TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO HONKONEN

TYÖVUON MALLINNUSTYÖKALUN TOTEUTUS

Diplomityö

Tarkastaja: Prof. Timo D. Hämäläinen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 6.4.2016

TIIVISTELMÄ

MIKKO HONKONEN: Työvuon mallinnustyökalun toteutus
Tampereen teknillinen yliopisto
Diplomityö, 57 sivua, 3 liitesivua
Toukokuu 2016
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmoitavat alustat ja laitteet
Tarkastajat: Prof. Timo D. Hämäläinen
Avainsanat: työvuon mallinnus, graafi, Woke

Modernien järjestelmäpiirien suunnitteluprosessi on yhä monimutkaisempaa. Suunnitteluprosessin hallinnassa voi olla avuksi perinteisesti yrityksen prosesseja korkealla tasolla tarkkaileva työvuomallinnus. Työvuomalleja käytetään tyypillisesti analysointiin ja simulointiin, mutta ne voivat myös olla suoraan tosielämän toimintaa ohjaavia.

Suunniteluprosessin mallinnukseen on käytetty olemassaolevia työkaluja kuten UML, YAWL ja BPMN. Nämä työkalut ovat tyypillisesti ominaisuuksiltaan hyvin laajoja, joka hankaloittaa niiden käyttöönottoa ja ymmärrystä. Tässä työssä tavoitteena oli kehittää ominaisuuksiltaan rajattu mutta silti oleelliset tilanteet kattava graafinen työvuomallinnustyökalu, joka soveltuu erityisesti nopeaan työskentelyyn.

Työn tuloksena on avoimen lähdekoodin sovellus Woke. Woke kehitettiin C++-kielellä Qt-ohjelmistokehyksen päälle. Woken kehityksessä keskityttiin uudenlaiseen mallin nopean rakentamisen mahdollistavaan näppäimistösyötteeseen perustuvaan toimintamalliin. Mallin visualisoinnissa keskityttiin erityisesti käyttökokemukseen taulutietokoneilla. Sovellus saavutti sille asetetut tavoitteet, ja tarkastellussa esimerkkitapauksessa työkalu suoriutui positiivisesti muihin vastaaviin työkaluihin verrattuna.

ABSTRACT

MIKKO HONKONEN: Implementation of a workflow modeling tool
Tampere University of Technology
Master of Science Thesis, 57 pages, 3 Appendix pages
May 2016
Master's Degree Programme in Information Technology
Major: Programmable platforms and devices
Examiner: Prof. Timo D. Hämäläinen
Keywords: workflow modeling, graph, Woke

The design process for modern System-on-Chip devices is becoming increasingly complex. Workflow modeling which is traditionally used for observing higher level business processes may be useful for managing this design process. Workflow models are generally used for analysis and simulation, but they can also be used to control real life activity.

There existing tools that have been used to model the design proces, such as UML, YAWL and BPMN. These tools typically have a vast array of features, which makes understanding them and taking them into use more difficult. The aim of this thesis was to design a graphical workflow modeling tool with limited features that can still be used to cover most relevant situations.

The result of this thesis work is an open source software called Woke. Woke was developed in C++ using the Qt-framework. A new kind of keyboard-based interaction model allowing the user to build the workflow model quickly was a core focus when developing Woke. For model visualization, the focus was on usability with tablet devices. The software reached the goals that were set for it, and in a case study the tool performed well compared to existing similar tools.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston Tietotekniikan laitokselle. Erityiskiitokset sekä esimiehenä, diplomityön ohjaajana että tarkastajana toimineelle Timo D. Hämäläiselle toimivasta yhteistyöstä työn toteutuksen aikana ja kärsivällisyydestä työn tarkastajana. Lisäksi haluan kiittää Erno Salmista työssä avustamisesta. Kiitos Myös kaikille työkavereille, erityisesti huonekavereille Joni-Matille, Laurille ja Eskolle.

Haluan myös kiittää perhettäni, sukulaisia ja ystäviä tuesta ja motivoinnista niin diplomityön kuin koko opintojen aikana.

