisältötyyppi, jolla erotetaan esimerkiksi binäärimuotoinen teksti kuvasta.
MVC	Model-View-Controller. Malli-näkymä-ohjain-suunnittelumalli. Toiminnallisuus jaetaan mallin nimen mukaisesti kolmeen osaan.

MVD	Model-View-Delegate. Qt-ohjelmistokehyksen toteutus MVC-suunnittelumallista, jossa ohjain on korvattu delegaatilla.
PDF	Portable Document Format. Yleisesti julkaisemiseen käytetty alustariippumaton asiakirjamuoto.
QCH	Qt Compiled Help. Qt-kehyksen ohjepaketti.
QHC	Qt Help Collection. Qt-kehyksen ohjekokoelma.
QHCP	Qt Help Collection Project. Qt-kehyksen ohjekokoelmaprojekti.
QHP	Qt Help Project. Qt-kehyksen ohjeprojektitiedosto.
Qt	C++-ohjelmointikielelle kehitetty käyttöliittymä- ja sovellusohjelmistokehys [15].
qthelp	Qt-ohjejärjestelmän tiedonsiirtoprotokolla.
RTF	Rich Text Format. Vanha Windows-käyttöjärjestelmän WordPad-sovelluksen dokumenttiformaatti.
SGML	Standard Generalized Markup Language. Metakieli, jolla XML on määritelty. [22].
SQLite	Tiedostopohjainen tietokantajärjestelmä.
UML	Unified Modeling Language. Dokumentointi- ja kaaviostandardi.
URL	Uniform Resource Locator. Käytetään internetissä resurssien osoitteistukseen.
widget	Widgetti, käyttöliittymään upotettu pienoisohjelma.
XHTML	Extensible HyperText Markup Language. XML-pohjainen HTML-merkkauskieli.
XML	Extensible Markup Language. Rakenteinen dokumenttistandardi. [22].
XSLT	Extensible Stylesheet Language Transformations. XML-dokumenttien transformaatiokieli, mahdollistaa XML-dokumentin rakenteen muuttamisen. [23].

1. JOHDANTO

Teollisuusautomaatiojärjestelmät ovat monesti hyvin monimutkaisia kokonaisuuksia. Ne koostuvat usein ohjattavista fyysisistä laitteista, elektroniikasta ja ohjelmistoista. Tällaisia järjestelmiä on nykypäivänä paljon, kun tuotantotehokkuutta pyritään jatkuvasti parantamaan kaikilla teollisen toiminnan alueilla. Tehokkuutta lisäämällä luonnollisesti parannetaan yrityksen kannattavuutta ja luodaan arvoa sen omistajille. Teollisuusautomaatiojärjestelmiä löytyy esimerkiksi niin tuotantolinjoista ja -laitoksista, kuin myös tuotteista. Tässä tuotteella ei tarkoiteta kuluttajamarkkinoille tehtyä tuotetta, vaan esimerkiksi toiselle yritykselle tarkoitettua suurempaa järjestelmää tai laitetta, esimerkiksi tuotantokonetta. Kaikkien suurten järjestelmien toteutuksessa pyritään huolehtimaan niiden helppokäyttöisyydestä, jotta niiden käyttäminen olisi ylipäänsä mahdollista. Järjestelmien laajuudesta ja luonteesta johtuen niiden käyttäminen ei kuitenkaan aina ole intuitiivista, vaan niiden sujuvaan ja turvalliseen käyttämiseen tarvitaan erityistä osaamista ja koulutusta. Oleellisessa osassa laitteiden käytön mahdollistamiseksi ovat erilaiset käyttöohjeet ja tekniset manuaalit.

Automaatiojärjestelmiä voidaan ohjata tietokonesovelluksien avulla, jopa hyvin etäältä varsinaisesta laitteesta. Internetiin liitettyjä laitteita voidaan ohjata periaatteessa mistä tahansa. Moderneissa työkoneissa ja -laitteissa ohjaus tapahtuu jopa täysin tietokoneen kautta. Tällöin kaiken järjestelmän käyttöön liittyvän ohjemateriaalin tarjoaminen tietokoneen avulla on hyvin kätevää, eikä paperisia manuaaleja tarvitse selata edestakaisin. Tietokonesovelluksissa käyttöohjeet ja manuaalit on jo pitkään pyritty tarjoamaan erilaisten ohjejärjestelmien avulla, jotka avustavat tarpeellisen tiedon löytämisessä. Ohjejärjestelmän avulla pyritään madaltamaan sovelluksen käytön oppimiskynnystä ja vähentämään käyttäjien muistin kuormitusta. Ohjejärjestelmät voivat jopa konkreettisesti auttaa ja ohjata käyttäjän toimintaa oikeaan suuntaan. Myös monet manuaaliset ja puuduttavat työtehtävät, kuten parametrien laskeminen ja syöttäminen laitteille, voidaan tehdä jopa täysin automaattisesti.

Tämän työn tavoitteena on suunnitella edellä kuvatun kaltaisen teollisuusautomaatiojärjestelmän ohjaukseen käytettävään tietokonesovellukseen kontekstisensitiivinen ohjejärjestelmä, joka tarjoaa käyttäjälle ohjemateriaalia kulloinkin meneillään olevaan tehtävään. Tarkemmin sanottuna jokaiseen kohdesovelluksen näkymässä olevaan komponenttiin voidaan liittää siihen liittyvää ohjemateriaalia. Työ keskittyy erityisesti siihen, miten ohjemateriaali ja käyttöliittymäkomponentit saadaan linkitettyä toisiinsa, sekä millaisella mekanismilla ohjemateriaali näytetään. Oleellisena osana työhön liittyy myös ohjepro-

sessi, joka määrittää miten ohjemateriaalia kirjoitetaan, ja miten siitä tuotetaan ohjepaketti. Ohjeprosessi kattaa koko polun sovelluksen osien suunnittelemisesta aina niiden käytön ohjeistamiseen ja ohjemateriaalin näyttämiseen. Suunniteltavan ohjejärjestelmän tulee integroitua hyvin tähän suurempaan kokonaisuuteen. Työssä suunniteltava osuus ja ohjeprosessi kokonaisuudessaan on tarkemmin esitetty kuvassa 3.1. Kohdesovellus on toteutettu Qt-ohjelmistokehyksellä ja samoin ohjejärjestelmä suunnitellaan sitä käyttäen. Ohjejärjestelmään liittyvän ohjeprosessin alussa ohjemateriaali kirjoitetaan DITA XML -merkkauskielellä, jota käytetään teknisten dokumenttien ja käyttöohjeiden sisällön ja ulkoasun määrittelyyn. DITA XML -muotoiset ohjemateriaalit prosessoidaan sitten sopivaksi ohjepaketiksi, jota voidaan käyttää kohdesovelluksessa. Käytännön suunnittelutyö tehdään UML-kaavioita ja olio-ohjelmoinnin menetelmiä hyödyntäen.

Työ johdattelee lukijan aiheeseen ensin tarkastelemalla toteutusympäristöä ja tutkimusongelmaa yleisestä näkökulmasta sekä esittelemällä olemassa olevia ratkaisuja. Kun kokonaiskuva on luotu, asetetaan tavoitteet ja tarkastellaan työhön vaikuttavaa ohjeprosessia. Sen jälkeen käydään läpi varsinainen tekninen suunnittelu ja tarkastellaan työn tuloksia. Työ koostuu viidestä pääluvusta ja niiden rakenne on seuraavanlainen:

Luku 1 tarjoaa yleiskatsauksen koko työhön ja sen sisältöön.

Luku 2 sisältää yleiskuvauksen toteutusympäristöstä ja tutkimusongelmasta, taustatutkimusta aihepiiriin liittyen ja olemassa olevien ratkaisujen tarkastelua. Luvun lopuksi asetetaan työn konkreettiset tavoitteet.

Luku 3 sisältää kuvauksen ohjejärjestelmään liittyvästä ohjeprosessista ja käytettävistä työkaluista.

Luku 4 sisältää ohjejärjestelmän suunnittelun ja kuvauksen toteutettavasta ratkaisusta.

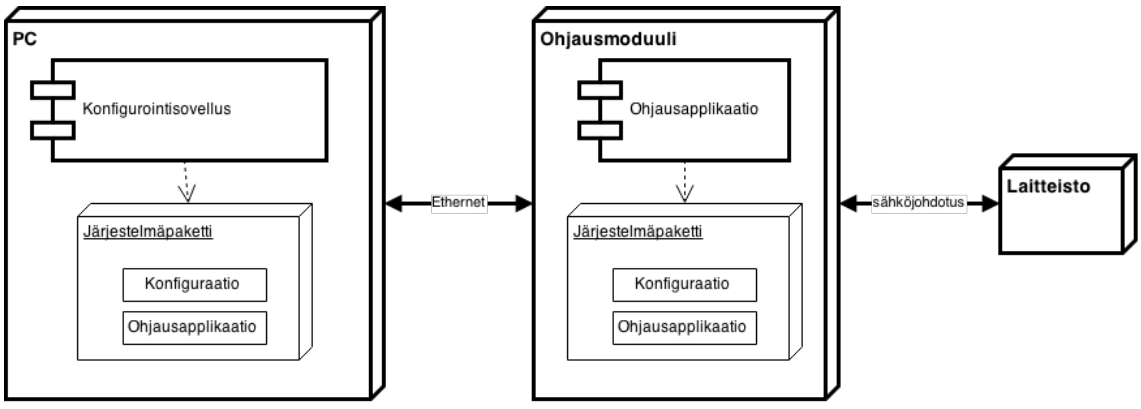
Luku 5 sisältää yhteenvedon ohjeprojektista ja lopputuloksen arviointia tarkastelemalla asetettujen tavoitteiden täyttymistä. Lopuksi esitellään myös jatkokehitysideoita ohjejärjestelmään liittyen.

2. YMPÄRISTÖ JA TAVOITTEET

Työhön liittyvällä ohjejärjestelmän kohdesovelluksella konfiguroidaan ja monitoroidaan automaatiojärjestelmää. Järjestelmään liittyy XML[1]-konfiguraatiotiedosto, joka kuvaa järjestelmän rakenteen, kytkennät ja käytettävät parametrit. Järjestelmän XML-kuvauksen perusteella kohdesovellukseen muodostetaan ajonaikaisesti dynaamisia käyttöliittymiä, joissa kuvataan järjestelmän säädettävät parametrit. Näiden näkymien avulla parametreja voidaan tarkastella, lisätä, poistaa ja muokata.

Automaatiojärjestelmä on yleiskäyttöinen, ja se konfiguroidaan lataamalla kohdesovelluksessa tuotettu järjestelmäpaketti järjestelmän moduuleille. Järjestelmäpaketti sisältää konfiguraation lisäksi moduuleilla ajettavat sovellusbinäärit, eli applikaatiot, jotka toteuttavat järjestelmän toiminnallisuuden. Applikaatiot ovat itsenäisiä sovelluksia, ja niiden toimintaa säädetään parametrien avulla.

Järjestelmän ollessa ajossa se käyttäytyy ajettavien applikaatioiden ja parametreissa säädettyjen arvojen perusteella. Kohdesovelluksella voidaan ottaa yhteys verkon yli ajossa olevaan järjestelmään ja monitoroida sen toimintaa, sekä säätää parametrien arvoja. Toteutettavan ohjejärjestelmän on tarkoitus kohdesovelluksen käyttöohjeiden lisäksi tarjota ohjeistusta järjestelmän parametrien säätämiseen. Automaatiojärjestelmän rakenne on esitetty yksinkertaistettuna kuvassa 2.1.



Kuva 2.1. *Automaatiojärjestelmän rakenne.*

Lähestytään käytössä olevaa automaatiojärjestelmää ja ohjejärjestelmän kohdesovellusta

[1]XML (Extensible Markup Language) on yksinkertainen, erittäin joustava tekstiformaatti, joka pohjautuu SGML-standardiin. Se on suunniteltu vastaamaan laajamittaisen elektronisen julkaisemisen haasteisiin, mutta sitä käytetään myös enenevässä määrin tiedonsiirtoon internetissä ja muualla. [22]

esimerkin kautta: suunnitellaan automaattinen kukankastelujärjestelmä. Kukankastelujärjestelmän olennaisimpana osana on fyysinen ohjausmoduuli, joka on itsenäinen sulautettu järjestelmä. Ohjausmoduulin lisäksi tarvitaan kukkaruukkuun kosteusanturi, vesipumppu, pinnankorkeusanturi vesisäiliöön ja merkkivalo ilmoittamaan vesisäiliön veden olevan lopussa. Kun laitteisto on kytketty, tarvitsee järjestelmään kehittää konfiguraatio ja ohjausapplikaatio. Konfiguraatio on järjestelmäkuvaus, jossa on määritelty ohjausmoduulista käytettävät sisään- ja ulostuloliitännät sekä järjestelmän säädettävät parametrit. Konfiguraatio siis sovittaa yleiskäyttöisen ohjausmoduulin käytettävään laitteistoon parametrien avulla, jotka ovat käytännössä asetuksia ja nimettyjä arvoja. Ohjausapplikaatio on ohjausmoduulissa ajettava erillinen sovellus, joka ohjaa järjestelmän toimintaa. Ohjausmoduulissa suoritettavia ohjelmia kutsutaan applikaatioiksi kyseisessä automaatiojärjestelmässä.

Kukankastelujärjestelmän toteutus aloitetaan käyttämällä konfigurointisovellusta järjestelmäpaketin kehittämiseen. Konfigurointisovellus on PC-työpöytäsovellus, joka on yhteydessä ohjausmoduuliin verkon yli. Konfigurointisovellus tarjoaa käyttöliittymiä parametrien määrittelyyn ja niiden arvojen asettamiseen. Kukankastelujärjestelmässä ohjausmoduuliin on laitteistoksi liitetty kosteusanturi, pinnankorkeusanturi, merkkivalo ja vesipumppu. Näiden tietojen pohjalta ohjelmoidaan ohjausapplikaatio, joka määrittää ohjelmakoodina miten kastelujärjestelmä toimii. Se voisi toimia esimerkiksi niin, että kukkaruukun kosteuden saavuttaessa alarajan käynnistetään vesipumppu, ja pidetään se aktiivisena 20 sekunnin ajan. Jos vesisäiliössä vedentaso on alle määritetyn alarajan, sytytetään merkkivalo ja pidetään se aktiivisena, kunnes vettä lisätään säiliöön. Edellä mainitut parametrit (kosteuden alaraja, vesipumpun aktivaatioaika ja vesitason alaraja) lisätään myös järjestelmän konfiguraatioon, jotta näitä parametreja voidaan käyttää ja säätää ajonaikaisesti.

Konfiguraation ja ohjausapplikaation ollessa valmiita, voidaan niistä muodostettu järjestelmäpaketti ladata verkon yli ohjausmoduulille, jolloin järjestelmä on käyttövalmis. Latauksen jälkeen järjestelmä aktivoituu, jolloin se alkaa suorittaa mittauksia ja ohjausta ohjausapplikaation ja sen parametrien perusteella. Tällöin konfigurointisovelluksella voidaan ottaa yhteys järjestelmään ja monitoroida sen toimintaa ajonaikaisesti. Esimerkiksi kukkaruukun kosteusanturin mittausarvoja voidaan seurata ja aktivaatiorajoja säätää. Järjestelmän ollessa ajossa sekä konfigurointisovellus että ohjausmoduuli käyttävät järjestelmäpaketissa olevaa konfiguraatiota. Konfigurointisovellus esittää käyttäjälle konfiguraatiossa olevat parametrit ja ohjausapplikaatio tarvitsee niitä ohjelmansa suorittamiseen.

Edellä olevassa esimerkissä on hyvin kattavasti kuvattu työssä esillä olevan automaatiojärjestelmän toiminnallisuus ja käyttötapa korkealla tasolla. Esiteltyyn automaatiojärjestelmään on suunnitteilla ohjeprosessi, jonka avulla tuotetaan ohjemateriaalia järjestelmän käyttäjille. Ohjeprosessi määrittää miten lähdemuotoisista ohjemateriaaleista muodoste-

taan ohjepaketti. Yksittäinen ohjepaketti sisältää automaatiojärjestelmän tietyn kokoonpanon ohjemateriaalit, ja se dokumentoi kyseisen järjestelmän toiminnallisuuden ja säädettävät parametrit. Osana ohjeprosessia konfigurointisovellukseen toteutetaan ohjejärjestelmä, joka pystyy lukemaan ohjepakettia. Ohjepaketin sisältöä pystyy tarkastelemaan konfigurointisovelluksesta, jonka käyttöliittymäkomponentit voidaan linkittää niihin liittyviin ohjesivuihin. Tällä mahdollistetaan ohjemateriaalin tehokas etsiminen sovelluksen käyttäjille. Ohjejärjestelmä on toistaiseksi rajattu toteutettavaksi ainoastaan konfigurointisovellukseen, mutta sitä voitaisiin hyödyntää myös muissa automaatiojärjestelmän osissa. Jatkossa konfigurointisovelluksesta käytetään myös termiä ohjejärjestelmän kohdesovellus.

Tarkemmin sanottuna tässä työssä suunnitellaan miten Qt-ohjelmistokehyksen tukema ohjepaketti saadaan ladattua ja integroitua konfigurointisovellukseen. Qt-kehys mahdollistaa ohjepaketin lataamisen käyttöön, mutta ei ota kantaa miten erillään kirjoitettu ohjemateriaali linkitetään käyttöliittymäkomponentteihin. Konfigurointisovellus sisältää kaikille automaatiojärjestelmän kokoonpanoille yhteisen sovellusrungon, jonka lisäksi sovelluksessa on dynaamisia kokoonpanokohtaisia näkymiä. Ohjemateriaalia täytyy voida linkittää sekä käyttöliittymärungon komponentteihin että dynaamisiin näkymiin. Esimerkiksi sovelluksessa on yhteinen kirjautumisnäkymä, kun taas sovelluksen näkymävalikon vaihtoehdot muuttuvat käyttäjätason ja käytettävän järjestelmäkokoonpanon mukaan. Erityisesti vielä dynaamisten näkymien käyttöliittymät muodostetaan applikaatioiden parametrien perusteella. Näissä dynaamisissa näkymissä applikaatioiden parametrien arvoja voidaan muokata. Qt-kehyksen ohjejärjestelmä tarjoaa avainsanamekanismin, jonka avulla ohjemateriaalia voidaan linkittää, mutta avainsanat on jotenkin saatava myös lähdemateriaalin ja käyttöliittymän tietoon. Linkitys on oleellinen osa työtä ja näyttelee suurta osaa järjestelmän suunnittelussa. Vaikka Qt-kehys tarjoaakin valmiita luokkia ohjepaketin lataamiseen, on kyseessä silti ohjelmistokehys ja varsinainen toteutus täytyy suunnitella kohdesovellukseen ja ohjeprosessiin sopivaksi.

2.1 Taustatutkimus

Ohjejärjestelmät yleisesti ottaen ovat tietokonesovelluksia, jotka tarjoavat käyttäjille avustusta ja ohjeita suurempaan sovellukseen tai tietojärjestelmään. Ohjejärjestelmät voivat olla varsinaiseen sovellukseen integroituja osia tai täysin erillisiä sovelluksia. Ohjejärjestelmät koostuvat käyttöliittymästä ja sisällöstä. Käyttöliittymä määrittää muun muassa miten viestit ja ohjemateriaali esitetään, miten käyttäjä pääsee käsiksi materiaaliin ja miten ohjejärjestelmän toiminnallisuutta tai kattavuutta voidaan lisätä. Sisältö tarkoittaa ohjejärjestelmän varsinaista tietosisältöä. Molemmat osat ovat yhtä tärkeitä ja kumman tahansa puutteet voivat tehdä ohjejärjestelmästä hyödyttömän. [9]

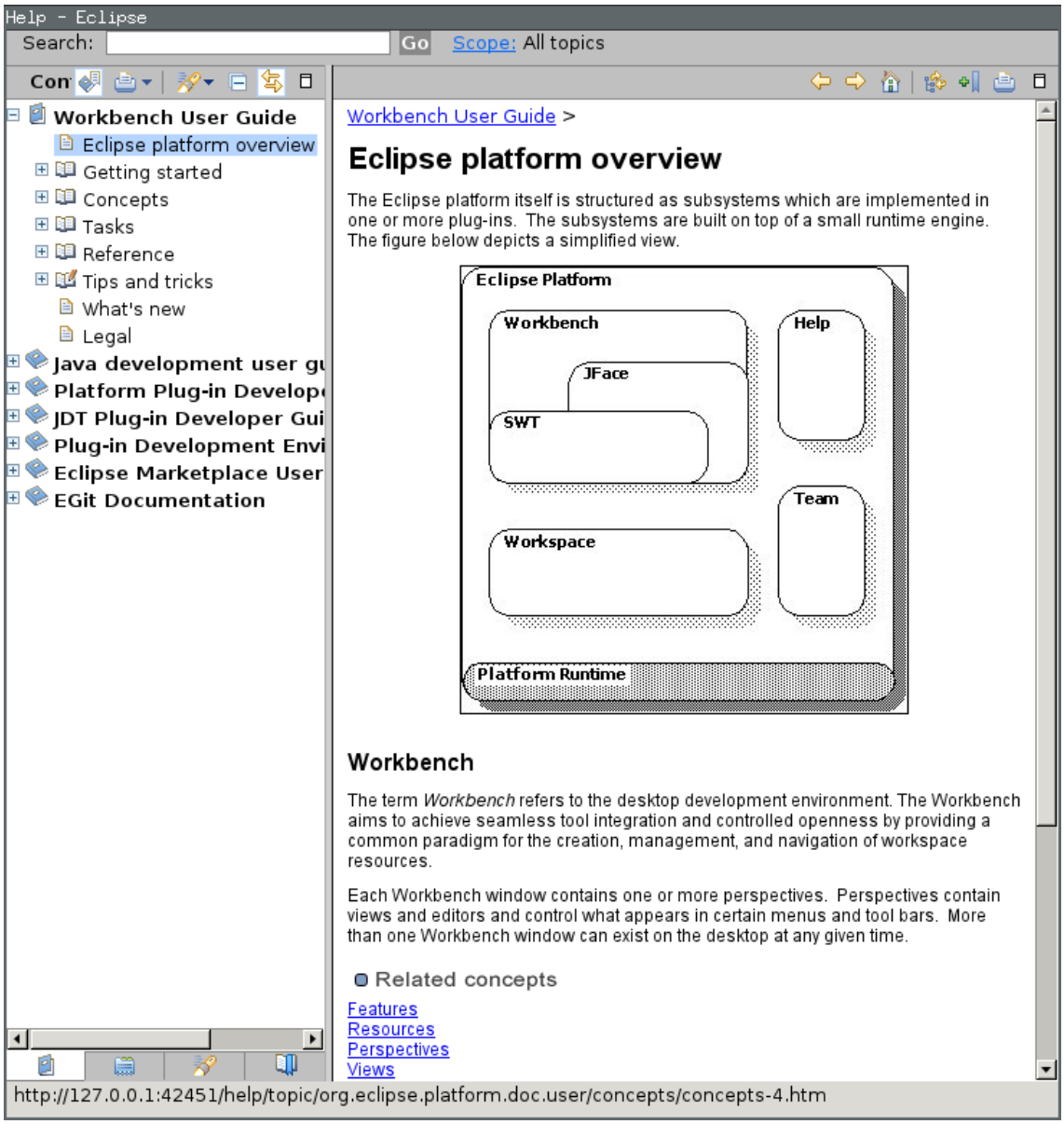
Kontekstisensitiivinen ohjejärjestelmä tarjoaa ohjemateriaalia ja informaatiota kulloisestakin työvaiheesta ja käytettävästä sovelluksen toiminnosta [9]. Järjestelmän toteutuksesta riippuu kuinka tarkasti eri ohjelman osat ja käyttöliittymän komponentit on eroteltu toisistaan, sekä minkälaisia ohjeita käyttäjälle tarjotaan. Yksinkertaisessa tapauksessa kontekstisensitiivinen ohje voi yhdistää sovelluksen näkymän ja siihen liittyvän ohjesivun tai -sivuston. Hienostuneemmassa ratkaisussa ohjemateriaalia ja vihjeitä voidaan tarjota käyttäjälle ilman, että käyttäjä eksplisiittisesti aktivoi ohjetoimintoa. Lisäksi sovellus voi interaktiivisesti opastaa käyttäjää, tai tehdä jopa tiettyjä manuaalista työtä vaativia toimintoja, täysin ilman käyttäjän vuorovaikutusta. Tällaisella järjestelyllä käyttäjää voidaan ohjata tehokkaampiin tai helpompiin toimintatapoihin ja vähentää käyttäjän työkuormaa, mutta tällainen toiminnallisuus on työlästä toteuttaa. [11]

Toteutettavan ohjejärjestelmän täytyy tarjota yhtenäinen toimintatapa ohjemateriaalin käyttämiseen sovelluksessa ja tähän kiinnitetään huomiota. Laajoissa järjestelmissä käytettyjen toteutustapojen monimuotoisuus asettaa haasteita jälkikäteen lisättäville ominaisuuksille. Tässä tapauksessa ohjejärjestelmä lisätään jälkikäteen laajaan ja monimuotoiseen sovellukseen. Tarvitaan mekanismi, joka on mahdollisimman yleiskäyttöinen sovelluksen sisällä, eikä sitä tarvitse erikseen sovittaa järjestelmän eri osiin. Kontekstisensitiivisiä ohjejärjestelmiä on olemassa runsaasti ja seuraavaksi tarkastellaan olemassa olevia ratkaisuja, jotka jossain määrin soveltuvat toteutettavan järjestelmän vaatimuksiin. Ohjejärjestelmät toteutetaan yleensä palvelemaan tiettyä rajattua tarkoitusta tai sovellusta. Tämänkin työn tapauksessa ohjejärjestelmä on rajattu tarjoamaan samana pysyvään konfigurointisovellukseen ja järjestelmän mukaan muuttuviin applikaatioihin ohjemateriaalia. Erinäisten ohjejärjestelmien toteutuksista voidaan kuitenkin yleistää joitain toistuvia ratkaisumalleja.