Tampereella 24.5.2016

Mikko Honkonen

SISÄLLYS

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

BF	Breadth-first
BPMI	Business Process Management Initiative
BPMN	Business Process Model and Notation
CSS	Cascading Style Sheets
DF-ALAP	Depth-first, as late as possible
DF-ASAP	Depth-first, as soon as possible
IDE	Integrated Development Environment
MVC	Model-View-Controller
MV	Model/View
OMG	Object Management Group
PNG	Portable Network Graphics
SoC	System-on-Chip
UML	Unified Modeling Language
Woke	Workflow Editor For Kactus 2
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language

1. JOHDANTO

Modernien moniytimisten järjestelmäpiirien (SoC, System-on-Chip) suunnittelu on yhä monimutkaisempaa suunnittelumetodien ja työkalujen kehityksestä huolimatta. Työkalukehykset tyypillisesti sisältävät järjestelmän mallinnusta useilla abstaktiotasoilla, komponenttien uudelleenkäyttöä sekä generaattoreita tarkasteluun ja mallinnuksen parantamiseen.

Suunnittelu vaatii useita työkaluja, ja kunkin työkalun tehokas käyttäminen vaatii syvää tietämystä kyseisestä työkalusta. Käytetyt työkalut ovat usein yhdistelmä vanhoja, kaupallisia, avoimen lähdekoodin sekä itse tehtyjä sovelluksia. Tämä johtaa useisiin vaihtoehtoisiin tehtäviin ja polkuihin jotka on vaikea dokumentoida ja vielä vaikeampia oppia ja käyttää.

Sama ongelma on havaittu myös yhden työkalun käytössä: Kactus2 IP-XACT -työkalun [12] saadessa uusia ominaisuuksia, on yhä vaikeampaa päättää mitä tietoa käyttäjälle näytetään ja milloin. Ratkaisuna oli uusi työkalu suunnitteluprosessin mallintamiseen. Työkalun nimi on Woke (WOrKflow Editor for kactus2, työvuoeditori kactus2:lle), ja sen alkuperäinen tarkoitus oli Kactus2-työkalun avustajien (wizard) mallintaminen. Woke ei kuitenkaan ole Kactus2-riippuva, vaan yleiskäyttöinen editori prosessimallinnukseen. Työkalussa on käytettävyyden kannalta uusia ideoita verrattuna muihin prosessimallinnustyökaluihin.

Vuonna 2013 suoritetussa kyselyssä [7] haastateltiin yli 2000 sulautettujen järjestelmien suunnittelun asiantuntijaa. 17 vaihtoehdosta selvästi yleisin (28 %) teknologiaan liittyvä haaste oli uusien teknologioiden ja työkalujen integrointi. Vertailun vuoksi IDE-työkalujen (integrated development environment, kehitysympäristö) ja SoC-väylien osuus oli vain 2 %.

Desagent OY ja Tampereen teknillinen yliopisto suorittivat vastaavan kyselyn suomalaisille sulautettuja järjestelmiä kehittäville yrityksille vuonna 2013 [10]. Kyselyssä oli mukana kaikkiaan 102 yritystä pienistä ja keskisuurista suuriin. Vain 3 % vas-

taajista kertoi kehitysprosessin olevan täysin mallinnettu. Noin 30 % käytti malleja kuten BPMN 2.0 (Business Process Model and Notation) osaan suunnitteluvuosta. Käytännössä yksikään ei mitannut kehitysprosessin tehokkuutta, suurimmaksi osaksi koska se on liian vaikeaa tai itse vuon selvittäminen voi olla liian aikaavievää.

Näistä tuloksista voidaan päätellä, että suunnitteluprosessin mallinnus on tärkeä tutkimusaihe myös sulautettujen järjestelmien saralla. Myös prosessin tehokkuus on otettava huomioon, ei vain sen lopputulos kuten SoC-arkkitehtuuri.

Luvussa 2 esitellään aiheeseen liittyvää aiempaa työtä. Luvussa 3 esitellään Woke:n käyttämälle metamallille asetetut tavoitteet sekä lopullinen metamalli. Luvussa 4 käsitellään graafin rakennukseen ja visualisointiin liittyvät periaatteet. Luvussa 5 keskitytään sovelluksen käyttöliittymään, luvussa 6 varsinaiseen toteutukseen, ja luvussa 7 lopullista sovellusta arvioidaan sekä verrataan muihin saatavilla oleviin työkaluihin. Lopuksi luvussa 8 esitetään lyhyt yhteenveto koko työstä.

2. TEORIA

Luvussa 2 käsitellään aiheeseen liittyvää aikaisempaa työtä. Aluksi esitellään malleihin ja graafeihin liittyvää termistöä ja teoriaa. Tämän jälkeen esitellään muutamia olemassaolevia työvuon kuvaukseen käytettäviä metamalleja ja kieliä.

2.1 Määrittelyä

Malli on vaihtoehtoinen esitysmuoto todelliselle ilmiölle tai järjestelmälle, jonka tarkoituksena on yksinkertaistaa tätä ja helpottaa asian ymmärtämistä. Malli voi käyttökohteesta riippuen mukailla kuvaamaansa järjestelmää tarkasti, tai vaihtoehtoisesti olla hyvinkin abstrakti kuvaus järjestelmästä.

Työvuomalli kuvaa yrityksen prosesseja. Tällaista mallia voidaan käyttää esimerkiksi prosessien dokumentointiin, analysointiin, simulointiin tai jopa varsinaisen työn ohjaukseen automatisoidussa järjestelmässä.

Tyypillinen esitystapa työvuomallillle on graafi. Graafi koostuu kahdesta peruselementistä: solmut ja kaaret. Solmut kuvaavat graafissa esimerkiksi tiettyä tapahtumaa tai objektia, ja kaaret yhdistävät näitä solmuja. Työvuomalleissa solmut kuvaavat yleisesti työvuon vaiheita ja kaarten avulla kuvataan näiden vaiheiden järjestys.