Ensimmäisenä esimerkkinä tarkastellaan Windows-käyttöjärjestelmän mukana tulevaa ohjejärjestelmää. Aikaisemmissa Windows-versioissa oli käytössä RTF-asiakirjaformaattiin (Rich Text Format) perustuva ohjejärjestelmä, mutta nykyisissä versioissa, Vista-versiosta eteenpäin, on käytössä HTML-pohjainen ohjejärjestelmä. Windows-käyttöjärjestelmä tarjoaa oman kontekstisensitiivisen ohjejärjestelmänsä sovellusten käyttöön, jonka avulla ne voivat tarjota omaa ohjemateriaaliansa käyttäjille. Ohjemateriaali kirjoitetaan HTML-sivuina, joista muodostetaan Microsoft Compiled HTML Help-tiedosto, joka on tiedostopäätteeltään .chm. Ohjepakettia voidaan jaella sovelluksen asennuspaketin mukana tai se voidaan ladata verkosta. [12] Windows-käyttöjärjestelmän ohjejärjestelmä, kuten myös monet muut ohjejärjestelmät, käyttää F1-näppäintä ohjeen aktivoimiseen. F1-näppäimen painallus avaa Windows-käyttöjärjestelmän ohjeselainsovelluksen, ja aina kun mahdollista nykyiseen näkymään liittyvän ohjesivun.

Eclipse-ohjelmistokehitysympäristön kontekstisensitiivinen ohjejärjestelmä on toinen

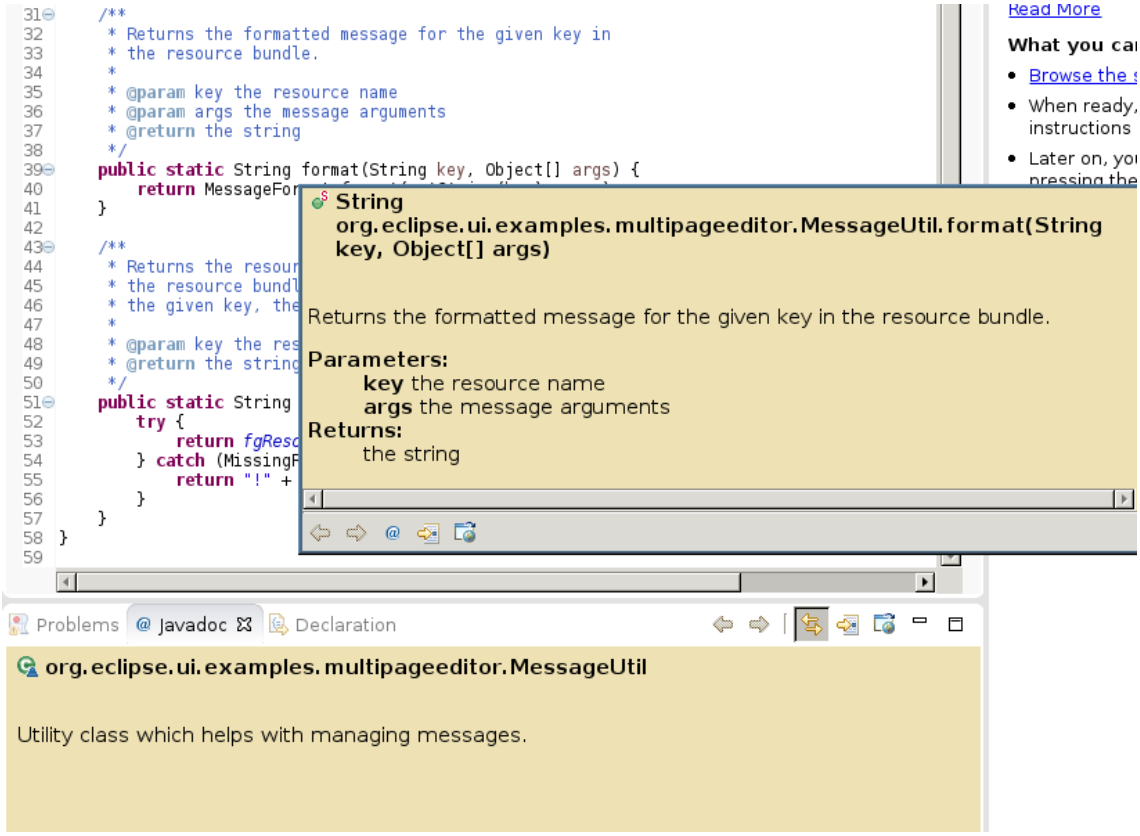
esimerkki tavoitteenmukaisesta järjestelmästä. Eclipse on käyttöjärjestelmäriippumaton Java-pohjainen alusta, joka on tarkoitettu applikaatioiden kehittämiseen ja ajettavaksi ohjelmabinääriksi kääntämiseen. Sen vahvuus pohjautuu liitännäisarkkitehtuuriin, joka mahdollistaa integroidun toiminnallisuuden nopean lisäämisen. [5] Eclipsen ohjejärjestelmä tarjoaa sekä sovelluksen käyttöohjeen että työkaluja kehitettävän ohjelmakoodin dokumentaation tarkasteluun.



Kuva 2.2. *Eclipse-ohjelmistokehitysympäristön ohjeselain. Näkyvissä Eclipse-kehitysympäristön käyttöohje [5].*

Kuvassa 2.2 on näkyvissä Eclipsen ohjeselain, jossa on avoinna Eclipsen käyttöohje. Ohjeselaimesta löytyy verkkoselaimista tutut toiminnot yläreunasta, kuten sivuhistoriassa liikkuminen, kotisivun avaaminen ja tulostaminen. Vasemmassa reunassa on kuitenkin verkkoselaimista poiketen ohjemateriaalin sisällysluettelo. Ikkunan yläreunassa on myös

hakukenttä, jonka avulla voidaan etsiä ohjesivuja hakutermin perusteella. Ohjeselain saadaan avattua valitsemalla se valikosta tai klikkaamalla ohjeselaimen avauspainiketta, kun kontekstisensitiivinen ohje on aktiivisena. Kontekstisensitiivinen ohjetila aktivoidaan valitsemalla Help-valikosta dynamic help -vaihtoehto, joka avaa Eclipsen päänäkymän oikeaan laitaan käyttäjän toimien mukaan sivuaan vaihtavan ohjeikkunan.



Kuva 2.3. *Eclipse-ohjelmistokehitysympäristön ohjelmakoodidokumentaation tarkastelutyökaluja. Näkyvissä Eclipsen mukana jaeltava MultiPageEditor-esimerkkisovellus [5].*

Kuvassa 2.3 on näkyvissä Eclipsen tarjoama ohjemekanismi ohjelmakoodin dokumentaation tarkasteluun. Eclipsessä koodidokumentaatio aktivoidaan siirtämällä hiiren osoitin ohjelmakoodissa avainsanan päälle. Tällöin aukeavassa pienessä työkaluohjeikkunassa on lyhyt ohje ja linkkejä tarkempien ohjeiden avaamiseen ohjeselaimessa. Kuvan keskellä on format-funktiota hiirellä osoitettaessa auennut pikaohje, sekä alalaidassa Javadoc-ohjeselain. Pikaohjeen painikkeiden avulla ohjesivu voidaan avata alareunan ohjeselaimeen tai ulkoiseen verkkoselaimeen tarkasteltavaksi.

Käytännössä Eclipsen ohjejärjestelmä on toteutettu liittämällä käyttöliittymäkomponenteihin kontekstitunnisteita. Jokaisella käyttöliittymäkomponentilla täytyy olla sen yksilöivä kontekstitunniste, engl. context ID, jonka perusteella siihen liitetään ohjemateriaalia. Näistä tunnisteista muodostetaan kontekstitiedostoja, joissa tunnisteisiin liittyvät ohjeaiheet on eksplisiittisesti määritetty. Eclipsen toiminnallisuutta voidaan lisätä tekemällä

liitännäisiä, jotka voivat sisältää sekä toiminnallisuutta että ohjemateriaalia. [7] Toteutettavan ohjejärjestelmän kohdesovellus, samoin kuin Eclipse, pohjautuu liitännäisarkkitehtuuriin, joten siihen voidaan toteuttaa samankaltainen ohjejärjestelmä kuin Eclipsessä.

2.2 Suunnittelutyön tavoitteet

Ohjejärjestelmän toteuttamiseksi tarvitaan tapa yhdistää tuotettu ohjemateriaali ja sovelluksen konteksti toisiinsa. Tämän lisäksi tarvitaan mekanismi, jolla ohjemateriaali avataan, sekä ohjeselain, jolla ohjemateriaalia voi tarkastella. Eritellään seuraavaksi työn konkreettiset tavoitteet, joiden toteutumista voidaan tarkastella työn lopuksi.

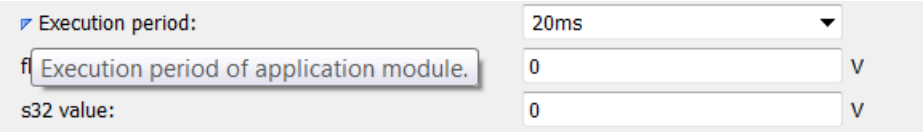
2.2.1 Ohjejärjestelmän perustoiminnallisuus

Ohjepaketti ladataan kohdesovelluksen käyttöön ajonaikaisesti, eikä kohdesovellus itsessään sisällä ohjemateriaalia. Ohjepaketti koostuu erillisistä tiedostoista, jotka toimitetaan kohdesovelluksen asennuspaketin mukana. Tällöin sovelluksesta riippumaton ohjepaketti voidaan vaihtaa ilman, että kohdesovellusta täytyy asentaa uudelleen.

Ohjesivuja tulee voida avata tarkasteltavaksi yksinkertaisesti F1-näppäintä painamalla ja klikkaamalla haluttua kohdetta kohdesovelluksen käyttöliittymässä. Tällöin hiiren kursorin alla olevan komponentin ohjemateriaalisivu avataan ohjeselaimeen, jos komponentille on saatavissa ohjemateriaalia. Lisäksi kaikkea saatavilla olevaa ohjemateriaalia voi selata kohdesovelluksen ohjeselaimessa. Ohjeselaimen vaatimukset on tarkemmin eritelty osiossa 2.2.3.

2.2.2 Käytettävyys

Käyttäjälle on selkeästi indikoitava milloin ohjemateriaalia on saatavilla. Sovelluksessa on jo käytössä työkaluvinkkejä, jotka antavat käyttäjälle lyhyitä vihjeitä kunkin käyttöliittymäkomponentin merkityksestä. Nämä vihjeet määritellään käyttöliittymäkomponentin luomisen yhteydessä, ja on järjestelmän kehittäjän vastuulla antaa merkityksellisiä vihjeitä. Työkaluvihjeen olemassa olo ilmaistaan käyttäjälle näyttämällä pieni sininen kolmioikoni parametrin nimen tai käyttöliittymäkomponentin vieressä, kuten kuvassa 2.4 on esitetty. Samassa kuvassa näkyy myös avoinna oleva työkaluvihje, joka näytetään kun hiiren kursori osoittaa kolmioindikaattoria, arvokentän nimeä tai arvokenttää.



Kuva 2.4. *Sininen kolmio parametrin nimen vieressä indikoi työkaluvihjettä.*

Ohjemateriaalin saatavuudelle tarvitaan samanlainen indikaatio kuin työkaluvihjeille, jotta käyttäjät näkevät suoraan käyttöliittymästä ilman mitään toimenpiteitä, mihin komponentteihin on saatavilla ohjemateriaalia. Ohjeikoni näytetään ainoastaan, jos käyttöliittymäkomponenttiin linkitetty ohjemateriaali on nykyisessä ohjepaketissa saatavissa. Pelkkä käyttöliittymäkomponentille asetettu viittaus ohjemateriaaliin ei vielä riitä ikonin näyttämiseen. Tästä seuraa, että aina kun ikoni on näkyvissä, niin ohjemateriaalia on saatavilla.

2.2.3 Ohjeselain

Ohjemateriaalia täytyy voida selata sovelluksesta, ja tätä varten suunnitellaan erillinen ohjeselainkomponentti. Ohjeselain avautuu, kun kontekstisensitiivinen ohje aktivoidaan tai käyttäjä valitsee sen valikosta. Koska ohjemateriaali on ohjepaketin sisällä XHTML-muodossa, tehdään myös ohjeselaimesta samantapainen kuin verkkoselaimista. Ohjeselaimessa on näkymä, jossa ohjemateriaalia voi tarkastella, avata linkkejä sekä palata edelliselle sivulle. Lisäksi näkyvissä on sisällysluettelo, josta voi suoraan siirtyä eri ohjesivuille. Ohjemateriaalissa esiintyvät linkit voivat olla materiaalin sisäisiä ristiviittauksia tai linkkejä ulkoiseen materiaaliin, kuten internetissä olevaan verkkosivuun.

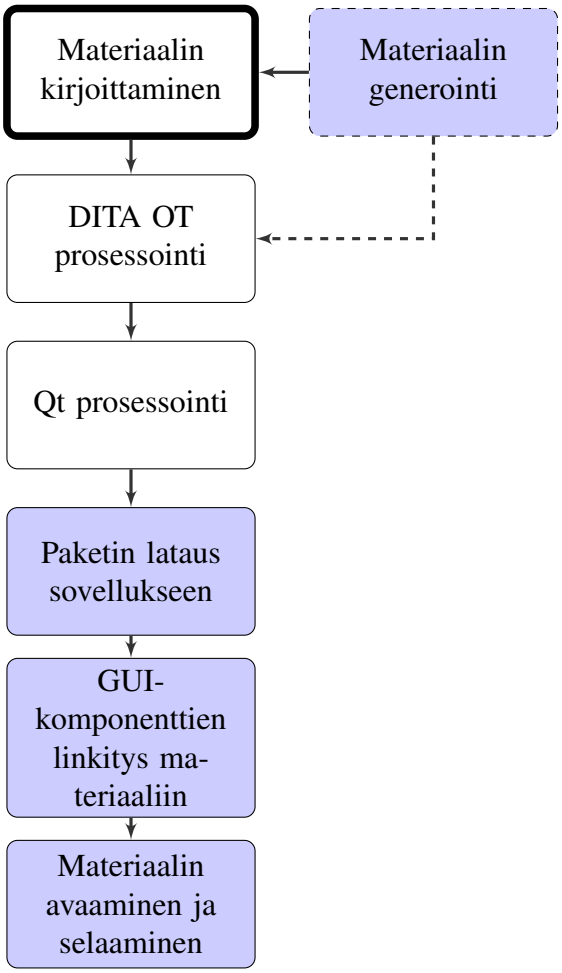
2.2.4 Käyttäjäryhmät

Kohdesovelluksessa on eri käyttäjätasoja, joilla on erilaisia oikeuksia nähdä ja muokata järjestelmäkonfiguraatioita. Järjestelmän kehittäjän ja loppukäyttäjän tarpeet ovat hyvin erilaiset, ja myös tietoturvan vuoksi kaikkea tietoa ei haluta sallia kaikille käyttäjätasoille. Tarvittavat käyttäjätasot määrittyvät suoraan kohdesovelluksen mukaan. Käyttöoikeustasot ovat kaikissa järjestelmissä samat, mutta samalla käyttäjällä käyttöoikeudet voivat vaihdella eri järjestelmien välillä. Käytännössä käyttöoikeudet jakaantuvat kahteen tasoon: muokkaus ja tarkasteluoikeuksiin. Järjestelmän kehittäjillä on muokkausoikeudet järjestelmäkonfiguraatioon, ja järjestelmän käyttäjillä on ainoastaan tarkasteluoikeudet. Molemmat käyttäjätasot voivat kuitenkin säätää ajonaikaisesti järjestelmän parametrien arvoja. Samaa ohjemateriaalia, ja sen osia, tulee voida käyttää eri käyttäjäryhmille suodatettuna. Tämä vaatimus täyttyy todennäköisesti jo ohjepaketin prosessointivaiheessa, mutta se täytyy kuitenkin huomioida ohjejärjestelmää suunniteltaessa.

3. OHJEPROSESSI JA TYÖKALUT

Perinteisesti automaatiojärjestelmän ohjemateriaali ja käyttöohjeet on kirjoitettu Microsoft Office Word -dokumentteina, jotka on muunnettu julkaistaessa PDF-asiakirjamuotoon. Tällöin kuitenkin jokaiselle käyttäjätasolle tarvitsee olla omat kohdennetut versionsa jokaisen eri järjestelmäpaketin materiaaleista. Tästä seuraa monia keskenään samankaltaisia dokumentteja, joita on kuitenkin ylläpidettävä erikseen.

DITA XML tarjoaa tähän ratkaisun, koska se mahdollistaa saman lähdemateriaalin kohdistamisen eri ryhmille. Ohjemateriaalin tuottaminen koostuu useista peräkkäisistä vaiheista, jotka on esitetty kuvassa 3.1. Luvussa 2 esitetyssä kukankastelujärjestelmäesimerkissä ohjeprosessi sijoittuu konfigurointivaiheiseen, kun ohjausapplikaatiota ollaan kehittämässä.



Kuva 3.1. *Ohjeprosessi kokonaisuudessaan. Työssä suunnitellut osat korostettu värillä.*

Kuten kuvasta 3.1 nähdään, ohjeprosessin alussa materiaalin tuottaminen voidaan aloittaa generoimalla kohdesovelluksella DITA XML -muotoiset rungot järjestelmän applikaatioiden parametreista ja niiden perustiedoista. Näitä DITA-runkoja kutsutaan myös DITA-aiheiksi, engl. DITA topic, joista jokainen on yleensä omassa tiedostossaan. Generoitu DITA-runko ei ole kuitenkaan riittävä määrä informaatiota, vaan aiherunkoja halutaan täydentää sisällyttämällä niihin tarkempia kuvauksia parametreista, niiden käyttötarkoituksesta ja ohjeita niiden virittämiseen. Lisäinformaatio aiherunkoihin kuuluu liittää sisällyttämällä se toisista tiedostoista, jolloin parametrien aiherungot voidaan uudelleen generoida ilman että lisätiedot katoavat. DITA XML tarjoaa aiheiden toisiinsa sisällyttämiseen valmiin mekanismin.

Materiaali kirjoitetaan erillisiin DITA-aiheisiin, jotka koostetaan sisällyttämällä ja linkittämällä niitä toisiinsa. Aiheista koostetaan myös ditamap-sisällysluettelo, jonka avulla saadaan muodostettua dokumenttimainen rakenne. Ohjeprojektissa käytetään ohjemateriaalin kirjoittamiseen ainoastaan englannin kieltä, mutta monikielinenkin dokumentaatio olisi mahdollista. Kun materiaali on kirjoitettu ja siitä on muodostettu ditamap-rakenne, voidaan se prosessoida DITA-OT-työkalujen avulla XHTML-muotoon. Tällöin dokumentti on verkkosivumuodossa, joka voitaisiin julkaista sellaisenaan internetiin tai lisätä sovelluksen asennuspakettiin paikallisesti verkkoselaimella selattavaksi.