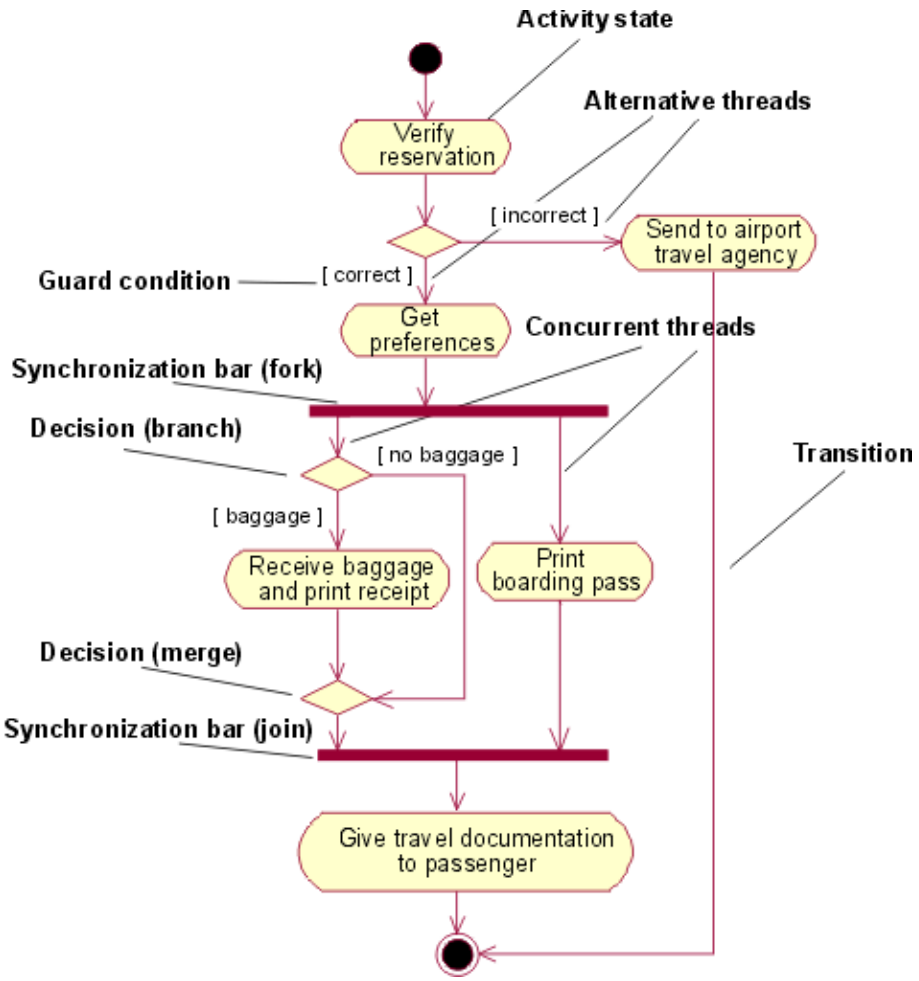
2.2 Muut mallit

Työvuon kuvaukseen on olemassa useita valmiita malleja. Seuraavissa alikohdissa esitellään näistä tarkemmin tilakaavioihin perustuva UML 1.x (Unified Modeling Language), vuokaavioihin perustuva UML 2.x, myös vuokaavioihin perustuva BPMN 2.0 (Business Process Model and Notation) sekä Petri-verkkoihin perustuva YAWL (Yet Another Workflow Language).

2.2.1 UML 1.x

UML 1.x -nimi kattaa UML-kielen versiot 1.0:sta 1.5:een. UML Partners -konsortio aloitti UML-kielen kehityksen vuonna 1996, ja ensimmäinen versio 1.0 esiteltiin OMG-konsortiolle (Object Management Group) standardoitavaksi vuonna 1997. Tätä versiota ei kuitenkaan standardoitu, vaan ensimmäinen virallinen standardoitu versio on myöhemmin samana vuonna esitelty UML 1.1, joka standardoitiin marraskuussa 1997 [14]. Viimeiseksi UML 1.x -versioksi jäi vuonna 2003 standardoitu UML 1.5 [15].

UML 1.x -standardi kattaa yhdeksän erilaista kaaviota, joista tämän työn piirissä oleellisin on prosessien kuvaamiseen tarkoitettu toimintakaavio (activity diagram). Kuvassa 2.1 on esimerkki UML 1.5 -toimintakaaviosta. UML 1.x:n toimintakaavio muodostuu toiminnoista (action) ja päätöksistä (decision), joita yhdistetään nuolilla. Kun suoritus etenee toimintoon, siinä pysytään kunnes toiminto on saatu valmiiksi. Päätös haarauttaa suorituksen kulun ehdollisesti esimerkiksi kahden eri polun välillä. Toiminnossa voi olla useita sisääntuloja, mutta vain yksi ulostulo. Päätöksissä voi olla sekä useita sisääntuloja että ulostuloja. UML 1.x -toimintakaaviolla ei voi kuvata prosessin datavuota.



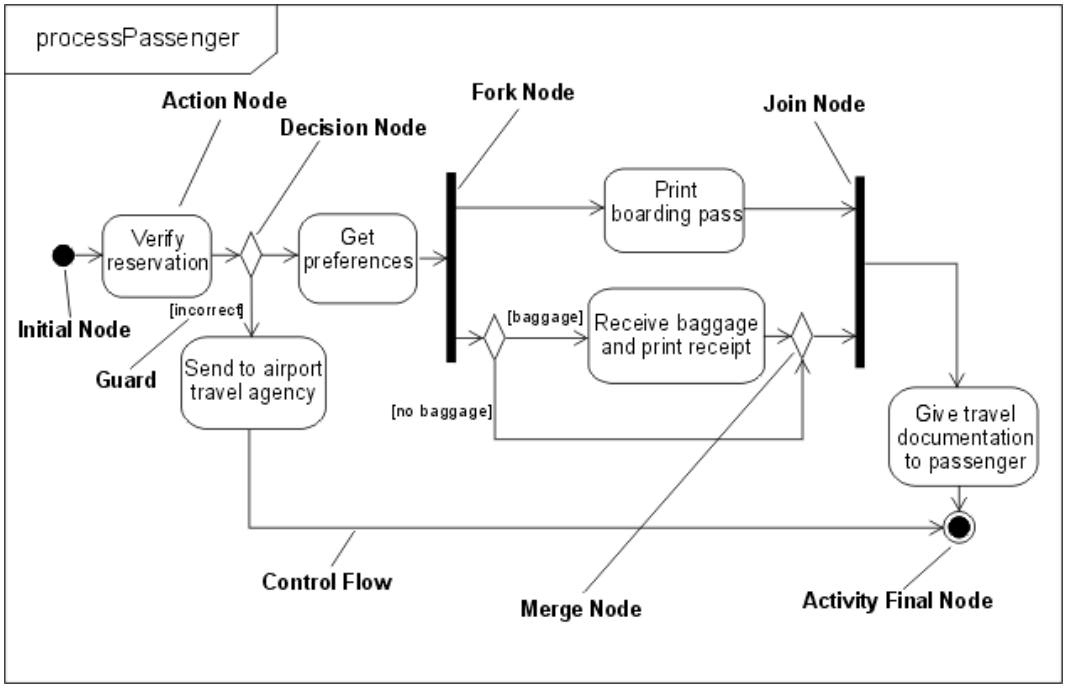
Kuva 2.1 *UML 1.5 -toimintakaavio. Kuvaa lentovarauksen vuota lentoyhtiön näkökulmasta. [23]*