XHTML-muotoisen materiaalin pohjalta voidaan seuraavaksi prosessoida valmis ohjepaketti Qt-kirjaston tarjoamilla työkaluilla. Prosessointia varten tarvitsee ensin generoida työkalujen tarvitsemat lähdetiedostot, jossa kuvataan halutun ohjepaketin sisältö ja rakenne. Nämä tiedostot on tarkemmin kuvattu osiossa 3.1.5. Prosessoinnin lopputuloksena on ohjepakettitiedosto ja ohjekokoelmatiedosto, jotka ovat tiedostopäätteiltään .qch ja .qhc. Käyttämällä Qt-kirjaston tarjoamia luokkia ohjemateriaalin käsittelemiseen ja esittämiseen voidaan prosessoitu ohjepaketti ladata ajonaikaisesti sovelluksen käyttöön, ja näyttää sen sisältämää ohjemateriaalia käyttäjälle. Kontekstisensitiivisen ohjetoiminnallisuuden toimimaan saamiseksi on vielä linkitettävä ohjemateriaalin aiheet käyttöliittymäkomponentteihin, jotta niitä aktivoidessa voidaan avata oikea ohjesivu. Kun ohjemateriaali on ladattu sovellukseen ja käyttöliittymäkomponentit on linkitetty niiden ohjesivuihin, voidaan ohjejärjestelmää alkaa käyttää. Käyttäjän aktivoidessa käyttöliittymäkomponentin ohjemekanismin, avataan ohjeselain oikealta sivulta. Tällöin käyttäjä voi tutkia ohjemateriaalia, sekä selata muita ohjeaiheita samalla kohdesovellusta käyttäessään.

3.1 Käytettävät työkalut

Ohjeprosessiin liittyy oleellisena osana eri työkaluja, jotka on määritelty kaikkien materiaalintuottajien käyttöön samoiksi. Työkaluja tarvitaan ohjemateriaalin kirjoittamiseen,

prosessointiin ja käyttämiseen. Ohjeprosessissa käytetyt työkalut on esitelty tässä osiossa.

3.1.1 DITA XML -merkkauskieli

Darwin Information Typing Architecture, lyhennettynä DITA, on XML-pohjainen arkkitehtuuri aihepohjaisen informaation tuottamiseen ja järjestämiseen. Se kattaa joukon dokumenttityyppejä, joita voidaan yhdistää, laajentaa ja rajoittaa. Informaatiotyypitettyä sisältöä voidaan uudelleenkäyttää, sekä se mahdollistaa yksilähteistämisen. Yksilähteistämisen avulla samasta lähdemateriaalista voidaan prosessoida useita eri dokumenttimuotoja. DITA on perinteisesti tarkoitettu suuren kokoluokan teknisiin dokumentteihin, mutta sitä voidaan käyttää myös mihin tahansa muihin julkaisuihin ja dokumentteihin. Koska DITA aiheet ovat XML-standardin mukaisia, niitä voidaan kirjoittaa ja muokata millä tahansa XML-työkalulla, joskin suurin hyöty saadaan käyttämällä DITA-standardia tukevia ja kirjoittamisessa avustavia työkaluja. [13]

DITA on suunniteltu uusien dokumenttityyppien luontiin ja niiden kuvaamiseen olemassa olevien dokumenttityyppien perusteella. Uusien dokumenttityyppien luomista kutsutaan spesialisaatioksi. Spesialisaation avulla voidaan luoda tarkkaan määriteltyjä ja tiettyyn käyttötarkoitukseen räätälöityjä dokumentteja, jotka edelleen voivat hyödyntää yleiskäyttöisiä transformaatioita eri loppudokumenteiksi. Mekanismi on hyvin samankaltainen kuin olioparadigmassa, jossa luokat voivat periä kantaluokkiensa ominaisuuksia. [13] Liitteessä A on esitelty DITA-muotoisia dokumentteja, joista esimerkkeinä käsitellään concept- ja reference-tyyppisiä aiheita. Nämä ovat abstraktin DITA-aiheen konkreettisia erikoistuksia, joita voitaisiin tarvittaessa erikoistaa edelleen.

Hyvä esimerkki DITA-järjestelmän tehokkuudesta on mahdollisuus käyttää käyttöliittymässä esiintyviä merkkijonoja suoraan ohjedokumenttipohjien luontiin. Tämä on mahdollista esimerkiksi Apachen Java-pohjaisessa Struts-ohjelmistokehyksessä, jonka käyttöliittymät kuvataan XML-tiedostoilla. Tämä kuvaus saadaan yhteensopivaksi DITA-järjestelmän prosessoinnin kanssa tekemällä XSLT-muunnos[1], jolloin käyttöliittymäkuvauksessa käytetyt merkkijonot voidaan suoraan prosessoida myös dokumentaatioon. [8, 10] Tällä on monia hyötyjä, joista merkittävin on se, ettei termejä ja käännettäviä merkkijonoja tarvitse ylläpitää monessa paikassa. Käytännössä tällöin käyttöliittymistä ja -komponenteista voidaan automaattisesti muodostaa DITA-muotoisia sivuja, joita voidaan sisällyttää ohjepakettiin.

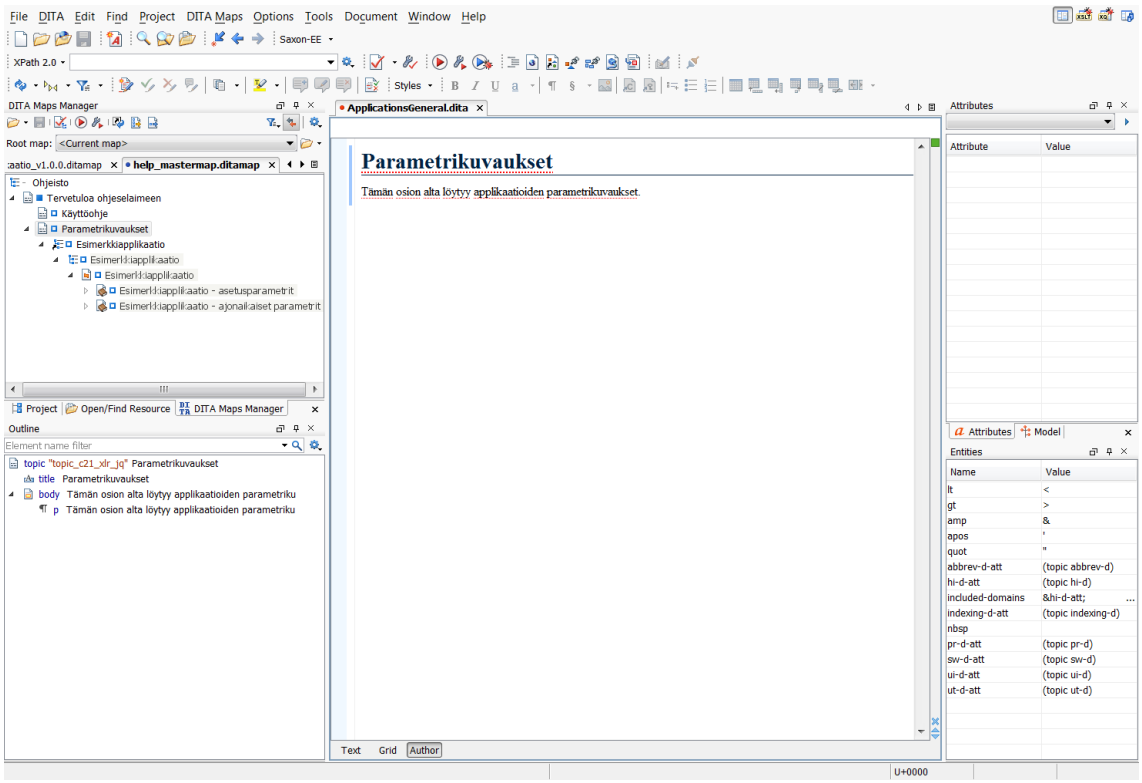
[1] XSLT on XML-dokumenttien muokkaukseen tarkoitettu kieli. Sillä voidaan muodostaa uusia XML-dokumentteja olemassa olevien pohjalta, mahdollisesti kuvaten ne uuteen muotoon tai suodattaen niiden sisältöä [23].

Tämä järjestely kuitenkin edellyttää, että käyttöliittymä on kuvattu XML-muodossa. Qt Creator -käyttöliittymäsuunnittelusovellus tallentaa käyttöliittymäkuvaukset XML-muodossa, .ui-päätteisinä tiedostoina, mutta kaikki kohdesovelluksen käyttöliittymät eivät ole suunniteltu Qt Creator -sovelluksella, vaan osa niistä generoidaan ajonaikaisesti. Koska kaikki käyttöliittymät eivät ole XML-muodossa, ei voida yleisenä ratkaisuna käyttää edellä kuvatun kaltaista XSLT-muunnosta DITA-ohjerunkojen tuottamiseen.

Useissa julkaisuissa on käytetty myös XSLT-muunnosta olemassa olevan dokumentaation tai muun aineiston muuntamiseen DITA XML -muotoon [2, 8, 10]. Tästä ei kuitenkaan ole hyötyä, jos ennalta olemassa olevaa lähdemateriaalia ei ole tai se on jossain binääriasiakirjamuodossa. Yleiset asiakirjamuodot, kuten Microsoft Office Word Document ja Portable Document Format, eivät ole sopivia dokumenttiformaatteja kyseiseen muunnokseen. Sen lisäksi etteivät ne ole XML-muotoisia, täytyy niiden rakenne lähtökohtaisesti uudelleensuunnitella, että niistä voidaan koostaa luontevia DITA-aiheita. Perinteisten tekstidokumenttien informaatiorakenne ei usein ole suoraan soveltuva DITA-aiheiksi, koska DITA-rakenteet ovat hyvin määriteltyjä ja aina samanlaisia, kun taas asiakirjat ovat hyvin monimuotoisia ja voivat sisältää luovia ratkaisuja. Ohjeprojektissa järjestelmän applikaatioiden parametreista generoitavat DITA-rungot on esitelty tarkemmin osiossa 4.1.6. Liitteessä A on myös käytännön esimerkkejä siitä millaisia DITA-elementtejä käytetään ohjemateriaaleissa.

3.1.2 oXygen XML -sovellus

XML-editorina projektissa käytetään oXygen XML Author -sovellusta, joka on järjestelmäriippumaton sovellus XML-dokumenttien muokkaukseen ja visualisointiin. Sovelluksen käyttäminen onnistuu ilman tarvetta kattavalle tietämykselle XML-standardista ja siihen liittyvistä teknologioista [21]. Sovellus tarjoaa kattavan tuen DITA-muotoisten dokumenttien käsittelyyn. Visuaalinen editointi mahdollistaa materiaalin tuottamisen samaan tapaan kuin tekstinkäsittelyohjelmistoissa, samalla kuitenkin tarjoten suuria etuja. Havainnollisen muokkauksen lisäksi materiaaleja voidaan helposti linkittää toisiinsa ja hyödyntää DITA-merkkauskielen mahdollistamia tekstin uudelleenkäytön mekanismeja. Sovellus sisältää myös työkaluja materiaalin prosessointiin kohdedokumenteiksi. [20]



Kuva 3.2. *Kuvaruutukaappaus oXygen XML Author -sovelluksesta.*

Kuvassa 3.2 on nähtävissä oXygen-sovellus, johon on avattuna ohjepaketin ylimmän tason ditamap-rakenne ja applikaatioiden pääsivu. Vasemmalla ylhäällä näkyy DITA Maps manager, joka helpottaa aiheiden keskinäisen rakenteen hallintaa. Siinä on sisällysluetteloa vastaava puurakenne, josta voidaan avata DITA-aiheita muokattavaksi. Sen alla oleva Outline-osio näyttää auki olevan DITA-aiheen elementtirakenteen. Tässä DITA-tiedostossa nähdään body-elementin sisällä olevan yksi p-tekstikappale-elementti. Elementtirakennetta vastaava visualisaatio on nähtävissä sovelluksen näkymän keskellä sisältöalueessa Author-välilehdellä. Text-välilehti näyttää XML-rakenteen tekstimuodossa ja Grid-välilehti näyttää hierarkkisen taulukkorakenteen sisällöstä. Keskellä olevassa sisältöalueessa sisältöä voidaan muokata samaan tapaan kuin tekstinkäsittelyohjelmistoissa, ja lisäksi kirjoittaessa sovellus tarjoaa täydennyksiä ja listan mahdollisista lisättävistä elementeistä. Oikean reunan ikkunat näyttävät lisätietoja tekstinmuokkauskursorin alla olevasta elementistä.

oXygen-sovellusta käytetään suunnitellussa ohjeprosessissa DITA-muotoisten ohjesivujen kirjoitustyössä. Se tarjoaa kattavat työkalut DITA-muotoisten ohjesivujen kirjoittamiseen ja ditamap-rakenteiden luontiin, joilla määritellään ohjeaiheiden keskinäinen rakenne. DITA-muotoisen tekstin käsittelyyn tarkoitetut työkalut helpottavat huomattavasti ohjemateriaalin kirjoitustyötä. Sama onnistuu tavallisellakin XML-muokkaimella tai jopa tekstieditorilla, mutta on luonnollisesti työläämpää. Sovellus mahdollistaisi myös kohdedokumenttien tuottamisen suoraan käyttöliittymän kautta, mutta Qt-ohjepaketti ei ole

valmiiksi tuettujen formaattien joukossa. Palaten luvussa 2 esiteltyyn kukankastelujärjestelmäesimerkkiin, oXygen-sovellusta käytetään järjestelmän konfigurointivaiheessa ohjemateriaalin kirjoittamiseen. Ohjemateriaalin kirjoittaa yleensä ohjausapplikaatioille ja kohdesovelluksen toiminnoille niiden kehittäjät. Ohjepaketin prosessoinnin jälkeen voidaan ohjemateriaalia hyödyntää monitorointivaiheessa parametrien säätämiseen ja ohjejärjestelmän kohdesovelluksen toimintojen käyttöohjeina.

3.1.3 DITA Open Toolkit -sovelluspaketti

DITA Open Toolkit, lyhennettynä DITA-OT, on joukko Java-pohjaisia avoimen lähdekoodin työkaluja, jotka mahdollistavat DITA-muotoisen lähdemateriaalin prosessoinnin valmiiksi dokumenteiksi. Työkalupaketti on ilmainen käyttää, ja se on lisensoitu käyttämään Apache lisenssin versiota 2.0. Mahdollisia prosessoitavia dokumenttimuotoja ovat esimerkiksi PDF, HTML5, XHTML, ODT ja RTF. [4]

DITA-OT pakettia käytetään ohjeprojektissa prosessoimaan DITA-muotoinen lähdemateriaali XHTML-sivuiksi. DITA-prosessoinnin yhteyteen on tehty myös skripti, joka muodostaa Qt-ohjepaketin prosessoinnissa tarvittavat asetustiedostot. Nämä asetustiedostot ovat tiedostopäätteiltään .qhp ja .qhcp, ja ne sisältävät nimet ja polut ohjepakettiin mukaan otettavista XHTML-tiedostoista. Osiossa 3.1.5 esitellään tarkemmin Qt Help -ohjelmistokehys, joka tarjoaa työkalut XHTML-tiedostojen prosessoimiseen valmiiksi Qt-ohjepaketiksi.

3.1.4 Qt-ohjelmistokehys

Qt on alustariippumaton sovellus- ja käyttöliittymäohjelmistokehys. Qt Creator -sovellus on sen mukana jaeltava kehitysympäristö. Qt Cloud -palvelu tarjoaa verkkosovellusten taustajärjestelmien toiminnallisuutta Qt-sovelluksille. [15] Qt on toteutettu ja sitä kirjoitetaan pääasiassa C++-ohjelmointikielellä, mutta sitä on mahdollista kehittää myös muilla ohjelmointikielillä, kuten python-skriptauskielellä. Qt tarjoaa monenlaisia valmiita käyttöliittymäkomponentteja ja yleiskäyttöisiä ratkaisuja, jotka helpottavat ja nopeuttavat sovellusten kehittämistä.

Projektin kohdesovellus on toteutettu Qt-ohjelmistokehystä hyödyntäen ja samoin ohjejärjestelmä suunnitellaan sitä käyttämään. Kehyksestä saadaan erityisesti hyötyä sen sisältämästä ohjejärjestelmäkehyksestä, joka on esitelty tarkemmin seuraavassa osiossa.

3.1.5 Qt Help -ohjelmistokehys

Qt-ohjelmistokehyksen ohjejärjestelmä, engl. Qt Help System, tarjoaa työkaluja ohjetiedostojen generointiin ja katseluun. Se tarjoaa myös luokkia, joilla ohjemateriaalin tarkastelu saadaan integroitua Qt-sovellukseen. Varsinainen ohjedata, tarkoittaen sisällysluetteloa, avainsanaindeksiä ja HTML-dokumentteja, sisältyy pakattuihin ohjetiedostoihin. Yleensä yksi tällainen ohjetiedosto sisältää yhden käyttöohjeen. Monesti kompleksisissa tuotteissa ohjemateriaali koostuu monista eri työkaluista ja niiden manuaaleista. Näiden kaikkien manuaalien tulisi olla saatavilla samanaikaisesti, ja ideaalisesti näiden manuaalien eri aiheiden välillä voisi olla ristiviittauksia. Tämän vuoksi Qt-kehyksen ohjejärjestelmä käyttää ohjekokoelmatiedostoja, jotka voivat sisältää monia eri ohjepaketteja. [19]

Qt-kirjaston ohjepaketti koostuu kokonaisuudessaan neljästä eri tiedostosta [19]:

- Ohjeprojekti (.qhp) — Sisältää sisällysluettelon, hakemiston ja viitteet varsinaisiin ohjetiedostoihin, jotka ovat XHTML-muotoisia tiedostoja.
- Ohjekokoelmaprojekti (.qhcp) — Sisältää viitteet pakattuihin ohjepaketteihin, jotka sisällytetään ohjekokoelmaan.
- Ohjepaketti (.qch) — Prosessoinnin ulostulo. Binääritiedosto, joka sisältää kaiken ohjeprojektitiedoston informaation sekä pakatut XHTML-ohjetiedostot.
- Ohjekokoelma (.qhc) — Prosessoinnin ulostulo. Sisältää viittaukset kaikkiin kokoelman sisältämiin pakattuihin ohjepaketteihin sekä mahdollisia lisätietoja, kuten sisältösuodattimia. Qt-sovelluksen on ladattava tämä tiedosto käyttöönsä, kun ohjemateriaalia halutaan käyttää.

DITA-prosessoinnin yhteydessä tuotetaan XHTML-ohjesivujen lisäksi .qhp ja .qhcp -asetustiedostot Qt-ohjepaketin prosessointia varten. Qt-kehyksen mukana saatavat qhelpgenerator- ja qcollectiongenerator-sovellukset tuottavat .qhc ja .qch-tiedostot, jotka ovat varsinainen sovellukseen ladattava ohjepaketti. Pohjimmiltaan ohjepaketit ovat SQLite-tietokantatiedostoja. Tästä johtuen ohjepaketteja voidaan tarkastella myös SQLite-työkaluilla. Tietokantatiedosto sisältää ohjetiedostot sekä sisällysluettelon organisoituna relaatioiksi.

Qt-kehyksen ohjejärjestelmä tarjoaa myös Qt Assistant -sovelluksen, jota voitaisiin käyttää ohjepakettien selaamiseen. Kuitenkin ohjeselain halutaan integroida kohdesovellukseen, ettei sovelluksen käyttäjien tarvitse asentaa ja käyttää erillistä sovellusta. Loppukäyttäjällä ei oleteta olevan Qt-ympäristöä asennettuna, vaan kaikki tarvittavat tiedostot toimitetaan kohdesovelluksen asennuspaketin mukana. Tämä tarkoittaa myös sitä, että käytettyjen jaettujen Qt-kirjastojen on sisällyttävä asennuspakettiin.

3.2 Vaihtoehtoisia työkaluja

Kuten jo aiemmin todettiin oXygen XML ei ole ainoa mahdollinen muokkain, jota DITA XML-dokumenttien kirjoittamiseen voidaan käyttää. Kaupallisia vaihtoehtoja on lukuisia, mutta avoimen lähdekoodin vastaavia vakavasti otettavia vaihtoehtoja ei löytynyt [1, 3]. Eräs harkinnan arvoinen kaupallinen vaihtoehto oXygen XML -sovellukselle on DITA Exchange -sovelluslisäosa, joka toimii suoraan Microsoft Office Word -tekstinkäsittelyohjelmassa. DITA Exchange mahdollistaakin jo yleisesti yrityksissä olevan ohjelmiston käyttämistä DITA XML -dokumenttien tuottamiseen.

Vaihtoehtoja löytyy myös merkkauskielten osalta. Hyvin samanlaista toiminnallisuutta kuin DITA XML tarjoaa LaTeX, jota on jo pitkään käytetty tieteellisten tutkielmien ja dokumenttien julkaisussa. LaTeX on ladontajärjestelmä, jota käytetään useimmiten keskisuurten ja suurten teknisten dokumenttien kirjoitustyössä. Sitä voidaan käyttää lähes millaisten tahansa tekstidokumenttien tuottamisessa ja julkaisemisessa. LaTeX ei kuitenkaan ole tekstinkäsittelyohjelma, vaan sitä käytettäessä keskitytään sisällöntuottamiseen ja muotoilut asetetaan erikseen. Tällöin kirjoittajien ei tarvitse huolehtia dokumenttinsa ulkoasusta kokoajan, vaan he voivat keskittyä oleelliseen. LaTeX perustuu ideaan, että on parempi jättää dokumentin muotoilu graafisten suunnittelijoiden huoleksi ja antaa kirjoittajien keskittyä dokumenttien sisältöön. [14] LaTeX mahdollistaa yhtä lailla aiheiden jakamisen eri tiedostoihin ja niiden keskinäisen koostamisen sisällyttämällä tiedostoja toisiinsa.

Luvun alussa kuvattu ohjeprosessi mahdollistaakin sen osien vaihtamisen ilman, että koko prosessi häiriintyy. Esimerkiksi DITA XML -kieli ja DITA OT -työkalupaketti voitaisiin korvata vastaavalla LaTeX-prosessoinnilla, joka edelleen tuottaisi XHTML-muotoisia tiedostoja Qt-ohjepaketin prosessointia varten. Tämä vaikuttaisi luonnollisesti myös ohjemateriaalirunkojen generointivaiheeseen ja ohjepaketin prosessointiin tarvittavien asetustiedostojen luomiseen. Muutokset olisivat kuitenkin rajallisia, eivätkä vaikuttaisi koko ohjeprosessiin. Materiaalin kirjoittaminen olisi tosin täysin erilaista, ja tällaisen muutoksen tekeminen jälkikäteen aiheuttaisi varmastikin suuria kustannuksia henkilöstön kouluttamisessa ja uusien työkalujen käyttöönotossa. Muutenkin kirjoittajien kouluttamisessa suurin osa haasteesta on todennäköisesti työkalujen käytön ja aihepohjaisen materiaalirakenteen kanssa, eikä niinkään DITA-muotoisten elementtien ja niiden ominaisuuksien käytössä [2].

4. OHJEJÄRJESTELMÄN SUUNNITTELU

Tässä luvussa käydään läpi toteutettavan järjestelmän rakenne. Erityisesti huomiota kiinnitetään osiossa 2.2 esitettyihin tavoitteisiin. Osiossa 2.1 käsiteltiin olemassa olevia toteutuksia samankaltaisen ongelman ratkaisemiseksi. Taustatiedon pohjalta täytyy toteutettava ratkaisu vielä suunnitella ja sovittaa käyttämään Qt-ohjelmistokehystä sekä sen tarjoamia työkaluja.