2.2.2 UML 2.x

UML 2.0 on vuonna 2005 julkaistu huomattava UML-standardin uudistus [16]. Standardia on kehitetty edelleen, ja viimeisin UML 2.x -versio on vuonna 2015 standardoitu UML 2.5 [17]. Uudistuksen myötä mm. UML:n kuvaamien kaavioiden määrä kasvoi yhdekästä kolmeentoista.

UML 2.x uudisti toimintakaavioiden rakennetta lisäämällä niihin elementtejä Petriverkoista. Suoritus voi edetä UML 1.x:n tavoin tilakonemaisesti, mutta suoritusta voidaan ohjata myös objektivuon (object flow) avulla. Kukin toiminto voi tuottaa ulostuloonsa objektin. Vastaavasti toiminnon sisääntulot voivat vaatia objektin, jo-

ka vaaditaan ennen kuin suoritus on mahdollista. Esimerkiksi toiminto joka vaatii kolmessa sisääntulossaan objektin voidaan suorittaa vasta kun jokainen sisääntulo on saanut objektin jonkun toiminnon ulostulosta. Objektivuota voidaan yhdistää vapaasti tilakonemaisen etenemisen kanssa. Kuvassa 2.2 on esimerkki UML 2.0 -toimintakaaviosta.



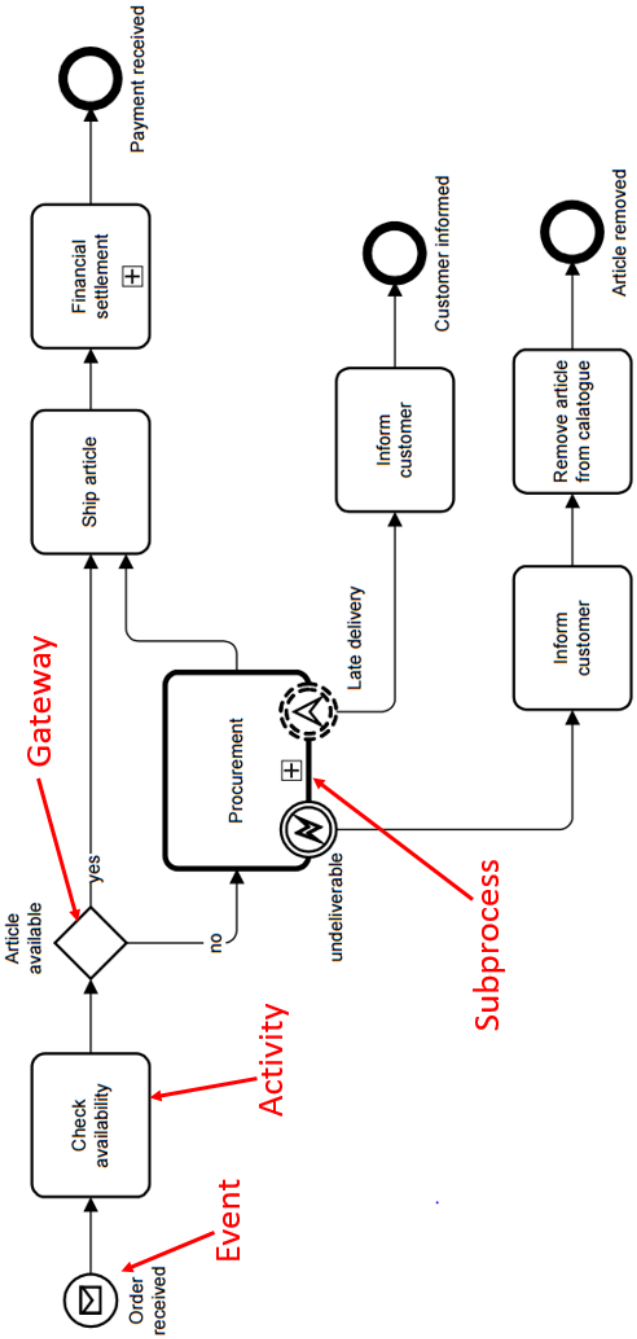
Kuva 2.2 *UML 2.0 -toimintakaavio. Kuvaa lentovarauksen vuota lentoyhtiön näkökulmasta [23]. Kaavio on semanttisesti identtinen kuvan 2.1 UML 1.5 -kaavion kanssa.*