4.1 Tavoitteiden toteuttaminen

Ohjejärjestelmästä tehdään dynaamisesti linkitetty kirjasto (DLL). Ohjejärjestelmä saadaan tällöin itsenäiseksi kokonaisuudeksi, joka voidaan ottaa käyttöön tarvittaessa ajonaikaisesti. Ohjejärjestelmän kohdesovellus, eli automaatiojärjestelmän konfigurointisovellus, toteuttaa liitännäisarkkitehtuuria. Siksi on varsin luonnollista tehdä myös ohjejärjestelmästä liitännäinen. Kohdesovelluksen toiminnallisuus on jaettu lähes täysin liitännäisiin, jonka ansiosta sovelluksen ydin on pienikokoinen. Ytimen oleellisin toiminnallisuus on liitännäistenhallinta, jonka avulla sovellus voi ladata ajonaikaisesti dynaamisia kirjastoja käyttöönsä, ja siten laajentaa toiminnallisuuttaan. Tämä käytäntö myös minimoi sovelluksen muistijalanjäljen, koska dynaamisia kirjastoja, joiden toiminnallisuutta ei käytetä, ei tarvitse ladata ollenkaan muistiin. Lisäksi tästä on hyötyä sovelluksen modulaarisuudelle, koska liitännäiset voidaan vaihtaa tai refaktoroida täysin, kunhan niiden rajapinnat pysyvät samoina. Käytännössä liitännäistenhallinta käyttää QLibrary-luokkaa dynaamisten kirjastojen ajonaikaiseen lataamiseen. Qt Help -ohjejärjestelmäkehys on ohjeliitännäisen tapaan myös dynaamisesti linkitetty kirjasto, joten sen .dll-tiedoston tulee olla ajonaikaisesti saatavilla.

Kohdesovelluksen kehitysympäristönä käytetään Microsoftin Visual Studio -ohjelmistoa, joka tuottaa dynaamisesti linkitetylle kirjastolle .lib ja .dll -päätteiset tiedostot. Näistä .dll-tiedosto on varsinainen dynaaminen kirjasto, joka tulee olla saatavilla kohdesovellusta käytettäessä. Toinen, .lib-tiedosto, on staattinen kirjasto, joka pystyy lataamaan dynaamisen kirjaston sovelluksen käyttöön. Sen avulla dynaamisen kirjaston funktioita voidaan kutsua tavallisten staattisen kirjaston funktioiden kautta. Ohjeprojektissa ei kuitenkaan tätä .lib-tiedostoa käytetä, vaan dynaamisen kirjaston lataaminen hoidetaan manuaalisesti.

Vertailun vuoksi mainittakoon, että Qt-kehyksen mukana tuleva Qt Creator -

kehitysympäristö käyttää MinGW-kääntäjää oletuksena Windows-ympäristössä, joka tuottaa .a ja .dll-tiedostot. Näistä .a-tiedosto on staattisesti linkitetty kirjasto. [16] Qt Creator tarjoaisi kaikki tarvittavat toiminnot ohjejärjestelmän kehittämiseen, ja sitä voitaisiin käyttää Visual Studion sijaan.

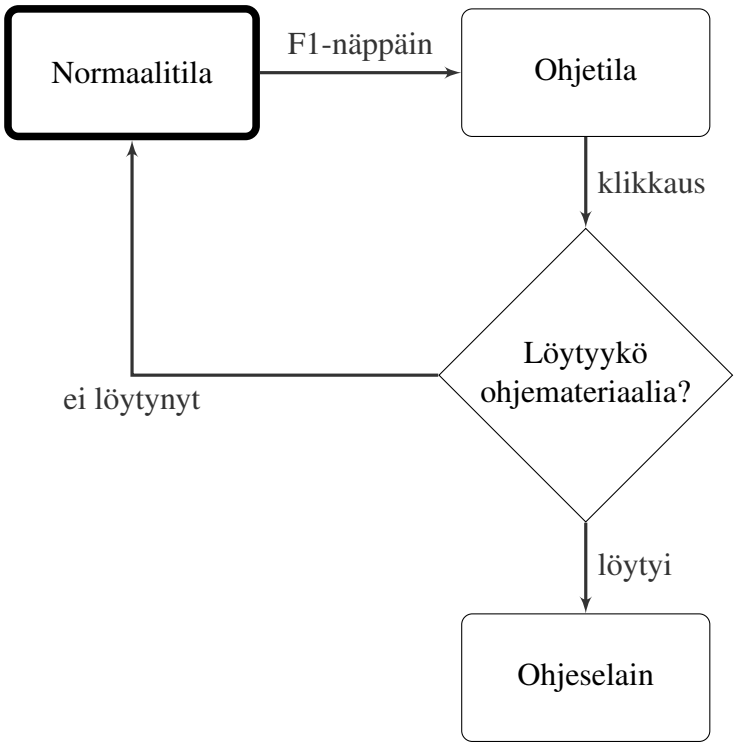
Ohjeliitännäinen tarjoaa IHelp-rajapinnan muun sovelluksen käyttöön, ja sen kautta liitännäisen toteuttamaa toiminnallisuutta voidaan kutsua. Kyseinen rajapinta ja sen tarjoama toiminnallisuus on esitelty tarkemmin osiossa 4.3.1.

4.1.1 Perustoiminnallisuus

Osiossa 2.1 todettiin ohjejärjestelmien usein käyttävän F1-näppäintä ohjeen avaamiseen. Toteutettavaan ohjejärjestelmään päätettiin kuitenkin käyttää F1-näppäimen painalluksen lisäksi hiirenklikkausta kontekstisensitiivisen ohjeen aktivointiin. Vaikka käyttöliittymässä on vihjeitä, mille komponenteille on saatavissa ohjemateriaalia, ei tämä kuitenkaan yksiselitteisesti kerro käyttäjälle miten toiminto kohdistetaan. Monesti kohdesovellusta käytettäessä käyttöliittymäkomponentti, josta halutaan ohjemateriaalia, ei ole kohdistettuna, jolloin F1-näppäimen painallus avaisi jonkin muun käyttöliittymäkomponentin ohjesivun. Onkin yksiselitteisempää F1-näppäimellä ohjeen avaamisen sijaan aktivoida ohjetila, jolloin hiirenkursori aktiivisesti osoittaa missä komponenteissa on ohjemateriaalia saatavissa, ja vasta hiirellä klikattaessa avata ohjesivu. Tällöin käyttäjä saa helposti kohdistettua ohjepyynnön haluamaansa komponenttiin, eikä käyttäjä joudu ihmettelemään miten komponentti tulisi valita. F1-näppäimellä aktivoitu ohjetila saadaan peruttua, avaamatta ohjeselainta, painamalla ESC-näppäintä.

Ohjeliitännäiseen toteutetaan tapahtumasuodatin, engl. event filter, jolla vastaanotetaan F1-näppäimen painallus ja hiiren painikkeen klikkaus. Tapahtumat ovat Qt-kehyksessä objekteja, jotka on periytetty QEvent-luokasta. Tapahtumat kuvaavat sekä sovelluksen sisällä että ulkoisesti tapahtuneita asioita, joista sovelluksen tulee saada tieto. Tapahtumia voidaan ottaa vastaan mihin tahansa QObject-luokasta perittyyn aliluokkaan, mutta niitä tarvitaan erityisesti widgeteissä vastaanottamaan käyttäjän syötettä. Widgetit, engl. widgets, ovat käyttöliittymään upotettavia pienoisohjelmia, eli toiminnallisia käyttöliittymäkomponentteja. Qt-kehys toimittaa tapahtumat tapahtumankäsittelijöiden käsiteltäviksi, jotka ovat kantaluokalta perittyjä virtuaalisia funktioita. Virtuaalifunktioiden rajapinta pysyy periytettäessä samana, mutta niiden toiminnallisuutta voidaan tarpeen mukaan erikoistaa, jolloin tapahtumat voidaan käsitellä halutulla tavalla. [16] Periytetty luokka voi uudelleentoteuttaa kantaluokkansa tapahtumankäsittelijän tai luokka voi tarvittaessa rekisteröidä täysin uuden tapahtumasuodattimen. QObject-luokka tarjoaa installEventFilter-funktion tapahtumasuodattimien rekisteröimiseen.

Ohjejärjestelmässä käytetään tapahtumasuodatinta vastaanottamaan näppäimien painalluksia ja hiiren painikkeiden klikkauksia, joiden perusteella siirrytään kontekstisensitiiviseen ohjetilaan ja avataan ohjeselain. Kyseessä on siis tilakone, jonka toiminta on selvennetty kuvassa 4.1.



Kuva 4.1. *Tapahtumasuodattimen muodostama tilakone.*

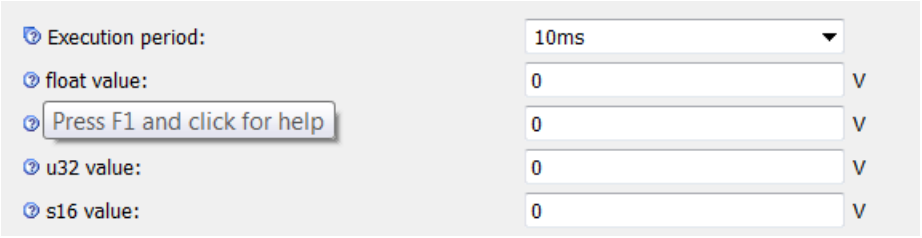
Normaalitilassa ohjejärjestelmä ei vaikuta hiiren kursorin ulkonäköön, ja ohjetilaan voidaan siirtyä koska tahansa painamalla F1-näppäintä. Kontekstisensitiivisen ohjetilan ollessa aktiivisena indikoidaan jatkuvasti hiiren kursorikuvakkeella, millä käyttöliittymäkomponenteilla on ohjemateriaalia. Ohjemateriaalin saatavuus tarkistetaan HelpID-tunnisteen perusteella ohjepaketista. HelpID-tunnisteita on esitelty tarkemmin osiossa 4.1.5. Hiirtä liikutettaessa kokoajan tarkastellaan löytyykö kursorin alla olevan käyttöliittymäkomponentin, tai sen yläkomponenttien, HelpID-tunnisteen perusteella ohjemateriaalia nykyisestä ohjepaketista. Tämä esitetään käyttäjälle vaihtamalla tavallinen hiiren kursori estoikoniksi, kun hiiren alla ei ole ohjemateriaalillista komponenttia. Kun hiiri siirretään ohjemateriaalillisen komponentin päälle, näytetään yleisesti käytetty käsi-ikoni, joka ilmaisee aktivoitavaa komponenttia. Tällöin hiirellä klikattaessa ohjemateriaalillista komponenttia, avataan kyseiseen komponenttiin liittyvä ohjesivu ohjeselaimeen. Klikattaessa ohjemateriaalitonta komponenttia, tai painamalla ESC-näppäintä, palataan takaisin normaalitilaan.

Ohjemateriaalin etsinnässä on varamekanismi, joka auttaa käyttäjää löytämään mahdollisimman lähellä aktivoitua käyttöliittymäkomponenttia olevan ohjesivun. Vaikka käyt-

töliittymäkomponentilla itsellään ei olisi ohjemateriaalia saatavilla, ei se tarkoita etteikö siihen liittyvää ohjemateriaalia olisi ollenkaan tarjolla. Komponentin alla olevat, eli sen yläkomponentit, käydään järjestyksessä läpi ja etsitään niille ohjemateriaalia. Tällöin käyttäjä pääsee mahdollisimman lähellä olevaan ohjesivuun, jos komponenttihierarkiassa edes jollain tasolla on ohjemateriaalia olemassa. Tästä johtuen kuitenkin QMainWindow-luokalle, joka on käyttöliittymän pääikkuna, ei tule asettaa HelpID-tunnistetta, ettei kaikille mahdollisille komponenteille löytyisi pääikkunan ohjesivua. Teknisesti varamekanismin tekee helpoksi Qt-kehyksen muistinhallintarakenne: jokaisella komponentilla tulisi olla yläobjekti, engl. parent object, joka on vastuussa alakomponenttiensa muistinhallinnasta. Miltä tahansa QObject-luokasta periytetyltä oliolta voi kysyä parent-funktiolla sen yläkomponenttia. Käyttöliittymässä tämä hierarkia sopii erinomaisesti ohjemateriaalin etsimiseen, koska näkyvän komponentin alla oleva komponentti on yleensä sen yläkomponentti.

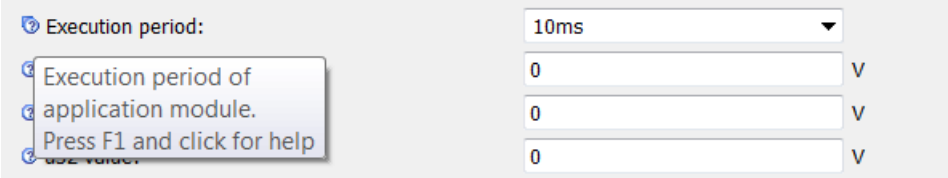
4.1.2 Käyttöliittymävihjeet

Applikaatioiden dynaamisissa parametrikäyttöliittymissä jokaisen ohjemateriaaliin viittaavan käyttöliittymäkomponentin vieressä näytetään kysymysmerkki-ikoni, jos viitattu ohjesivu löytyy ohjepaketista. Jos komponentilla on työkaluvihje, näytetään komponentin vieressä myös sitä kuvaava ikoni. Ohjejärjestelmä tarjoaa toiminnallisuuden kysymysmerkki-ikonin lisäämisen minkä tahansa kuvan tai ikonin reunaan.



Kuva 4.2. *Kysymysmerkki-ikonit indikoivat komponenttiin liittyvää ohjemateriaalia.*

Kuvassa 4.2 on float value -parametri, jolla ei ole työkaluvihjettä, mutta sillä on ohjemateriaalia. Kysymysmerkki-ikonin vasemmassa ylänurkassa ei näytetä työkaluvihjeestä ilmaisevaa sinistä kolmiota tässä tapauksessa. Tällöin komponentin työkaluvihjeenä näytetään vihje ohjeselaimen avaamiseen painamalla F1-näppäintä ja klikkaamalla joko kyseistä ikonia, parametrin nimeä tai sen arvokenttää. Vertailun vuoksi mainittakoon, että osiossa 2.2.2 on nähtävissä pelkkä työkaluvihjettä ilmaiseva sininen kolmio, ilman ohjemateriaalia ilmaisevaa kysymysmerkki-ikonia.

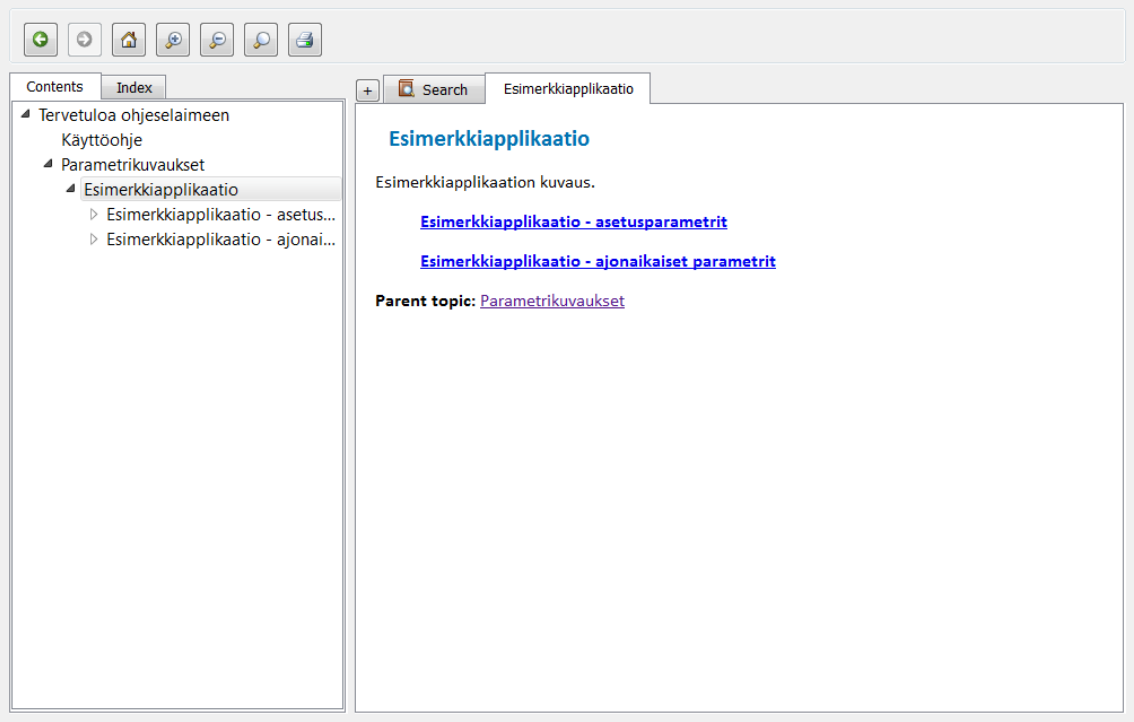


Kuva 4.3. *Parametri, jolla on sekä ohjemateriaalia että työkaluvihje.*

Kuvassa 4.3 on Execution period -parametri, jolla on sekä työkaluvihje että ohjemateriaalia saatavilla. Kuten kuvasta nähdään tällöin kysymysmerkki-ikonin vasemmassa yläkulmassa on työkaluvihjeen ilmaiseva sininen kolmio. Työkaluvihjeen loppuun lisätään myös vihje, joka jälleen ilmaisee mahdollisuutta avata ohjesivu.

4.1.3 Ohjeselain

Ohjeselain on osa ohjeliitännäistä, ja sen avulla käyttäjä voi selata ohjemateriaalia. Kontekstisensitiivinen ohjemateriaali avataan ohjeselaimessa tarkasteltavaksi, kun käyttäjä aktivoi sen F1-näppäintä painamalla ja hiirellä klikkaamalla. Ohjeselain muistuttaa suurelta osin tavallista verkkoselainta, mutta se sisältää myös tavallisesta selaimesta poiketen esimerkiksi ohjemateriaalin sisällysluettelon.

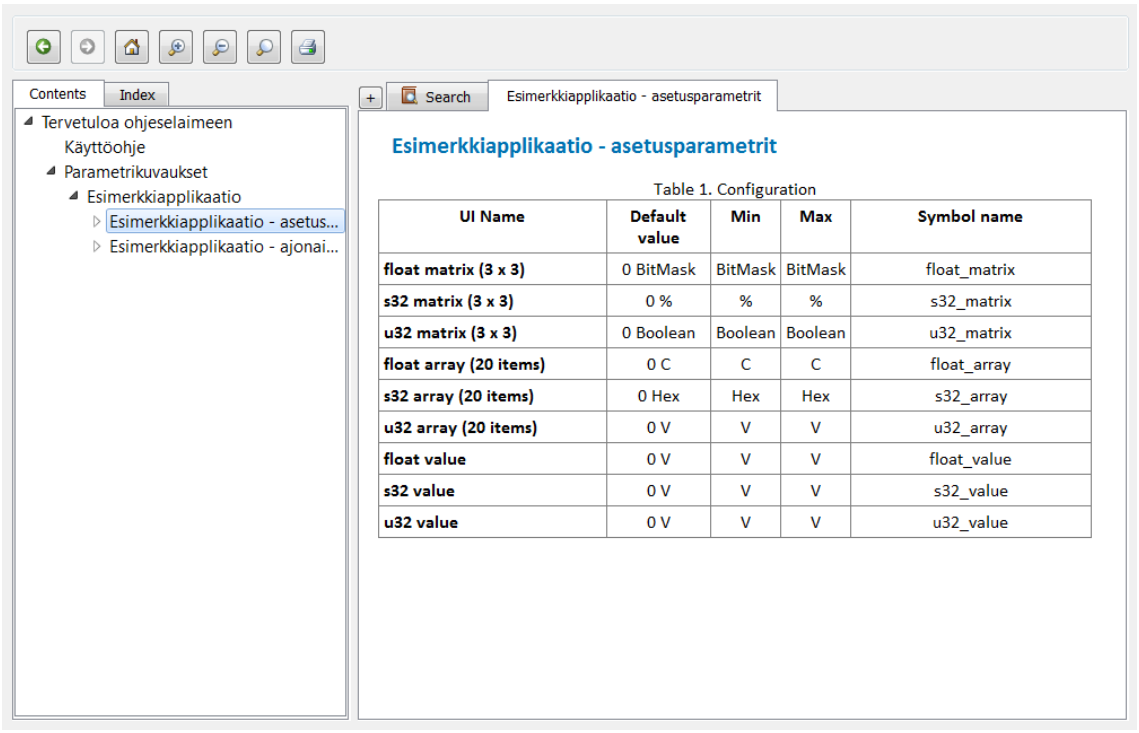


Kuva 4.4. *Ohjeselaimen perusnäkymä, jossa on Esimerkkiapplikaation ohjesivu näkyvissä.*

Kuvassa 4.4 näkyy ohjeselaimen perusnäkymä. Yläreunassa olevat toimintopainikkeet mahdollistavat internet-selaimista tutut toiminnot. Järjestyksessä painikkeet vasemmalta oikealle ovat: sivuhistoriassa taakse- ja eteenpäin siirtyminen, aloitussivulle siirtyminen, suurennus, pienennys, suurennuksen palautus oletusarvoon ja sivun tulostus. Painikkeiden alla on vasemmalla sisällysluettelo ja oikealla sisältöalue.

Sisällysluettelossa on toisena välilehtenä hakemisto, jossa on yksinkertainen lista ohjesivuista. Sisältöalueen ensimmäisenä välilehtenä on aina hakutoiminto, jolla voidaan etsiä hakusanan perusteella ohjesivuja nykyisestä ohjepaketista. Hakemisto ja hakutoiminto ovat paremmin esillä kuvassa 4.6. Plussa-painikkeesta saa lisättyä välilehtiä sisältöalueeseen, joissa jokaisessa voi olla eri ohjesivu avattuna. Uusia välilehtiä saa avattua myös klikkaamalla ohjemateriaalissa olevia linkkejä hiiren oikealla painikkeella ja valitsemalla uuteen välilehteen avaamisen.

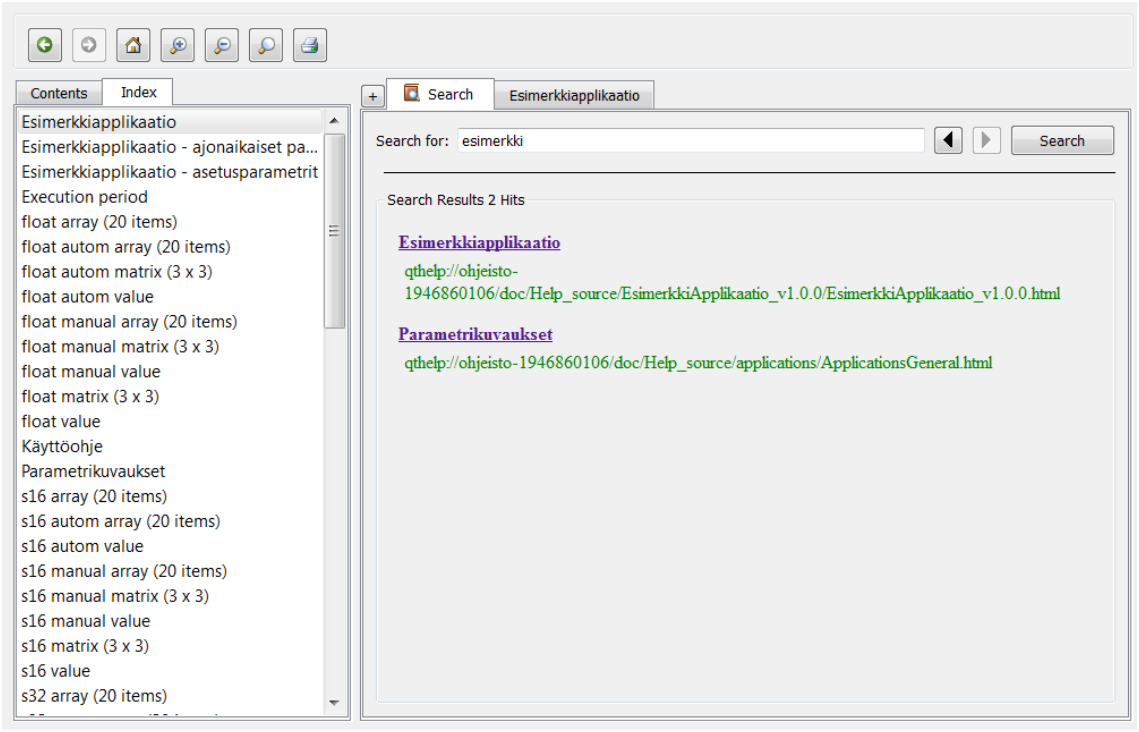
Sisällysluettelo ja hakemisto saadaan suoraan ohjepaketista, eikä toteutuksen vastuulle jää kuin tiedon esittäminen. Sisältöalueen toteutuksen pohjana on QWebView-luokka ja se tarjoaa toiminnallisuuden myös ylärivin perustoimintopainikkeille. Ohjeselaimen toteutuksesta suurin osa on eri komponenttien yhteensovittamista ja pieniä lisäyksiä niiden toiminnallisuuteen.



Kuva 4.5. *Ohjeselaimessa applikaation parametrisivu.*

Kuvassa 4.5 on Esimerkkiapplikaation parametrisivu, josta on nähtävissä kaikkien applikaation parametrien perustiedot. Kuten luvussa 2 esitettiin, käytetään applikaatioita automaatiojärjestelmän toiminnallisuuden toteuttamiseen, ja applikaatioiden parametreilla

säädetään järjestelmän toimintaa. Tämä taulukkosivu on automaattisesti generoitu DITA-runko, joka on tuotettu kohdesovelluksen applikaatiomuokkaimella. DITA-runkojen generoinnista on kerrottu tarkemmin osiossa 4.1.6. Taulukkosivun yläsivuna oleva applikaation pääsivu on samaan tapaan generoitu, ja näitä sivuja voitaisiin jatkaa sisällyttämällä lisätietoja muista DITA-aiheista niihin. Taulukkosivun sekä esimerkkiapplikaation muiden sivujen DITA-muotoiset kuvaukset on esitelty tarkemmin liitteessä A.



Kuva 4.6. *Ohjeselaimen indeksi ja hakutoiminto.*

Ohjeselaimen hakutoiminto, kuten kuvassa 4.6 on nähtävissä, näyttää haun jälkeen ensin osumien yhteismäärän ja sitten linkin jokaiseen hakuehdon sisältävään sivuun. Qt Help -ohjejärjestelmä tarjoaa valmiin mekanismin haun toteuttamiseen, eikä hakualgoritmia tarvitse itse toteuttaa.

4.1.4 Käyttäjäryhmät

Kuten osiossa 2.2.4 arveltiin, tapahtuu käyttäjätasojen erottelu jo ohjepaketteja generoitaessa. Käyttämällä DITA-merkkauskielen tarjoamaa audienssiattribuuttia voidaan materiaalia erotella tehokkaasti jo generointivaiheessa, ja jokaisen käyttäjätason pakettiin sisällytetään ainoastaan kyseiselle tasolle kuuluva materiaali. Ohjeliitännäisen vastuulle jää ainoastaan ladata oikean käyttäjätason ohjepaketti käyttöön.

Kaikkien käyttäjätasojen ohjepaketit toimitetaan kohdesovelluksen asennuspaketin muka-

na. Tämän osalta tuleekin huomioida tietoturvaan liittyvät kysymykset. Ohjepaketit ovat pohjimmiltaan SQLite-tietokantatiedostoja, ja niiden sisältöä pystyy tutkimaan sopivilla työkaluilla. Ohjepaketit eivät siis voi sisältää arkaluontoista materiaalia millekään käyttäjätasolle ilman erillistä salausta, jonka ohjejärjestelmä pystyy purkamaan ajonaikaisesti. Jatkokehitysideoihin osiossa 5.2 on lisätty ohjepakettien tehokas salaaminen tämän mahdollisen ongelman korjaamiseksi.

4.1.5 HelpID-tunnisteet

Ohjepaketissa olevat ohjesivut linkitetään määrämuotoisella HelpID-tunnisteella kohdesovelluksen käyttöliittymäkomponentteihin. HelpID-tunniste upotetaan käyttöliittymäkomponentteihin Qt-ohjelmistokehyksen ominaisuusjärjestelmän avulla, joka mahdollistaa nimettyjen ominaisuuksien asettamisen. Ominaisuusjärjestelmä on toteutettu käyttämällä luokkakääreitä, engl. wrapper, jotka laajentavat luokkien toiminnallisuutta. Näitä luokkakääreitä kutsutaan Qt-kehyksessä metaobjekteiksi. [17] Qt-metaobjektit tarjoavat olennaisimpina toimintoina objektien välisen kommunikaation signaalimekanismilla, ajonaikaista tyyppi-informaatiota ja dynaamisen ominaisuusjärjestelmän. Qt-kehyksen metaobjektijärjestelmä tarjoaa mahdollisuuden asettaa ominaisuuksia, engl. property, kaikille QObject-kantaluokasta periytyville luokille [18]. Käytännössä kaikkien käyttöliittymän luokkien periytymishierarkiassa on QObject-kantaluokka mukana, joten luokkiin ei tarvitse tehdä erikseen jäsenmuuttujaa HelpID-tunnisteen säilömiseen, vaan tunniste voidaan asettaa objektin ominaisuudeksi. Sama HelpID-tunniste upotetaan dynaamisten käyttöliittymien elementteihin XML-kuvauksessa, jolloin myös dynaamiset käyttöliittymäkomponentit saadaan linkitettyä ohjesivuihin.

HelpID-tunnisteet ovat määrämuotoisia, sijainnin määrittäviä osoitteita. Muodostetaan esimerkin vuoksi yksittäisen applikaation parametrin HelpID-tunniste. HelpID-tunnisteessa ensin määritellään kategoria, joka tässä esimerkissä on applikaatio. Kategoria viittaa yleensä näkymään, jossa viitattava komponentti sijaitsee. Käyttöliittymän runkokomponenteille, jotka näkyvät kaikissa näkymissä, on myös oma kategoriansa. Kategorian ollessa applikaatio, viittaa HelpID-tunniste applikaation sisältämästä parametrista dynaamisesti luotuun käyttöliittymäkomponenttiin. Kategorian jälkeen tunnisteessa on applikaation nimi. Sitten C-kirjain, joka kuvaa että kyseessä on applikaation asetusparametri. Lopuksi tunnisteessa on varsinainen parametrin nimi. Jos parametri on jonkin säiliön sisällä, tulee säiliön nimi vielä ennen parametrin nimeä tunnisteeseen. HelpID-tunniste rakentuu siis kokonaisuudessaan seuraavaan tapaan: **Applikaatio-EsimerkkiApplikaatio-C-säiliö-parametri**. DITA-formaatin merkkirajoituksista johtuen käytetään HelpID-tunnisteen erottimena viivamerkkiä perinteisen kauttaviivan sijaan.

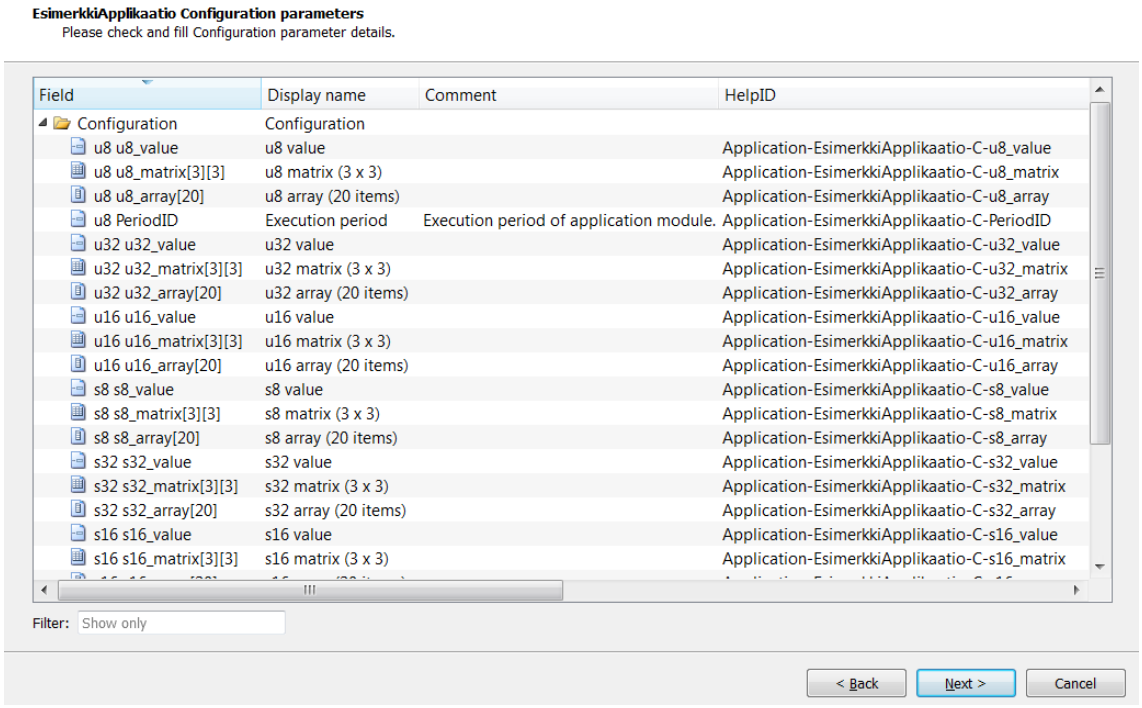
Generoituja käyttöliittymäkomponentteja lukuun ottamatta HelpID-tunnisteet täytyy asettaa kunkin käyttöliittymäkomponentin HelpID-ominaisuuteen. HelpID-tunniste voidaan myös muodostaa ajonaikaisesti kullekin komponentille, joten jatkokehityksessä tämä voitaisiin automatisoida. Tällöin enää ohjemateriaalin tulee sisältää sopiva HelpID-tunniste, joka muodostetaan kullekin viitattavalle komponentille ajonaikaisesti. Käytännössä HelpID-tunnisteet voivat olla mitä tahansa DITA-formaatin ja Qt Help -ohjejärjestelmän hyväksymiä merkkijonoja, pois lukien tyhjä merkkijono. Edellä kuvattu HelpID-tunnisteformaatti on valittu käytettäväksi ohjeprosessissa.

QHelpEngine-luokka tarjoaa tarvittavan toiminnallisuuden ohjesivujen etsimiseen HelpID-tunnisteiden perusteella. HelpID voidaan siis QHelpEngine-luokan avulla muuntaa qthelp-protokollan mukaiseksi URL-osoitteeksi, jonka ohjeselain pystyy avaamaan.

4.1.6 Ohjemateriaalin generointi

Kohdesovellukseen lisätään toiminnallisuus, jolla saa kehitysaikana generoitua applikaatioille DITA-muotoisen ohjemateriaalirungon. Runko sisältää tiettyyn applikaatioon ja sen parametreihin liittyvät perustiedot, kuten kuvauksen, arvoalueen, asetetun arvon, yksikön sekä generoidun HelpID-tunnisteen. Toiminnallisuus sijoittuu luvussa 2 esitettyyn kukankastelujärjestelmään järjestelmäpaketin konfiguraation kehityksen yhteyteen. Kun konfigurointisovelluksella tuotetaan ohjausapplikaation perustiedoista ja parametreista XML-konfiguraatio, samalla generoidaan DITA-rungot ohjemateriaalin pohjaksi. Nämä generoidut ohjerungot voidaan ottaa mukaan ohjepaketin prosessointiin, jolloin ne päätyvät mukaan valmiiseen ohjepakettiin. Tällöin kukankastelujärjestelmää käytettäessä voidaan ohjeselaimen avulla lukea tietoa parametreista ja niiden säätämisestä. Ohjemateriaali kirjoitetaan siis samanaikaisesti kun ohjausapplikaatiota kehitetään.

Kohdesovelluksen applikaatiomuokkaimella saadaan määriteltyä applikaation parametrit ja generoitua applikaation XML-kuvaus. Generoidut XML-kuvaukset voidaan sisällyttää konfiguroitaviin järjestelmiin. Tämän avulla eri järjestelmien toiminnallisuutta saadaan modularisoitua ja eri osia voidaan uudelleenkäyttää. DITA-materiaalin generointi lisätään osaksi applikaatiomuokkainta. Tällöin HelpID-tunnisteet lisätään applikaation XML-kuvaukseen ja XML-kuvauksien luomisen yhteydessä generoidaan DITA-rungot ohjemateriaalia varten.



Kuva 4.7. *Applikaatiomuokkain HelpID-tunnisteiden ja DITA-runkojen generointiin.*

Kuvassa 4.7 on esillä kohdesovelluksen applikaatiomuokkain. Näkyvissä on EsimerkkiApplikaatio-applikaation asetusparametrit, engl. configuration parameters, joiden avulla voidaan säätää applikaation toimintaa. Nämä voisivat esimerkiksi olla prosessoinnissa tarvittavia matemaattisia arvoja tai mitä tahansa muita säätöarvoja, joita ohjausapplikaatio käyttää. Field-kentässä on parametrin tekninen nimi ja Display name -kentässä käyttöliittymässä käyttäjälle näytettävä parametrin selkokielinen nimi. Comment-kenttä näytetään parametrin työkaluohjeena käyttöliittymissä. Oikeanpuoleisimmassa sarakkeessa näkyy kullekin parametrille generoitu HelpID-tunniste. HelpID-tunnisteita voidaan muokata saman käyttöliittymän kautta, mutta tämä ei yleensä ole tarpeellista.

Kun applikaation ja sen parametrien määrittely on valmis, voidaan sen XML-kuvaus generoida. Samalla tuotetaan DITA-rungot, jotka sisältävät applikaation ja sen parametrien perustiedot. DITA-rungot sekä XML-kuvaus sisältävät applikaatiolle ja sen parametreille generoidut HelpID-tunnisteet, joiden avulla niihin liittyviin ohjesivuihin voidaan kohdesovelluksesta viitata. Tällöin, kun applikaatio otetaan käyttöön ja siihen liittyvä käyttöliittymä generoidaan applikaation XML-kuvauksen perusteella, saadaan käyttöliittymäkomponentit liitettyä oikeaan ohjesivuun HelpID-tunnisteiden avulla.

Applikaatiosta luodaan seuraavat DITA-muotoiset tiedostot:

- EsimerkkiApplikaatio_v1.0.0.ditamap, joka sisältää hierarkkisen kuvauksen applikaation ohjetiedostoista.

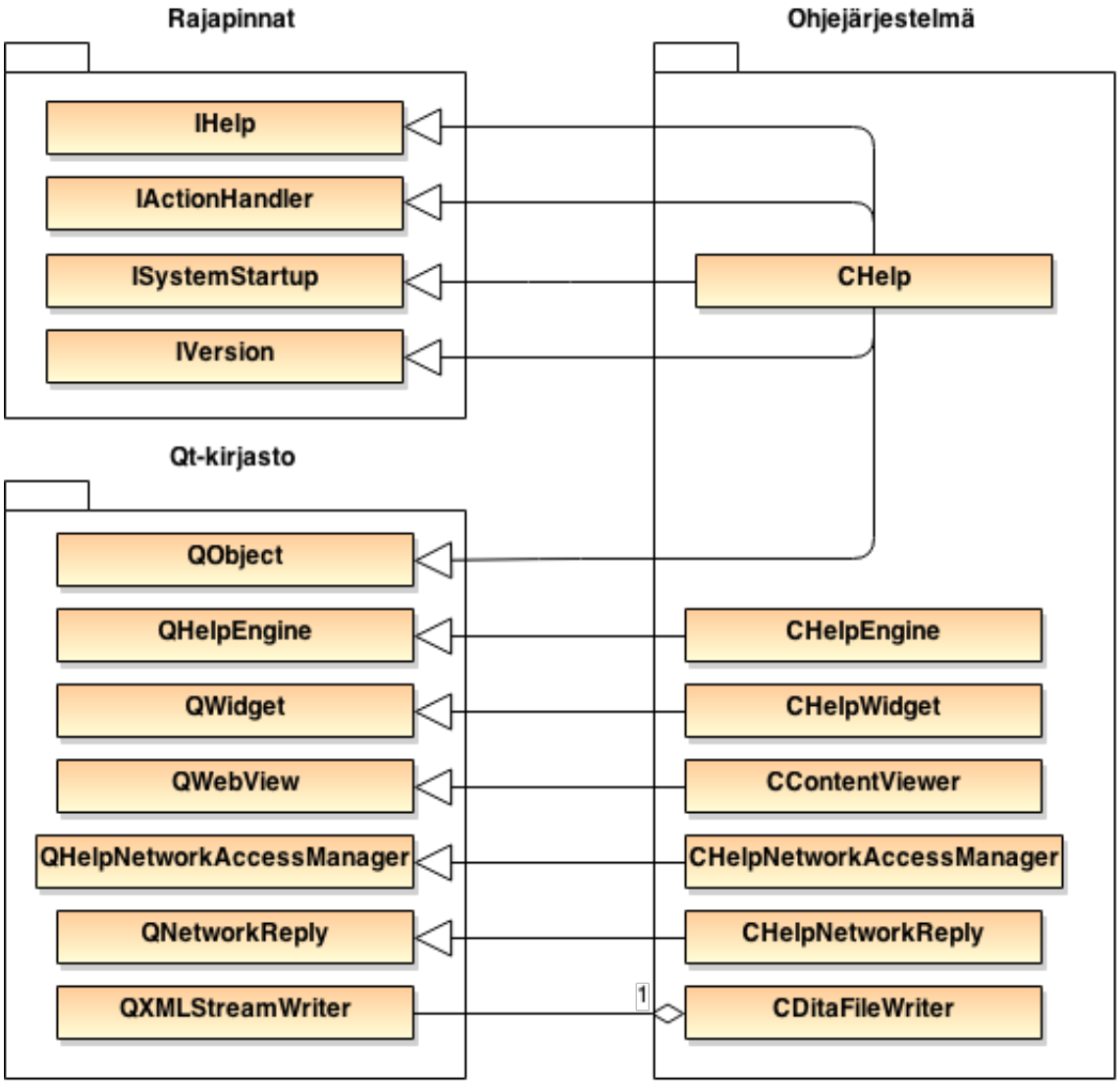
- EsimerkkiApplikaatio_v1.0.0.dita, jossa on yleiskuvaus applikaatiosta.
- EsimerkkiApplikaatio_config_table_v1.0.0.dita, jossa on taulukko applikaation parametreista.
- parameters-kansio, jossa on vielä jokaiselle yksittäiselle parametrille oma ohjesivu.

Jokaisessa tiedoston nimessä on mukana applikaation versionumero, koska applikaatiomuokkaimella voidaan tehdä uusia versioita applikaatioista, ja jokaisella versiolla parametrit voivat olla erilaiset. Tästä syystä ohjepakettiin on otettava mukaan oikean applikaatioversion ohjemateriaalit. Liitteessä A on esitelty generoituja applikaation DITA-runkoja. Liitteessä on nähtävissä millainen DITA-merkkauskieli käytännössä on ja millaiset rakenteet edellä esitellyillä tiedostoilla on.

4.2 Suunniteltu arkkitehtuuri

Edellä kuvattiin järjestelmän osat yleisellä tasolla, vaatimuksia silmällä pitäen, ja seuraavaksi käydään läpi tarkempi tekninen suunnittelu. Ohjeprojektin työkieli on englanti, ja siitä johtuen esiteltyjen luokkien nimet ja ohjelmakoodi ovat englanninkielisiä. Nimeämiskäytäntönä on liittää luokkien nimen eteen C-kirjain ja rajapintojen eteen I-kirjain. Ohjejärjestelmän luokat ovat Plugins::GUI::Help -nimiavaruudessa. Qt-kehys toteuttaa oliosuunnittelumallia ja myöskin ohjejärjestelmä suunnitellaan olio-ohjelmoinnin menetelmiä hyödyntäen.

Qt-kehyksessä suuressa roolissa on MVC-suunnittelumalli ja siihen tullaan myös jatkossa viittaamaan. MVC-suunnittelumallin mukaan toiminnallisuus jaetaan malliin, näkymään ja ohjaimeen. Malli on vastuussa tiedon säilömisestä, näkymä tiedon esittämisestä ja ohjain käyttäjän syötteen vastaanottamisesta ja mallin päivittämisestä. [6] Qt-kehys toteuttaa MVC-mallin muuten perinteiseen tapaan, mutta ohjain on korvattu delegaatilla. Delegaatti ottaa myös käyttäjän syötettä vastaan, mutta se määrittelee myös syöttö- ja esitystavan. Delegaatti ei ole välttämättä oma osansa vaan se on yleensä osa näkymää, ja delegaatista johtuen Qt-kehyksessä toteutettua MVC-mallia kutsutaan MVD-malliksi. [16]



Kuva 4.8. *Ohjejärjestelmän arkkitehtuurikaavio.*

Ohjejärjestelmä toteutetaan käytännössä dynaamisesti linkitettynä kirjastona, joka ladataan ajonaikaisesti kohdesovelluksen käyttöön. Ohjejärjestelmän arkkitehtuurikaavio on nähtävissä kuvassa 4.8. Kuvassa on nähtävissä kolme eri pakettia: rajapinnat, Qt-kirjasto ja ohjejärjestelmä. Rajapintapaketissa on ohjejärjestelmän käyttämät rajapinnat ja sen tarjoama IHelp-rajapinta. Qt-kirjastopaketissa on Qt-ohjelmistokehyksen tarjoamat luokat, joista ohjejärjestelmän luokat on periytetty. Ohjejärjestelmäpaketti sisältää ohjejärjestelmän toiminnallisuuden. **IHelp**-rajapinta on ohjeliitännäisen muulle sovellukselle tarjoama julkinen rajapinta. Rajapinnat **IActionHandler**, **ISystemHandler** ja **IVersion** tarjoavat liitännäisarkkitehtuuriin liittyviä yleisiä toimintoja. **CHelp**-luokka on liitännäisen pääluokka, ja se tarjoaa IHelp-rajapinnan muun sovelluksen käyttöön. **CHelpEngine** on QHelpEngine-luokasta peritty toteutus, jolla ohjepakettia luetaan. **CHelpWidget** on ohjeselain, ja sisältää sen toiminnallisuuden ja käyttöliittymäkuvauksen. **CContentViewer** luokan vastuulla on näyttää ohjemateriaalia, ja se on osa CHelpWidget-luokan ohjese-

lainta. **CHelpNetworkAccessManager** ja **CHelpNetworkReply** ovat apuluokkia, joilla toteutetaan paikallinen tiedonsiirto ohjeselaimessa. **CDitaFileWriter** tuottaa DITA-muotoisia runkoja ohjesivujen pohjiksi, käyttäen sisäisesti QXMLStreamWriter-luokkaa. CDitaFileWriter-luokka ei ole osa ohjeliitännäistä, vaan se on osa kohdesovelluksen applikaatiomuokkainta.

4.3 Luokkakuvaukset

Jokaisella ohjejärjestelmän luokalla on oma vastuualueensa ja toiminnallisuus, jonka ne toteuttavat. Seuraavissa osioissa on esitelty ohjejärjestelmän luokat ja niiden tekninen toteutus.

4.3.1 IHelp-rajapinta

IHelp-rajapinta on ohjeliitännäisen julkinen rajapinta muun sovelluksen käyttöön. Tämän rajapinnan tarjoamat funktiot ovat ainoita, jotka näkyvät liitännäisen ulkopuolelle. Rajapinta tarjoaa neljä päätoiminnallisuutta: liitännäisen alustustoimenpiteet, HelpID-tunnisteen olemassaolon tarkistus, työkaluohjetekstiin vihjetekstin lisääminen sekä ikonivihjeiden lisääminen kuviin. Kun liitännäisen IHelp-rajapinnan toteuttavasta CHelp-luokasta instantioidaan objekti, se luo samalla käyttöönsä instanssin CHelpEngine-luokasta. CHelp ei itsessään ole Singleton-suunnittelumallin[1] mukainen luokka, mutta sitä käyttävä kohdesovelluksen liitännäistenhallinta käyttää sitä samaan tapaan. Liitännäistenhallinta luo jokaisesta liitännäisestä tarvittaessa yhden instanssin, ja sen jälkeen jakaa osoitinta kyseiseen instanssiin.

HelpID-tunnisteen olemassaolon tarkistus mahdollistaa ohjepaketin sisällön tutkimisen. Toiminto kertoo onko kyseisen HelpID-tunnisteen mukainen ohjesivu olemassa käytössä olevassa ohjepaketissa. Tämän perusteella voidaan esimerkiksi päättää näytetäänkö käyttöliittymässä ohjemateriaalin saatavilla oloa ilmaisevia vihjeitä, kun applikaatioiden dynaamisia näkymiä rakennetaan. Tapahtumasuodatin käyttää myös kyseistä toimintoa, kun käyttäjä aktivoi käyttöliittymäkomponentin ohjeen. Työkaluvihjetekstien ja vihjekuvien lisäämistä käytetään käyttöliittymien rakentamisessa. Näiden avulla saadaan luotua osiossa 2.2.2 esitellyt käyttöliittymävihjeet ohjemateriaalin saatavuudesta. Huomionarvoista on, että rajapinnassa ei ole funktiota, jolla voi avata ohjeselaimen tietyn HelpID-tunnisteen mukaiselle sivulle. Tämä johtuu siitä, että ohjeselain avataan ainoas-

[1] Singleton-suunnittelumallin mukaisesti luokasta on vain yksi instanssi olemassa, johon päästään käsiksi globaalisti. Tällöin resurssin jakaminen eri sovelluksen osiin on helppoa, ja voidaan varmistua ettei sitä tarpeettomasti monisteta. [6]

taan näppäin- ja hiiritapahtuman perusteella tapahtumasuodattimessa tai valitsemalla ohjeselain sovelluksen päävalikosta. Päävalikosta valitsemisen tapauksessa pyyntö ohjeselaimen avaamiseksi tulee IActionHandler-rajapinnan kautta. Tarvittaessa IActionHandler-rajapintaa voidaan käyttää myös muualta sovelluksesta ohjeselaimen avaamiseen, mutta tämä ei nykyisellään ole merkittävä käyttötapaus.

IHelp-rajapinnassa määritellään HelpID-ominaisuus, johon HelpID-tunniste säilötään. Aiemmin esitelty Qt-metaobjektimalli mahdollistaa nimettyjen ominaisuuksien asettamisen objekteihin, tässä tapauksessa HelpID-ominaisuuteen säilötään HelpID-tunniste merkkijonona. Ominaisuudet ovat yksinkertaisia nimi-arvo-pareja, joista nimi on merkkijono ja arvo voi QVariant-luokkana sisältää kaikkia QVariant-luokan tukemia tietotyyppejä. QVariant itsessään on yksinkertaisesti muuttuvatyyppinen tietosäiliö. Rajapinnassa määritellään myös HelpID-tunnisteille oma datarooli, engl. data role, jolla HelpID-tunniste voidaan upottaa MVD-mallin tietorakenteisiin. Qt-kehyksen MVD-malli voi sisältää dataa eri esitysmuotoja varten, ja näitä kutsutaan datarooleiksi. Esimerkiksi QTableView-taulukon solun käyttäjälle esitettävä data on talletettu DisplayRole-rooliin ja näkymä osaa automaattisesti käyttää sitä taulukon piirrossa. Vastaavana esimerkkinä ToolTipRole-roolia käytetään solun työkaluvihjeen säilömiseen. Ohjemateriaalia etsittäessä on HelpID-tunniste helpompi noutaa mallista dataroolin perusteella kyseiselle solulle, kuin objektin ominaisuudesta. Informaatioarkkitehtuurin kannalta ei kuitenkaan ole samantekevää voidaanko taulukon yksittäinen solu kohdentaa ohjesivuun vai onko taulukolle aina yksi ohjesivu, jossa on myös yksittäisiin soluihin liittyvää tietoa. Esimerkiksi kohdesovelluksessa on useimmiten luontevampaa olla vain yksi ohjesivu taulukkoa kohti, koska taulukot ovat yleensä toiminnallisesti yksi automaatiojärjestelmän parametri.

4.3.2 CHelp-luokka

CHelp on liitännäisen pääluokka, joka toteuttaa ohjejärjestelmän IHelp-rajapinnan. Se toteuttaa myös kolme muuta rajapintaa, jotka ovat: IActionHandler, ISystemStartup ja IVersion. IHelp-rajapinnan toiminnallisuus on kuvattu tarkemmin edeltävässä osiossa. IActionHandler-rajapinta toteuttaa käsittelyn valikon kautta ajettaville toiminnoille, jotka on toteutettu QAction-luokkina. Tämän rajapinnan avulla avataan ohjeselain, kun käyttäjä valitsee sen sovelluksen Help-valikosta. QAction-luokkia käytetään yleisestikin valikkovaihtoehtojen kuvaamiseen Qt-kehyksessä. Niitä voidaan käyttää sekä sovelluksen päävalikoissa että hiiren oikeaa painiketta painettaessa avattavissa kontekstivalikoissa. Käytännössä ne yhdistävät valikkovaihtoehtoja niitä suorittaviin funktioihin. ISystemStartup-rajapinnan kautta kutsutaan liitännäisiä suorittamaan tarvittavat toiminnot sovelluksen käynnistyessä. ISystemStartup-rajapintaa ei toistaiseksi tarvita, koska ohjepakettia ei ladata sovelluksen käyttöön käynnistyessä. Ohjepaketti ladataan vasta, kun käyttäjä pyy-

tää ohjemateriaalia tarkasteltavakseen. IVersion-rajapinnan kautta liitännäiseltä voidaan kysyä sen versiotietoja, ja se onkin kyseisen rajapinnan ainoa käyttötarkoitus. Nämä 3 viimeksi mainittua rajapintaa tulevat vaatimuksena suoraan kohdesovelluksen liitännäisarkkitehtuurista, eivätkä ne vaikuta sen enempää ohjejärjestelmän suunnitteluun.

IHelp-rajapinnan esittelemän toiminnallisuuden lisäksi CHelp-luokka toteuttaa tapahtumasuodattimen, jolla vastaanotetaan näppäin- ja hiiritapahtumia. Erityisesti ohjeliitännäisessä halutaan vastaanottaa F1-näppäimen painallus, jolla aktivoidaan kontekstisensitiivinen ohjetila, ja sitä seuraavasta hiiren klikkauksesta, jolla avataan ohjeselain oikealle ohjesivulle. Käytännössä tapahtumasuodatin toteutetaan uudelleenmäärittelemällä QObject-luokan eventFilter-funktio. Qt-kehys tarjoaa tapahtumamekanismin QEvent-rajapinnan avulla, joka välittää tapahtumat ja niiden tiedot niille objekteille, jotka ovat rekisteröityneet niitä vastaanottamaan. Normaalisti tapahtumat toimitetaan virtuaalifunktioiden avulla, eli Qt-kehyksen tapahtumanvälittäjä kutsuu olion tapahtumankäsittelijäfunktioita. Tapahtumasuodattimella voidaan kuitenkin käsitellä kaikki objektille lähetetyt tapahtumat. Tapahtumasuodattimet rekisteröidään installEventFilter-funktion avulla, jolloin tapahtumat toimitetaan objektin eventFilter-funktiolle. [16] Ohjejärjestelmän kontekstisensitiivisen ohjeen aktivointiin käyttämä tapahtumasuodatin on tilakone ja toimii osiossa 2.2.1 esitetyllä tavalla.

Tapahtumasuodattimessa täytyy huomioida hiiren osoittimen koordinaattimuunnos sovelluksen globaaleista koordinaateista widgetin paikalliseen koordinaatistoon, jos hiiren alla on MVD-mallin mukainen näkymä. MVD-mallin mukaisen komponentin, esimerkiksi QTableView-taulukon, tapauksessa QCursor-luokan palauttamat koordinaatit ovat globaalissa koordinaatistossa. Ne täytyy tällöin muuntaa paikallisiksi koordinaateiksi taulukon näkymäikkunan, engl. viewport, muuntotoiminnon avulla. Paikallisten koordinaattien avulla saadaan taulukosta yksittäinen solu ja sen HelpID-tunnistetta voidaan tarkastella. Jos hiiren osoittimen alla on tavallinen komponentti, eikä MVD-mallin mukainen näkymä, niin HelpID-tunniste saadaan yksinkertaisesti pyytämällä HelpID-ominaisuuden arvo property-funktiolla. Ohjemateriaalin etsimisen varamekanismi saadaan helposti toteutettua pyytämällä widgetiltä sen yläkomponentti parent-funktiolla, ja näin voidaan etsintää jatkaa hierarkiassa ylöspäin kunnes yläkomponenttia ei ole tai ohjemateriaalia löytyy.

Työkaluvihjetekstin lisääminen on hyvin yksinkertainen toimenpide. Ensin tarkistetaan että HelpID-tunniste löytyy ja sen perusteella löytyy ohjemateriaalia. Jos molemmat ehdot täyttyvät, lisätään vakiomerkkijono olemassa olevan työkaluvihjeen perään. Kohdesovelluksessa on liitännäinen kuvien ja käyttöliittymän ikonien hallinnointiin, jolta haetaan käyttöliittymässä esitettävät kuvat. Ikonivihjeiden lisääminen kuviin onkin luontevampaa toteuttaa kyseisessä kuvienhallintaliitännäisessä. Ohjeliitännäinen tarkistaa lisätäänkö kuvaan ikoneita, ja sitten toimittaa niistä listan kuvien hallinnointiliitännäiselle, joka suorittaa ikonien lisäämisen kuvan nurkkiin. Käytännössä QPainter-luokan drawPixmap-

funktiota käytetään ikonin lisäämiseen QPixmap-objektiin ladattuun kuvaan.

4.3.3 CHelpEngine-luokka

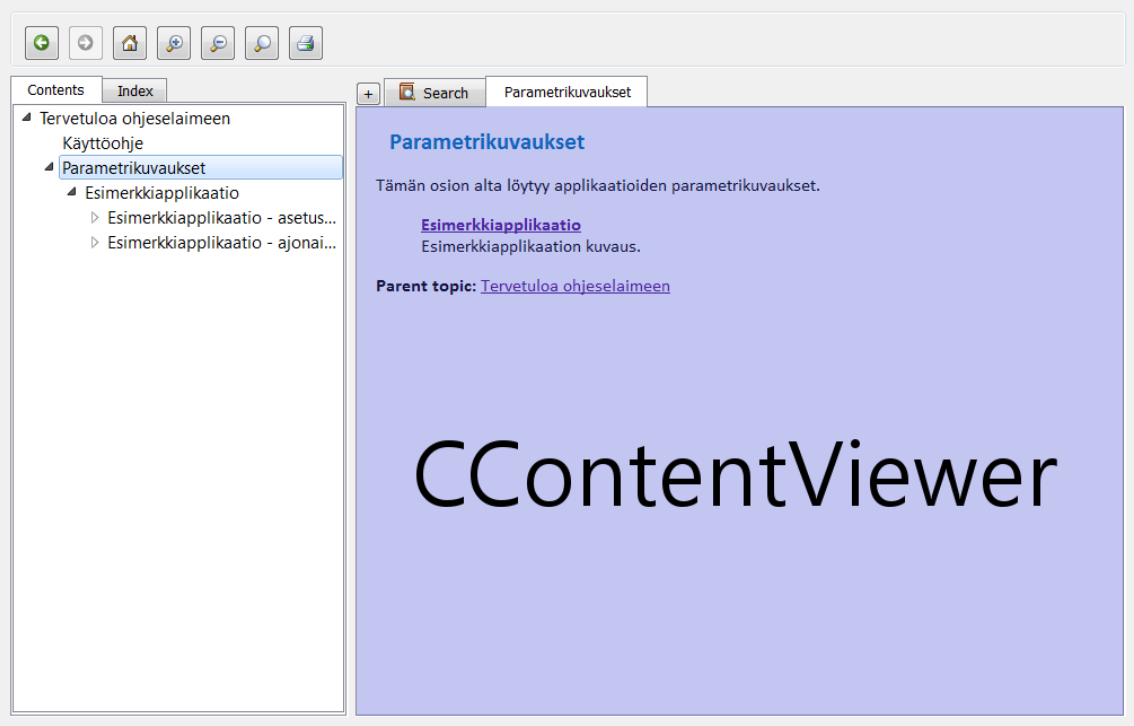
CHelpEngine-luokka on QHelpEngine-luokasta periytyvä Singleton-suunnittelumallin mukainen luokka, joka lukee ohjepakettia. Ohjepaketti itsessään on tiedostojärjestelmässä oleva tietokantatiedosto, ja QHelpEngine-luokka toimii rajapintana sen lukemiseen. Tästä syystä ohjepaketin rakenteesta tai sen sisäisestä toteutuksesta ei tarvitse tietää mitään. Oleellisinta on tietää polku, jossa ohjepaketti sijaitsee. CHelpEngine-luokan vastuulla on ohjemateriaalin etsiminen sekä noutaminen ohjejärjestelmän käyttöön. Tätä varten se hyödyntää qthelp-protokollaa, jolla paikallisessa ohjepaketissa olevia ohjesivuja voidaan noutaa ohjeselaimella näytettäviksi. Ohjesivujen sijainnit muistuttavat internetissä käytettyjä URL-osoitteita. Kuvassa 4.6 on nähtävissä polkuja, jotka ovat hakutulosten sijainteja.

Singleton-suunnittelumallin mukaisesti luokka tarjoaa funktion, jolla siitä luodaan instanssi tai palautetaan jo olemassa oleva instanssi. Luokan rakentaja, kopionrakentaja ja sijoitusoperaattori ovat yksityisiä jäsenfunktioita, eli niiden käyttö on estetty. Tämän avulla varmistetaan, että suunnittelumalli ei vuoda ja sitä käytetään oikein. Luokka tarjoaa liitännäisen käyttöön toiminnot ohjepaketin lataamiseen, HelpID-tunnisteen olemassa olon tarkistukseen sekä ohjemateriaalin noutamiseen. CHelp-luokka käyttää hyödykseen näitä toimintoja tarjotakseen IHelp-rajapinnassa esitellyn toiminnallisuuden. Ohjepaketti ladataan sovelluksen käyttöön kutsumalla setCollectionFile- ja setupData-funktioita. Ensimmäinen asettaa sisäisesti ohjekokoelmatiedoston sijainnin. Jälkimmäinen funktio lukee ohjekokoelmatiedoston ja alustaa ohjejärjestelmän sisäisen tilan. [16] Kutsumalla setupData-funktiota saadaan myös ohjemateriaalin sisäinen hakemisto rakennettua välittömästi ohjeselaimessa näytettäväksi. CHelpEngine-luokkaan kytketään signaali kohdesovelluksen ytimestä, joka emittoidaan käyttäjän kirjautuessa ulos sovelluksesta. Tällöin ohjepaketti poistetaan käytöstä ja ohjejärjestelmän tila alustetaan uudelleen.

CHelpEngine-luokka perii QHelpEngine-luokalta toiminnallisuuden, jolla saadaan noudettua ohjemateriaalia ohjepaketista. Käyttöliittymäkomponenttiin säilötty HelpID-tunniste muutetaan ensin qthelp-osoitteeksi, jonka perusteella ohjemateriaali noudetaan. Noudettu ohjemateriaali on binäärimuodossa, ja esittämistä varten se tarvitsee ensin muuttaa sopivaan muotoon. Noudetun datan tyyppi tunnistetaan siihen liitetyn MIME-tyypin perusteella. Esimerkiksi binäärimuotoinen teksti täytyy käsitellä tekstinä ja kuvat käsitellään niiden oman formaatin mukaisesti. Sovelluskehittäjän ei tarvitse näistä binäärimuodoista juuri välittää, vaan QWebView-luokka osaa käsitellä ne, kunhan MIME-tyypit tunnetaan. MIME-tyypeistä on kerrottu tarkemmin osiossa 4.3.7.

4.3.4 CContentViewer-luokka

CContentViewer-luokka on käyttöliittymän kannalta katsottaessa sisältöalue ohjeselaimessa, joka kokonaisuudessaan on CHelpWidget-luokan toteuttama. Kuva 4.9 selventää tilannetta. Kuvassa nähdään minkä osan käyttöliittymästä CContentViewer-luokka toteuttaa, ja loppu on CHelpWidget-luokan vastuulla.



Kuva 4.9. *CContentViewer-luokan osuus ohjeselaimen käyttöliittymästä korostettuna.*

CContentViewer-luokka periytyy QWebView-luokasta, joka on toteutettu QtWebKit-paketissa. QtWebKit on Qt-ohjelmistokehyksen mukana toimitettava kirjasto, joka täytyy erikseen ottaa mukaan käännökseen samaan tapaan kuin Qt Help -ohjejärjestelmä. QtWebKit-kirjasto toteuttaa WebKit-verkkosivumoottorin, joka renderöi HTML-sivuja ja suorittaa JavaScript-ohjelmakoodia. [16] QtWebKit-kirjasto tarjoaa siis käytännössä sovellukseen upotettavan verkkoselaimen. CContentViewer-luokan vastuulla on ohjemateriaalin hakeminen ohjepaketista käyttäjälle esitettäväksi CHelpEngine-luokan avustuksella. CHelpEngine-luokka suorittaa ohjemateriaalin noutamisen ohjepaketista ja CContentViewer-luokan pääasiallisena vastuuna on sen esittäminen.

QWebView-luokka on näkymä, joka esittää verkkosivuja. Luokan nimen View-osa viittaa Qt-ohjelmistokehyksen MVD-suunnittelumalliin. Verkkosivun sisältö toimii luokan mallin datana, ja QWebView-luokka hoitaa näkymän vastuun, eli sivun esittämisen käyttäjälle. Tarkemmin sanottuna QWebView-luokka sisältää QWebPage-luokan, joka esittää

varsinaista näkymässä olevaa verkkosivua. Näihin luokkiin liittyy oleellisesti käsite linkkien delegointipolitiikka, engl. link delegation policy. Delegointipolitiikka tarkoittaa toimintatapaa jota noudatetaan, kun verkkosivulla aktivoidaan linkki. Delegointipolitiikka määrittää, mihin kohteeseen käsiteltäväksi sivulla aktivoitu linkki annetaan. Oletuksena sivun linkit yritetään käsitellä suoraan QWebPage-luokan toimesta, joka ohjejärjestelmän linkkien tapauksessa johtaa siihen, ettei linkkiä pystytä avaamaan. [16] Edelleen tästä seuraa, että QWebPage-olio delegoi linkin käyttöjärjestelmän oletusselaimen käsiteltäväksi, joka ei myöskään pysty ohjepaketin polkua avaamaan. Ohjeselaimen sivuille tulee asettaa QWebPage::DelegateAllLinks -delegointipolitiikka, jonka mukaisesti linkin aktivointi emittoi linkClicked-signaalin. Tämä signaali voidaan käsitellä ohjeliitännäisen sisällä, jolloin linkin osoittama ohjesivu pystytään avaamaan.

Edellä esitellyn linkkien delegointipolitiikan lisäksi CContentViewer-luokan rakentajassa asetetaan hallintaluokka verkkopyynnöille, engl. network access manager. Tämä tehdään kutsumalla QWebPage-luokan setNetworkAccessManager-funktiota, jolle annetaan CHelpNetworkAccessManager-luokan instanssi parametriksi. Kun ohjeliitännäisen sisäinen luokka on asetettu hallinnoimaan verkkopyyntöjä, saadaan ohjesivupyynnöt käsiteltyä liitännäisen sisäisesti. Käytännössä nämä pyynnöt täytetään hakemalla ohjesivu ohjepaketista CHelpEngine-luokan avulla.

Koska CContentViewer-luokka on vastuussa sisällön näyttämisestä, se toteuttaa myös ohjesivun tekstin fontin suurentamisen ja pienentämisen. Käytännössä suurennus ja pienennys toteutetaan käyttämällä QWebView-luokan setZoomFactor-funktiota, jolle annetaan skaalauskerroin. Linkin aktivoinnista emittoitava linkClicked-signaali on yhdistetty CHelpWidget-luokkaan, joka linkin perusteella pyytää CContentViewer-luokan instanssia vaihtamaan näyttämäänsä sivua. Näin toimimalla CHelpWidget-luokka voi lisätä omaa toiminnallisuutta sivun vaihtamisen yhteyteen, esimerkiksi ohjeselain voi valita mihin välilehteen linkki avataan. Tämä on hakutoiminnon yhteydessä haluttua toiminnallisuutta, koska ensimmäisen välilehden halutaan aina näyttävän hakutoiminto. Hakutuloksista aktivoitu linkki avataan oletuksena viimeksi aktiivisena olleeseen välilehteen. Vaihtoehtoisesti käyttäjä voi myös valita kontekstivalikosta linkin avaamisen uuteen välilehteen.

4.3.5 CHelpWidget-luokka

CHelpWidget-luokka toteuttaa ohjeselaimen käyttöliittymän ja toimintalogiikan. Se perii käyttöliittymäikkunan ominaisuutensa QWidget-luokalta, joka on käyttöliittymäkomponenttien kantaluokka. Ohjeselaimelle ei aseteta yläkomponenttia, jotta siitä ei tule modaalia ikkunaa. Modaali-ikkuna, tai modaalinen dialogi, estää sen alla olevan sovelluksen käyttämisen. Se vaatii, että ikkuna suljetaan ennen kuin muuta sovellusta voi jälleen käyt-

tää. Tämä on haluttua toiminnallisuutta esimerkiksi virheilmoitusten ja muunlaisten käyttäjän välitöntä huomiota vaativien viestien kanssa. Ohjeselaimen halutaan kuitenkin olevan vapaasti selattavissa samalla, kun kohdesovellusta käytetään, ja sen voi siirtää vaikka toiselle näytölle näkyviin. CHelpWidget-luokka sisältää kuvassa 4.9 esitetyn käyttöliittymän toiminnallisuuden, pois lukien CContentViewer-luokan toteuttaman sisältöalueen. Ohjemateriaalissa olevia linkkejä klikattaessa, CHelpWidget-luokka ottaa aktivointisignaalin vastaan, ja pyytää CContentViewer-luokan aktiivisena olevaa instanssia vaihtamaan näytettävää ohjesivua.

CHelpWidget-luokan rakentajassa määritetään QWebSecurityOrigin-luokan addLocalScheme-luokkafunktion avulla qthelp-protokolla paikalliseksi tiedonsiirtoprotokollaksi. Tämän avulla ohjejärjestelmä käsittelee ohjemateriaalin sijainteja paikallisina sijainteina, ja voi siten myös verkkoturvapolitiikan mukaan lukea muita paikallisia tiedostoja. Oletusarvoisesti kaikki ohjemateriaali on ohjepaketin sisällä, mutta tarvittaessa sisältöä voidaan linkittää myös muihin paikallisiin dokumentteihin. Verkkoselaimissa yleisestikin käytetään resurssin alkuperää rajaamaan osoiteavaruutta, josta resursseja voidaan ladata lisää verkkosivulle. Tällä pyritään ehkäisemään hyökkäyksiä, joissa hyökkääjä ujuttaa omaa ohjelmakoodiaan suoritettavaksi muuten viattomalle verkkosivulle, ja sitä kautta käyttäjän selaimen suoritettavaksi.

Näkymän alustuksessa CHelpEngine-luokasta luodaan instanssi ja ohjemateriaali alustetaan sen setupData-funktiolla. Käyttöliittymän runko on määritelty Qt Designer -sovelluksella tehdyssä .ui-tiedostossa. Qt Designer -sovellus tarjoaa graafisen työkalun Qt-sovelluksen käyttöliittymien suunnitteluun. Sisällysluettelo ja ohjesivuhakemisto saadaan suoraan CHelpEngine-luokalta indexWidget- ja contentWidget-funktioiden palauttamina osoittimina. Ne ovat valmiita käyttöliittymäkomponentteja, jotka ei tarvitse kuin upottaa omaan paikkaansa käyttöliittymässä. Alustuksessa kytketään myös kaikkien painikkeiden clicked-signaalit niitä vastaaviin käsittelijöihin. Alustuksen lopuksi luodaan uusi sisältövälilehti, johon avataan ohjemateriaalin etusivu. CHelpWidget-luokka pitää sisäisesti kirjaa nykyisestä avoimesta välilehdestä, viimeisimmistä hakukyselyistä, nykyisen välilehden CContentViewer-sisältönäkymästä ja hakutulosnäkymästä. Näiden tietojen avulla saadaan hallittua käyttöliittymän tilaa.

Hakuvälilehti alustetaan sisältämään QWebView-instanssi ja sen käyttöliittymäkomponentit kytketään hakutoimintoihin. Myöskin hakusivulle asetetaan setLinkDelegationPolicy-funktiolla QWebPage::DelegateAllLinks-ominaisuus, jotta klikatut hakutulokset avataan ohjeselaimeen, eikä niitä lähetetä ulkoiselle selaimelle avattavaksi. Hakumoottorin tarjoama edistynyt hakutoiminto, engl. advanced search, on piilotettu, koska ohjeselaimessa yksinkertainen hakutoiminto on riittävä. Ohjemateriaalin hakutoiminto toteutetaan käytännössä käyttäen suoraan Qt Help -kirjaston tarjoamaa hakumoottoria. Hakumoottori suorittaa ohjesivujen etsinnän käyttäjän antamien haku-

termien perusteella asynkronisesti, ja emittoi signaalin kun hakutulokset ovat valmiit. Hakutulokset voidaan pyytää hits-funktion avulla, joka palauttaa listan löydettyjen ohjesivujen nimistä ja niiden poluista. Nämä voidaan esittää käyttäjälle esimerkiksi generoimalla yksinkertainen HTML-sivu, ja asettamalla se hakutulossivun sisällöksi.

Oikealla hiiren painikkeella ohjeselaimessa klikattaessa näytettävä kontekstivalikko on toteutettu QMenu- ja QAction -luokkien avulla. Klikkauksesta syntyvä signaali on kytketty välilehti-widgetistä ja sisältöalueesta käsittelijäfunktioon, joka luo ja näyttää valikon käyttäjälle. Esimerkiksi oikealla hiiren painikkeella klikattaessa sisältöalueen välilehteä, näytetään vaihtoehdot uuden välilehden avaamiseen ja nykyisen välilehden sulkemiseen. Sisältöaluetta oikealla hiiren painikkeella klikattaessa valikossa näytetään vaihtoehdot esimerkiksi linkin avaamiseen uudessa välilehdessä ja sivuhistoriassa liikkumiseen.

Välilehden otsaketekstin, eli sivun otsikon, asettaminen tehdään etsimällä sivun HTML-sisällöstä title-elementti. Jos title-elementtiä ei löydy, asetetaan otsikoksi sivun polku. Title-elementtiä käytetään yleisesti HTML-sivuissa otsikon asettamiseen, ja tavallisissakin verkkoselaimissa välilehden nimi asetetaan samaan tapaan. Ohjeprojektissa olevan käytännön mukaan jokaisella ohjesivulla tulee olla sivun otsikko määriteltynä. Sivuhistoriassa liikkuminen toteutetaan QWebView-luokan back- ja forward -funktioilla. Ohjesivun tulostamiseen käytetään QPrinter- ja QPrintDialog -luokkia, jotka tarjoavat suoraan nykyisen ohjesivun tulostamiseen liittyvän toiminnallisuuden. Suurennus, pienennys ja suurennuksen palauttaminen oletukseen delegoidaan CContentViewer-luokalle.

Ohjeselaimesta löytyy funktio ohjesivun avaamiseen HelpID-tunnisteen perusteella. Tätä kutsutaan CHelp-luokan tapahtumasuodattimesta, kun käyttäjä on painanut F1-näppäintä ja klikannut käyttöliittymäkomponenttia, jolla on ohjemateriaalia saatavilla. Funktio käyttää CHelpEngine-luokan linksForIdentifier-funktiota, jolta se saa HelpID-tunnistetta vastaavat ohjemateriaalipolut. Näistä ensimmäinen avataan ohjeselaimeen, koska ei ole tarkoituksenmukaista, että HelpID-tunniste viittaa useampaan kuin yhteen ohjesivuun. Varsinaisen sivun sisällön lataamisen suorittaa CContentViewer-luokka, CHelpEngine-luokan avustuksella. Ohjesivun lataamiseen liittyvä sekvenssi on tarkemmin esitelty osiossa 4.4.

4.3.6 CDitaFileWriter-luokka

CDitaFileWriter-luokka lisätään osaksi kohdesovelluksen nykyistä applikaatiomuokkainta. Luokan vastuulla on ohjepaketissa käytettävien DITA-runkojen generointi applikaatioiden perustietojen pohjalta. Luokka tarjoaa DITA-runkojen tuottamiseen tarvittavan toiminnallisuuden ja oleellisten tietojen keräämisen suoraan applikaatioiden parametreista. Applikaatiomuokkainta ja generoitavia DITA-runkoja on esitelty tarkemmin osios-

sa 4.1.6 ja liitteessä A.

Palataksemme aiemmin luvussa 2 esillä olleeseen kukankastelujärjestelmäesimerkkiin, applikaatiomuokkainta käytettiin siinä ohjausapplikaation XML-kuvauksen luomiseen. DITA-runkojen generoinnin lisäämisen jälkeen, applikaatiomuokkain tuottaa myös ohjemateriaalirungot kukankastelujärjestelmän parametreista. Kun näitä parametrien perustietoja vielä täydennetään lisäämällä ohjeita niiden virittämiseen, saadaan aikaan loppukäyttäjälle hyödyllistä ohjemateriaalia. Loppukäyttäjän ei tällöin tarvitse tuntea ohjausapplikaation sisäistä toteutusta, vaan hän pystyy säätämään järjestelmää täysin ohjemateriaalin perusteella.

Käytännössä luokka toteuttaa rajapinnan DITA-muotoisten tiedostojen tuottamiseen. Toteutus käyttää QXmlStreamWriter-luokkaa DITA-tiedostojen kirjoittamiseen, koska pohjimmiltaan nekin ovat XML-tiedostoja. QXmlStreamWriter-luokka tarjoaa yksinkertaisen rajapinnan XML-muotoisten tiedostojen tuottamiseen, ja tuotetut tiedostot ovat automaattisesti sisennettyjä ja järkevästi luettavissa tavallisellakin tekstieditorilla. CDitaFileWriter-luokalla ensin avataan tiedostokahva tiedostoon, johon ollaan kirjoittamassa DITA-sisältöä. Kun tiedosto on avattu, voidaan sinne luoda taulukkosivu, parametrisivu, konseptisivu tai ditamap-rakenne. Jokaisen eri DITA-sivutyypin rajapinta koostuu kolmesta eri osasta: otsikkotiedoista, sisällöstä ja tiedoston lopetuksesta. Näiden vaiheiden jälkeen tiedostokahva voidaan sulkea, ja tiedosto sisältää validin DITA-muotoisen sivun. Tässä validiudella tarkoitetaan rakennetta, joka täyttää DITA XML -spesifikaation vaatimukset ja käyttää ainoastaan siinä määriteltyjä, tai siitä erikoistettuja, elementtejä. Tarvittaessa rajapintaan on yksinkertaista lisätä uusia sivutyyppejä, mutta toistaiseksi nämä tyypit riittävät applikaatioiden ohjemateriaalin tuottamiseen.

Jokaisella sivutyypillä on oma käyttötarkoituksensa applikaation ohjemateriaalissa, ja yhdessä ne muodostavat kattavan rungon applikaation ohjemateriaalille. Taulukkosivu alustetaan lisäämällä siihen DITA-aiheen referenssitunniste, sivun otsikko ja taulukon otsikko. Näistä referenssitunnistetta voidaan käyttää DITA-aiheiden toisiinsa sisällyttämiseen ja viittaamiseen. Taulukkosivua käytetään parametrilistauksena, joten alustuksen jälkeen siihen voidaan lisätä riveittäin parametreja. Jos parametri sisältää aliparametreja, kuten monta parametria sisältävän tietueen tapauksessa, lisätään ne myös rekursiivisesti taulukon riveiksi sopivasti sisennettyinä. Parametrin nimen lisäksi niistä lisätään perustiedot rivin sarakkeiksi. Esimerkki ohjepakettiin prosessoidusta taulukkosivusta on nähtävissä kuvassa 4.5. Parametrisivu toimii samantapaisesti, mutta taulukon sijaan sivulle eritellään yhden parametrin tiedot. Parametrisivu on pohjimmiltaan konseptisivu, joka sisältää parametrin tiedot. Ditamap sisältää otsikkotietojensa jälkeen viittauksia toisiin DITA-aiheisiin topicref-elementeillä, ja ne muodostavat dokumentin sisällyksen. Konseptisivu on aiempia geneerisempi, joten siihen voidaan vapaasti kirjoittaa DITA-muotoista sisältöä.

DITA-sivujen generointia käytetään toistaiseksi vain applikaatioiden ohjemateriaalien generointiin. Jatkossa toiminnallisuutta on mahdollista laajentaa tuottamaan DITA-runkoja myös muille kohdesovelluksen osille. Esimerkiksi kohdesovelluksessa luodun järjestelmäpaketin perustiedoista voitaisiin generoida DITA-runkoja. Liitteessä A on tarkemmat esimerkit kaikista edellä esitellyistä DITA-sivuista. Osiossa 4.1.6 on esitelty eri DITA-tiedostojen käyttötarkoitukset.

4.3.7 CHelpNetworkAccessManager-luokka

CHelpNetworkAccessManager-luokan kantaluokka QNetworkAccessManager mahdollistaa applikaation lähettää verkon yli pyyntöjä ja vastaanottaa niihin vastauksia [16]. Ohjejärjestelmän tapauksessa halutaan kuitenkin tehdä resurssipyyntöjä paikalliselle ohjepaketille, eikä verkon yli. Tästä syystä QNetworkAccessManager-luokasta periytetään CHelpNetworkAccessManager-luokka, jonka toiminnallisuus uudelleenmääritellään. Sen sijaan, että tehtäisiin verkkopyyntö, käydäänkin qthelp-protokollan osoitteiden mukaiset ohjesivut noutamassa paikallisesta ohjepaketista. CHelpEngine-luokka tarjoaa tarvittavat työkalut ohjesivujen noutamiseen ohjepaketista qthelp-osoitteen tai HelpID-tunnisteen perusteella.

Ohjepakettiin kohdistuvan verkkopyynnön vastauksena annetaan CHelpNetworkReply-luokan instanssi, joka sisältää varsinaisen vastausdatan. Ohjesivupyynnön tapauksessa CHelpNetworkReply-luokalle annetaan sitä rakennettaessa CHelpEngine-luokalta haettu ohjesivun sisältö ja sisällön MIME-tyyppi. MIME-tyyppi kuvaa binäärimuotoisen sisällön tyyppiä, esimerkiksi onko data tyypiltään tekstiä vai kuva, jotta binäärimuotoinen data pystytään käsittelemään sopivalla tavalla sen esittämiseksi [16]. CHelpNetworkAccessManager-luokka sisältää taulukon käytetyistä MIME-tyypeistä, joista pyritään valitsemaan parhaiten datan tyyppiä kuvaava vaihtoehto. MIME-tyyppejä voitaisiin myös vaihtoehtoisesti hakea QMimeDatabase-luokalta. MIME-taulukon ja sisällön URL-osoitteen perusteella päätellään, minkä tyyppistä sisältöä ollaan palauttamassa. Jos sopivaa MIME-tyyppiä ei löydy, käytetään application/octet-stream -tyyppiä, joka tarkoittaa geneeristä binääridataa.

4.3.8 CHelpNetworkReply-luokka

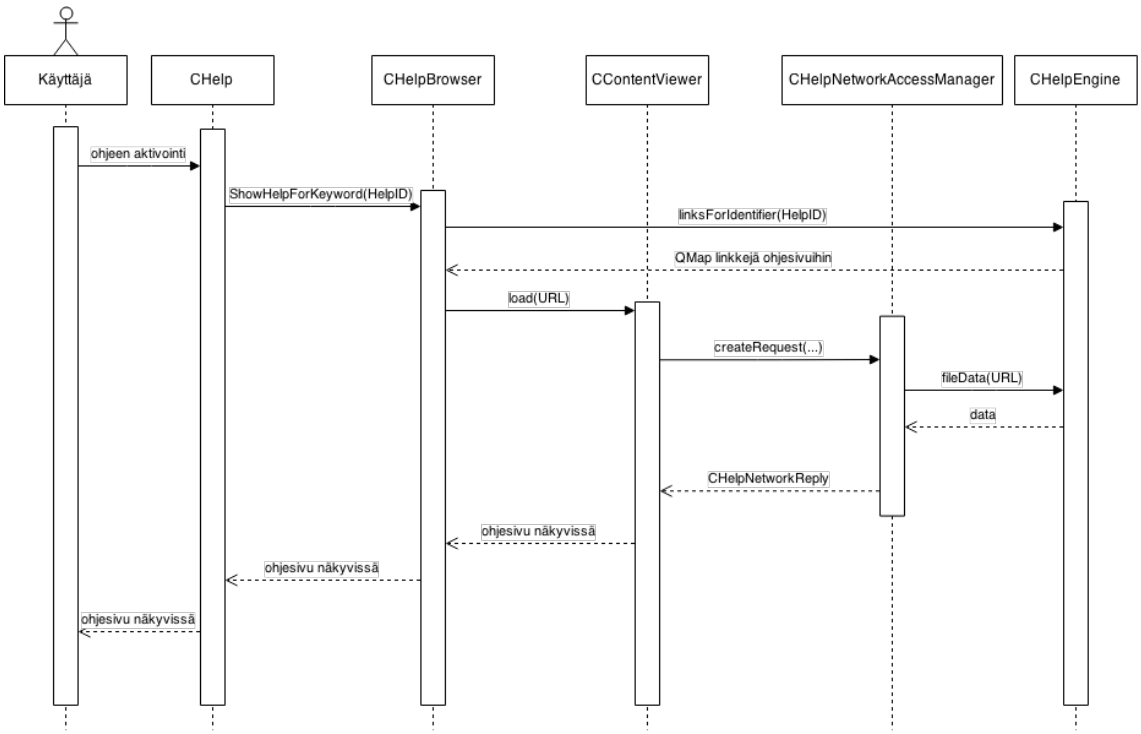
QNetworkReply-luokka sisältää vastauksen ja otsikkotiedot QNetworkAccessManager-luokan avulla lähetettyyn verkkopyyntöön [16]. CHelpNetworkReply-luokkaan tehdään rakentaja, jonka avulla olion sisällöksi voidaan suoraan asettaa ohjepaketista noudettu ohjemateriaali. CHelpNetworkAccessManager-luokan instanssi luo

CHelpNetworkReply-luokasta vastausinstanssin verkkopyynnön käsittelyn yhteydessä, eikä CHelpNetworkReply-luokan vastuulla ole kuin datasäiliönä toimiminen.

Käytännössä ohjejärjestelmä käyttää CHelpNetworkReply-luokkaa vain tietorakenteena CHelpEngine-luokalta haetun ohjemateriaalin välittämiseen eteenpäin. Vastauksessa on oleellista oheistietoa datan lisäksi. Esimerkiksi siitä löytyy tieto onnistuiko kysely ja mistä polusta sisältö haettiin. Verkkopyynnön vastauksen tilaa voi seurata vastaanottamalla QNetworkReply-luokalta perittyjä signaaleja. Näihin lukeutuu tärkeimpinä error- ja finished-signaalit. Error-signaali ilmaisee tapahtuiko pyynnön suorittamisessa virhe ja finished-signaali ilmaisee pyynnön asynkronisen suorittamisen valmistumisen.

4.4 Ohjesivun lataamissekvenssi

Ohjejärjestelmän rakenne ja sen luokkien vastuualueet sekä toteutustavat esiteltiin edeltävissä osioissa. Seuraavaksi tarkastellaan miten kontekstisensitiivisen ohjeen aktivoinnista seuraava tapahtumaketju käytännössä kulkee ohjeliitännäisen läpi. Käsitellään esimerkkinä tilanne, jossa käyttäjä on F1-näppäintä painamalla ja hiiren painiketta klikkaamalla aktivoinut käyttöliittymäkomponentin ohjeen. Oletetaan yksinkertaisuuden vuoksi, että käyttöliittymäkomponentilla on HelpID-tunniste asetettuna, ja sen viittaama ohjesivu on saatavilla nykyisessä ohjepaketissa.



Kuva 4.10. *Kontekstisensitiivisen ohjeen aktivoinnin sekvenssikaavio.*

Kuvassa 4.10 on näkyvissä kontekstisensitiivisen ohjeen aktivointiin liittyvä sekvenssikaavio. Kaaviosta nähdään miten käyttäjän aktivoima ohjepyyntö kulkee ohjeliitännäisen luokkien läpi. CHelp-luokan tapahtumasuodattimen vastaanotettua ohjeen aktivointi, se kutsuu CHelpBrowser-luokan ShowHelpForKeyword-funktiota, antaen sille parametrina aktivoidun HelpID-tunnisteen. ShowHelpForKeyword-funktiossa CHelpBrowser-luokka hakee HelpID-tunnistetta vastaavat ohjesivujen osoitteet CHelpEngine-luokalta linksForIdentifier-funktiolla, antaen edelleen HelpID-tunnisteen parametrina. Vastauksena saadaan QMap-tietorakenne, joka on avainnettu ohjesivun otsikon perusteella ja arvona on sivun qthelp-protokollan mukainen URL-osoite. Listasta valitaan ensimmäinen ohjesivu ja kutsutaan viimeksi aktiivisena olleen sisältövälilehden CContentViewer-luokan load-funktiota ohjesivun URL-osoite parametrina.

Koska CContentViewer-luokan rakentajassa on asetettu QWebPage-luokan setNetworkAccessManager-funktion avulla CHelpNetworkAccessManager-luokan instanssi verkkopyyntöjä hallinnoimaan, päätyy resurssin latauspyyntö CHelpNetworkAccessManager-luokan instanssille. Tämän avulla ohjesivun lataus pysyy ohjeliitännäisen sisällä, ja sivu voidaan noutaa ohjepaketista kutsumalla CHelpEngine-luokan fileData-funktiota. Ohjesivun data haetaan fileData-funktiossa URL-osoitteen perusteella, eikä tässä vaiheessa enää tarvita HelpID-tunnistetta. Ohjepaketista haettu ohjesivun data säilötään CHelpNetworkReply-luokan instanssiin, joka annetaan paluuarvona takaisin kutsujalle.

Nyt CContentViewer-luokan instanssilla on hallussaan ohjesivun data, ja se voidaan asettaa näkyville ohjeselaimeen. CContentViewer-luokka on perinyt kaiken ohjesivujen esittämiseen tarvitun toiminnallisuuden QWebView-luokalta, eikä sivujen esittämistä varten tarvitse toteuttaa omaa toiminnallisuutta. Sekvenssissä ShowHelpForKeyword on ainoa itse toteutettu funktio. Kaikki muut esillä olleet funktiot on peritty Qt-ohjelmistokehyksen tarjoamilta luokilta.

Tilanteessa, jossa käyttäjä olisi klikannut ohjetilan ollessa aktiivisena käyttöliittymäkomponenttia, jolla ei ole ohjemateriaalia saatavilla, olisi ohjeliitännäinen poistunut ohjetilasta. Pyyntö ei olisi edennyt CHelp-luokkaa pidemmälle, koska tapahtumasuodatin jo tarkastaa, onko ohjemateriaalia saatavilla. Tällöin hiiren kursori olisi myös palannut normaalitilaan, eikä käyttäjälle olisi näytetty ohjeselainta.

5. YHTEENVETO

Toteutettu ohjejärjestelmä sisältää noin 4000 riviä Qt/C++-koodia, joten se on varsin tiivis ja helposti ylläpidettävä kokonaisuus. Työn oleellisin haaste oli järjestelmän integraatio, eli ohjeprosessin osien sovittaminen toisiinsa ja olemassa olevaan ympäristöön. Lopputulos on hyvin samankaltainen kuin osiossa 2.1 esitellyt olemassa olevat ohjejärjestelmäratkaisut.

Toteutetussa ohjejärjestelmässä on huomattavaa samankaltaisuutta taustatutkimuksessa esitellyn Eclipsen ohjejärjestelmän kanssa. Molempien järjestelmien ohjeselaimet ovat hyvin samankaltaiset, niin toiminnallisuuden kuin ulkonäön osaltakin. Eclipsen ohjesivuihin viittaamiseen käyttämät kontekstitunnisteet ovat myös läsnä, niitä vain kutsutaan HelpID-tunnisteiksi. Toisin kuin Eclipsessä, ei toteutetussa ohjejärjestelmässä ole ohjemateriaalin pikakatseluikkunoita, vaan ohje avataan aina suoraan ohjeselaimeen. Ohjemateriaalia ei voi myöskään laajentaa kohdesovelluksen liitännäisten avulla, vaan kaikki ohjemateriaali on yhteisessä ohjepaketissa.

5.1 Tulokset

Kokonaisuudessaan kaikki osiossa 2.2 asetetut tavoitteet täyttyivät ja tilaaja hyväksyi lopputuloksen. Tavoitteet on eritelty toteutumineen taulukossa 5.1. Taulukossa olevat tavoitteiden numerot viittaavat osioon, jossa tavoite on esitelty.

Taulukko 5.1. *Asetetut tavoitteet ja niiden toteutumat.*

Numero	Tavoite	Tulos
2.2.1	Perustoiminnallisuus	Toteutui
2.2.2	Käytettävyys	Toteutui
2.2.3	Ohjeselain	Toteutui
2.2.4	Käyttäjäryhmät	Toteutui

Ohjejärjestelmän kehitykseen ei juuri vaikuttanut budjetti- tai aikatauluvaatimuksia, vaan kehitys oli hyvin iteratiivista. Jokaisen iteraation jälkeen lopputulos arvioitiin ja päätettiin mitä tehdään seuraavaksi. Ohjejärjestelmän kehitysprojekti päätettiin, kun riittävä määrä

perustoiminnallisuutta oli valmiina.

Jos ohjejärjestelmän kehitystä olisi jatkettu hieman pidemmälle, olisi dataroolien tarpeellisuutta voitu vielä miettiä. Osiossa 2.1 mainittiin, että järjestelmään tarvitaan yhtenäinen toteutustapa, jota ei tarvitse erikseen sovittaa järjestelmän eri osiin. HelpID-tunnisteet ovat hyvin geneerinen mekanismi, mutta niitä säilötään kahden eri mekanismin avulla. Tämä rikkoo hieman vaatimusta yhtenäisestä toteutuksesta. Olisi parempi, jos HelpID-tunnisteet voitaisiin aina säilöä luokkien ominaisuuksissa, eikä MVD-mallin mukaisissa käyttöliittymäkomponenteissa tarvitsisi käyttää datarooleja tunnisteen säilömiseen.

Perustoteutuksen valmistuttua kohdesovelluksen toimintaan lisättiin säikeistystä, ja tämän seurauksena myös CHelpEngine-luokan toteutus täytyi muuttaa säieturvalliseksi. Tämä toteutettiin lisäämällä luokalle QMutex, joka poissulkee säikeitä käsittelemästä jaettua tietoa samanaikaisesti. Säieturvallisuus tuleekin huomioida aina kun on tiedossa, että toteutettavaa luokkaa tullaan mahdollisesti käyttämään säikeistetyssä ympäristössä. Qt-ohjelmistokehys on perusluonteeltaan säikeistystä tukeva ja raskasta suoritusta ei pitäisi tehdä käyttöliittymäsäikeessä, koska se lukkiuttaa käyttöliittymän. Oikea tapa suorittaa raskasta laskentaa on käyttää QThread tai QConcurrent-luokkia prosessoinnin säikeistämisessä. QFuture-luokka auttaa myös asynkronisessa ohjelmoinnissa, ja sen avulla voidaan monella tapaa hallita rinnakkaista toiminnallisuutta. Tosin Qt-signaalimekanismi toimii myös monisäikeisessä ympäristössä, eikä säikeiden väliseen viestintään tai synkronointiin tarvita sen monimutkaisempia mekanismeja. Ohjejärjestelmän säieturvallisuus olisi ollut hyvä huomioida suunnittelussa jo lähtökohtaisesti.

Ohjejärjestelmä on perustoiminnallisuudeltaan valmis, mutta laaja käyttöönotto tarvitsee jatkokehitystä. Järjestelmä on otettu käyttöön sisäisesti, ja on vielä vuoden 2015 alussa koekäytössä. Varsinainen laaja käyttöönotto on pitkä prosessi, ja vaatii huomattavan määrän koulutusta ohjemateriaalin kirjoittajille DITA XML -merkkauskielen ja oXygen XML -sovelluksen käytöstä. Vanhoja ohjedokumentteja muunnetaan hitaasti aina tarvittaessa uuteen DITA XML -muotoon, ja uudet dokumentit pyritään kirjoittamaan suoraan kyseiseen muotoon.

5.2 Jatkokehitys

Ohjejärjestelmälle on suunniteltu paljon enemmän toiminnallisuutta, kuin mitä työssä toteutettiin. Näitä jatkokehitysideoita, ja niistä saavutettavia hyötyjä, on esitelty tässä osiossa.

Toistaiseksi kaikilla järjestelmäpaketeilla on sama ohjepaketti. Tämä ei ole käytännöllistä, vaan jokaiselle järjestelmälle täytyy olla omat käyttäjätasojen mukaiset ohjepakettin-

sa. Jokaisen järjestelmän konfiguraatio ja ohjausapplikaatio ovat ainakin osittain erilaiset, joten myös ohjepakettien sisällöissä on eroja. Käytännössä tämä ei tarvitse suuria muutoksia ohjejärjestelmään, vaan lähinnä ohjepaketin prosessointia tulee kehittää siten, että se tukee useamman eri ohjepaketin rakentamista. Vaihtoehtoisesti jokaiselle järjestelmälle voidaan tehdä oma juuritason ditamap-tiedostonsa, ja erikseen rakentaa näistä omat ohjepakettinsa nykyisellä prosessointimekanismilla.

Ohjemateriaalin versiointimekanismi on vielä toteuttamatta, jonka avulla samasta ohjemateriaalista voisi olla olemassa samaan aikaan useita eri versioita. Osa automaatiojärjestelmän konfiguraatiosta on versioitu, ja jokaiselle versiolle tarvitaan omat ohjemateriaalinsa, jotka ovat kuitenkin osittain samanlaisia muiden versioiden välillä. Ohjemateriaaleissa olisi hyödyllistä olla upotettuna versioinformaatiota, jonka perusteella voidaan valita järjestelmäkohtaisesti ohjemateriaalista näytettävä versio. Tällöin voitaisiin jopa ohjesivukohtaisesti valita sopiva versio sivusta näytettäväksi käyttäjälle. Tähän varmaankin paras ratkaisu on suodattaa ohjejärjestelmän toimesta kohdesovelluksessa ajonaikaisesti ohjemateriaalista näytettävää versiota.

Ohjepaketti voitaisiin jatkossa salata, jos se sisältää arkaluontoista materiaalia. Salattua Qt-ohjepakettia tulisi pystyä lukemaan suoraan, ilman että sitä puretaan kokonaan väliaikaistiedostoon. Väliaikaistiedostot ovat selkeä tietoturvaongelma, eikä niiden kopiointia voida juuri estää. Tehokas salaus tarvitsee jonkin välikerroksen CHelpEngine-luokan ja ohjemateriaalipaketin välille, joka mahdollistaa ohjepaketin lukemisen suoraan, ilman väliaikaistiedostoon purkamista.

Jos kohdesovellukseen toteutetaan jo suunniteltu monen yhtäaikaisen järjestelmän tuki, tarvitsee ohjejärjestelmän CHelpEngine-luokan Singleton-suunnittelumalli vaihtaa Multiton-suunnittelumalliksi. Jokaisella avoimella järjestelmällä tulee olla oma ohjepakettinsa käytössä, ja jokaiselle näistä täytyy olla myös oma instanssinsa CHelpEngine-luokasta.

Ohjepaketit voitaisiin ladata tarvittaessa palvelimelta, eikä niitä tarvitsisi toimittaa kohdesovelluksen asennuspaketin mukana. Ei ole mitenkään mielekästä toimittaa kaikkien mahdollisten järjestelmäpakettien ohjepaketteja kohdesovelluksen asennuspaketin mukana. Tämä pienentäisi asennuspaketin kokoa ja parantaisi tietoturvaa, kun asiakkaille voidaan jaella ainoastaan paketit joihin heillä on käyttöoikeus. Tämän toteutukseen riittäisi todennäköisesti HTTPS-yhteyden yli tapahtuva tiedostonjako, joka tarkistaa käyttöoikeudet ennen tiedostojen lähettämistä.

LÄHTEET

[1] Abel, S. Yes, You Can Do DITA With Microsoft Office and SharePoint! 2011. URL: http://thecontentwrangler.com/2011/06/01/yes-you-can-do-dita-with-microsoft-office-and-sharepoint (viitattu 10. 12. 2014).

[2] Carpenter, S. G. Implementing DITA XML in a Production Environment. Proceedings of the 20th Annual International Conference on Computer Documentation, SIGDOC '02. Toronto, Ontario, Canada, ACM, 2002, pp. 17–19. ISBN: 1-58113-543-2. DOI: 10.1145/584955.584958. URL: http://doi.acm.org/10.1145/584955.584958.

[3] DITA Exchange ApS. DITA Exchange. 2014. URL: http://www.ditaexchange.com/ (viitattu 10. 12. 2014).

[4] DITA Open Toolkit project. DITA Open Toolkit. 2014. URL: http://www.dita-ot.org/ (viitattu 06. 12. 2014).

[5] Eclipse Foundation. Eclipse documentation. 2014. URL: http://help.eclipse.org (viitattu 13. 12. 2014).

[6] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p. ISBN: 0-201-63361-2.

[7] Halsted, K. L. & Roberts, J. H. Eclipse Help System: An Open Source User Assistance Offering. Proceedings of the 20th Annual International Conference on Computer Documentation, SIGDOC '02. Toronto, Ontario, Canada, ACM, 2002, pp. 49–59. ISBN: 1-58113-543-2. DOI: 10.1145/584955.584964. URL: http://doi.acm.org/10.1145/584955.584964.

[8] Harrison, N. The Darwin Information Typing Architecture (DITA): Applications for Globalization. International Professional Communication Conference proceedings, IEEE, 2005, pp. 115–121.

[9] Kearsley, G. Online Help Systems: Design and Implementation. Intellect Books, 1988. 126 p. ISBN: 9780893914721.

[10] Larner, I. Information development with DITA. 2004. URL: http://xml.coverpages.org/dita.html%5C#Nokia1119 (viitattu 10. 12. 2014).

[11] Matejka, J., Grossman, T. & Fitzmaurice, G. Ambient Help. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11. Vancouver, BC, Canada, ACM, 2011, pp. 2751–2760. ISBN: 978-1-4503-0228-9. DOI:

10.1145/1978942.1979349. URL: http://doi.acm.org/10.1145/1978942.1979349.

[12] Microsoft. Microsoft HTML Help 1.4. 2012. URL: https://msdn.microsoft.com/en-us/library/ms670169(v=vs.85).aspx (viitattu 08. 02. 2015).

[13] OASIS Open. Darwin Information Typing Architecture (DITA) Version 1.2. 2014. URL: http://docs.oasis-open.org/dita/v1.2/os/spec/DITA1.2-spec.html (viitattu 06. 12. 2014).

[14] LaTeX project. LaTeX – A document preparation system. 2014. URL: http://latex-project.org/contact.html (viitattu 10. 12. 2014).

[15] Qt Project. Qt Project. 2014. URL: http://qt-project.org/ (viitattu 15. 11. 2014).

[16] Qt Project. Qt Project Documentation. 2014. URL: http://qt-project.org/doc (viitattu 15. 11. 2014).

[17] Qt Project. The Meta-Object System. 2014. URL: http://qt-project.org/doc/qt-5/metaobjects.htmll (viitattu 10. 12. 2014).

[18] Qt Project. The Property System. 2014. URL: http://qt-project.org/doc/qt-5/properties.html (viitattu 10. 12. 2014).

[19] Qt Project. The Qt Help Framework. 2014. URL: http://qt-project.org/doc/qt-5/qthelp-framework.html (viitattu 06. 12. 2014).

[20] Syncro Soft. DITA Support. 2014. URL: http://www.oxygenxml.com/doc/brochures/Datasheet_DITA_Support.pdf (viitattu 06. 12. 2014).

[21] Syncro Soft. Oxygen XML Author Documentation. 2014. URL: http://www.oxygenxml.com/doc/ug-author/ (viitattu 06. 12. 2014).

[22] W3C. Extensible Markup Language (XML). 2014. URL: http://www.w3.org/XML/ (viitattu 13. 12. 2014).

[23] W3C. XSL Transformations (XSLT). 1999. URL: http://www.w3.org/TR/xslt (viitattu 13. 12. 2014).

Liite A: ESIMERKKIAPPLIKAATION DITA-RUNGOT

Tässä liitteessä on käytännön esimerkki generoiduista applikaation DITA-rungoista. Osioiden otsikoina toimivat tiedostojen nimet, joista parametrikohtaiset DITA-sivut ovat params-kansion alla.

EsimerkkiApplikaatio_v1.0.0.ditamap

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE map PUBLIC "urn:pubid:esimerkki.com:doctypes:dita:map" "map.dtd">
3 <map>
      <title>Esimerkkiapplikaatio</title>
5     <topicref href="EsimerkkiApplikaatio_v1.0.0.dita">
          <topicref href="EsimerkkiApplikaatio_config_table_v1.0.0.dita">
7             <topicref href="params/Application-EsimerkkiApplikaatio-C-s8_matrix_v1.0.0.
                  dita"/>
              <topicref href="params/Application-EsimerkkiApplikaatio-C-u8_value_v1.0.0.
                  dita"/>
9         </topicref>
      </topicref>
11 </map>
```

Ohjelma A.1. Applikaation ditamap-koontitiedosto.

Ohjelmassa A.1 on applikaation ditamap-koontitiedosto, joka linkittää applikaation DITA-aiheet yhteen. Tämä ditamap voidaan sisällyttää ohjepaketin ylimpään ditamap-tiedostoon, jolloin applikaation ja sen parametrien ohjesivut otetaan mukaan ohjepaketin generointiin. Ditamap-tiedoston pääsisältö on map-elementin alla. Ensin on DITA-aiheen otsikko, joka tässä tapauksessa kuvaa ditamap-koonnin sisältöä ja applikaatiota kokonaisuudessaan. Otsikon jälkeen topicref-elementit kuvaavat hierarkkisen rakenteen, ja se vastaa suoraan ohjemateriaalin rakennetta, joka näkyy lopullisessa ohjepaketissa. Elementit sisältävät href-attribuutin, joka on linkki sisällytettävään DITA-tiedostoon.

Tässä esimerkissä muodostuu siis seuraavanlainen rakenne applikaation ohjeen sisällöksi: EsimerkkiApplikaatio_v1.0.0.dita -tiedosto on applikaation etusivu, jonka alla on EsimerkkiApplikaatio_config_table_v1.0.0.dita -parametritaulukkotiedosto. Parametritaulukkotiedoston alta löytyy vielä parametrien yksilölliset kuvaussivut, eli params/Application-EsimerkkiApplikaatio-C-s8_matrix_v1.0.0.dita ja params/Application-EsimerkkiApplikaatio-C-u8_value_v1.0.0.dita. Tämä hierarkkinen rakenne on näkyvissä ohjeselaimeen avatun prosessoidun ohjepaketin sisällysluettelossa.

EsimerkkiApplikaatio_v1.0.0.dita

```
1 <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE concept PUBLIC "urn:pubid:esimerkki.com:doctypes:dita:concept" "concept.dtd">
3 <concept id="Application-EsimerkkiApplikaatio">
      <title>Esimerkkiapplikaatio</title>
5     <shortdesc> Esimerkkiapplikaation kuvaus.</shortdesc>
      <conbody/>
7 </concept>
```

Ohjelma A.2. Applikaation yleiskuvaus.

Ohjelmassa A.2 on applikaation yleiskuvauksen sisältävä DITA-tiedosto. Listaus alkaa jälleen syntaksi- ja dokumenttityyppimäärityksillä: ?xml- ja !DOCTYPE-elementit. Näiden jälkeen määritellään konseptityyppinen DITA-aihe, engl. concept, jolle annetaan myös attribuuttina sen ID-tunniste. ID-tunnisteen avulla voidaan viitata DITA-aiheiden eri osiin ja lainata tai sisällyttää niiden sisältöä muihin sivuihin. Konseptin sisältönä on sivun otsikko eli title-elementti, lyhyt kuvaus eli shortdesc-elementti ja varsinainen sivun sisältö, joka on tässä tapauksessa tyhjä conbody-elementti.

EsimerkkiApplikaatio_config_table_v1.0.0.dita

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE reference PUBLIC "urn:pubid:esimerkki.com:doctypes:dita:reference" "reference.dtd">
<reference id="Application-EsimerkkiApplikaatio-C">
    <title>Esimerkkiapplikaatio – asetusparametrit</title>
    <refbody>
        <section>
            <table>
                <title>Configuration</title>
                <tgroup cols="5">
                    <colspec colwidth="4.55*"/>
                    <colspec colwidth="1.98*"/>
                    <colspec colwidth="1*"/>
                    <colspec colwidth="1*"/>
                    <colspec colwidth="4.7*"/>
                    <thead>
                        <row>
                            <entry align="center">UI Name</entry>
                            <entry align="center">Default value</entry>
                            <entry align="center">Min</entry>
                            <entry align="center">Max</entry>
                            <entry align="center">Symbol name</entry>
                        </row>
                    </thead>
                    <tbody>
                        <row>
                            <entry><ph id="s8_matrix">
                                    <parmname>s8 matrix (3 x 3)</parmname>
                                </ph>
                            </entry>
                            <entry align="center">0 V</entry>
                            <entry align="center">0 V</entry>
                            <entry align="center">0 V</entry>
                            <entry align="center">s8_matrix</entry>
                        </row>
                        <row>
                            <entry><ph id="u8_value">
                                    <parmname>u8 value</parmname>
                                </ph>
                            </entry>
                            <entry align="center">0</entry>
                            <entry align="center">0</entry>
                            <entry align="center">0</entry>
                            <entry align="center">u8_value</entry>
                        </row>
                    </tbody>
                </tgroup>
            </table>
        </section>
    </refbody>
</reference>
```

Ohjelma A.3. Applikaation parametritaulukko.

Ohjelmassa A.3 on applikaation parametritaulukon DITA-kuvaus. Taulukkosivu on referenssimuotoinen DITA-aihe, engl. reference, joka on yleisesti tarkoitettu teknisek-

si hakemistoksi tai muuksi viitemateriaaliksi. Sivun varsinainen sisältö alkaa refbody-elementistä, ja se sisältää taulukon section- ja table-elementeillä kuvattuna. Taulukon otsikko on Configuration, joka kuvastaa sen sisältävän asetusparametreja. Otsikon jälkeen määritellään ryhmä taulukon asetuksia tgroup-elementillä, joka sisältää asetukset taulukon sarakkeiden leveyksille colspec-elementeillä ja colwidth-attribuuteilla kuvattuna. Tämän jälkeen thead-elementin sisällä on kuvattu sarakkeiden otsikkorivien sisältö ja niiden asettelu row- ja entry-elementeillä. Otsikkorivien jälkeen alkaa taulukon varsinaiset sisältörivit tbody-elementillä kuvattuna. Jokainen row-elementti määrittelee yhden rivin ja sisältää kaikki sen rivin sarakesolut. Jokainen solu erotellaan entry-elementillä, joka sisältää solun sisällön sekä align-attribuutissa solun tasauksen. Parametrin nimen sisältävä solu on yksilöity ph-elementin id-attribuutilla, ja sen sisältöä korostetaan parmname-elementillä, joka vaikuttaa esitettävän tekstin ulkonäköön. Jälleen ph-elementin ID-tunniste mahdollistaa yksittäiseen parametrisoluun viittaamisen muista DITA-aiheista.

params/Application-EsimerkkiApplikaatio-C-u8_value_-v1.0.0.dita

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "urn:pubid:esimerkki.com:doctypes:dita:concept" "concept.dtd">
<concept id="Application-EsimerkkiApplikaatio-C-u8_value">
    <title>u8 value</title>
    <shortdesc></shortdesc>
    <conbody>
        <p>
            <dl>
                <dlentry>
                    <dt>Symbol name</dt>
                    <dd>
                        <codeph>u8_value</codeph>
                    </dd>
                </dlentry>
                <dlentry>
                    <dt>Data type</dt>
                    <dd>u8</dd>
                </dlentry>
                <dlentry>
                    <dt>Default value</dt>
                    <dd>0</dd>
                </dlentry>
            </dl>
        </p>
    </conbody>
</concept>
```

Ohjelma A.4. Applikaation u8 value -parametrin kuvaus.

Ohjelmassa A.4 on etumerkittömän 8-bittisen kokonaislukuparametrin yksityiskohtainen tietosivu. Parametrikohtaiset sivut käyttävät jälleen konseptityyppistä DITA-aihetta. Oleellisimpana erona applikaation yleiskuvaussivuun on listarakenne, jolla esitellään parametrin perustiedot. Conbody-sisältöelementin sisällä on tekstikappaletta ilmaiseva p-elementti, ja sen sisällä on dl-elementin mukainen määritelmälistaus, engl. definition list. Jokainen listan jäsen erotellaan dlentry-elementeillä, ja jokaisella jäsenellä on sekä dt-termielementti että dd-määritelmäelementti. Parametrin nimeä on korostettu codeph-elementillä, joka vaikuttaa parametrin nimen esitystapaan.