2.2.3 BPMN 2.0

BPMN 2.0 on BPMN-standardin viimeisin versio, joka julkaistiin vuonna 2011 [19]. BPMN-standardin versio 1.0 julkaistiin vuonna 2004, jolloin siitä vastasi BPMI-konsortio (Business Process Management Initiative). Versiosta 1.1 alkaen standardin on julkaistu OMG-konsortion alaisuudessa.

Kuvassa 2.3 on esimerkki BPMN 2.0 -graafista. BPMN-mallissa prosessi etenee toimintojen (activity) välillä porttien (gateway) kautta. Porteilla voidaan esimerkiksi ohjata suoritusta ehdollisesti tai synkronoida useiden haarojen suoritus. Mallissa voi olla myös tapahtumia (event), joilla voidaan merkitä esimerkiksi ajan kulumista toimintojen välillä. BPMN-mallissa prosessi voidaan myös jakaa aliprosesseihin

(subprocess), joiden sisältö voidaan näyttää tai piilottaa tilanteen mukaan.

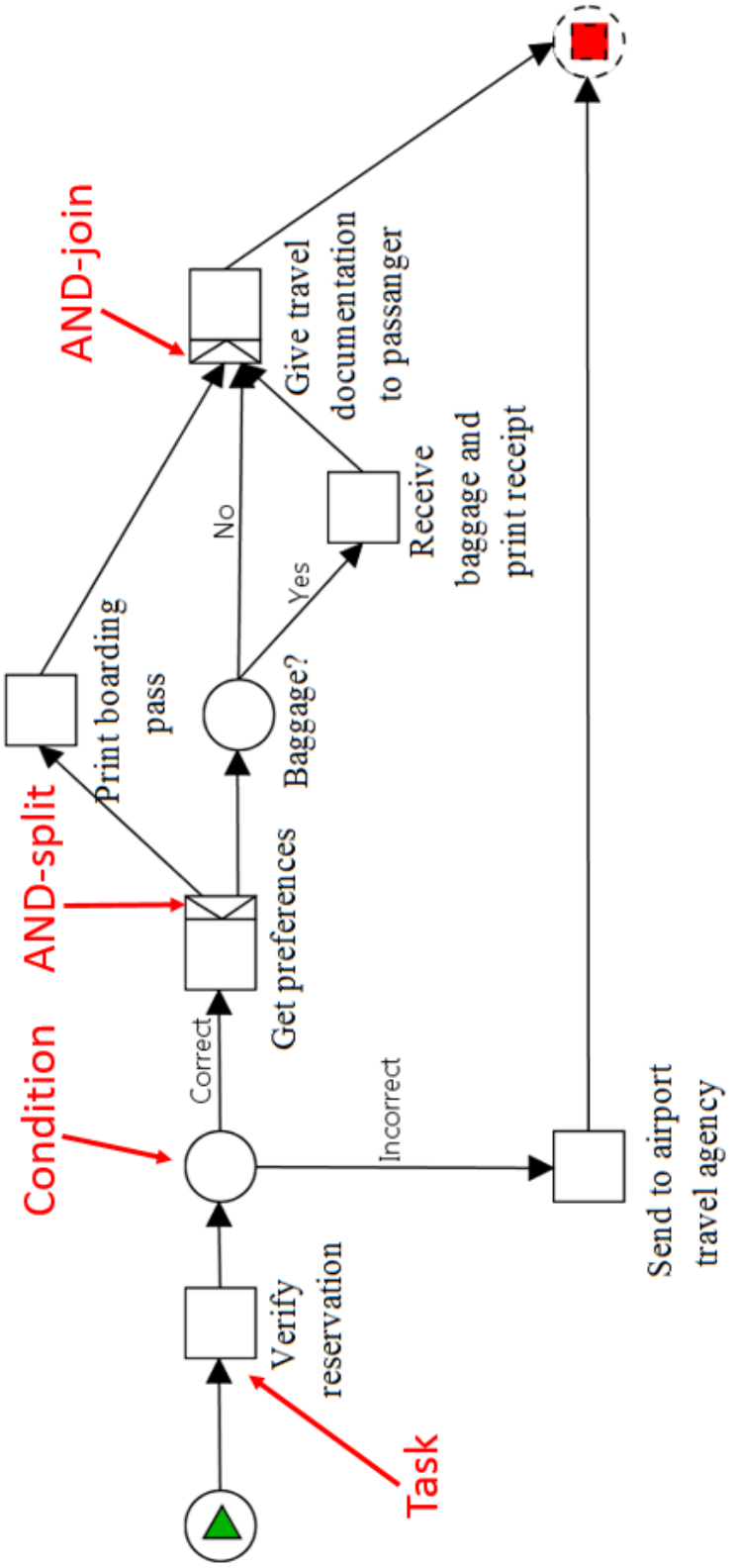


Kuva 2.3 *BPMN 2.0 -kaavio. Kuvaa tilausprosessia tilauksen toimittajan näkökulmasta [18, s. 5].*

2.2.4 YAWL

YAWL on avoimen lähdekoodin järjestelmä, jonka ensimmäinen versio julkaistiin vuonna 2003 [25]. Koska kyseessä on avoimen lähdekoodin järjestelmä, kuka tahansa voi osallista YAWL:n kehitykseen. Projektin organisoinnista vastaa YAWL Foundation [5]. YAWL:n viimeisin versio, 4.0, julkaistiin huhtikuussa 2016 [27].

YAWL:n lähtökohtana työvuon mallintamiseen on toiminut Petri-verkot. YAWL laajentaa Petri-verkkoja tiettyjen työvuokaavioissa esiintyvien tilanteiden mallintamisen helpottamiseksi. Näitä ovat tuki usealle saman tehtävän instanssille, kehittyneet synkronointimallit ja mitätöintimallit [25, s. 9]. YAWL-graafin koostuu tehtävistä (task) ja ehdoista (condition). Tehtävillä voi olla useita sisääntuloja ja ulostuloja, joka mahdollistaa suorituksen haarautumisen. Kuvassa 2.4 on esimerkki YAWL-graafista.



Kuva 2.4 *YAWL-kaavio. Kuvaa lentovarauksen vuota lentoyhtiön näkökulmasta. Kaavio on semanttisesti identtinen kuvan 2.1 UML 1.5 -kaavion kanssa.*

3. METAMALLI

Tässä luvussa esitellään Woke-sovelluksessa käytetty metamalli. Ensin esitetään lähtökohdat ja tavoitteet mallin suunnittelulle. Tämän jälkeen lopullinen malli esitellään kokonaisuudessaan, sekä perustellaan miksi lopullisen mallin ratkaisuihin on päädytty.

3.1 Tavoitteet

Tavoite Woke-sovelluksen metamallille oli luoda supppea mutta tehokas malli työvuon kuvaamiseen. Mallia ei suunniteltu alusta asti itse, vaan sen perustaksi valittiin hyväksi koettuja ominaisuuksia muista ratkaisuista. Taulukkoon 3.1 on koottu tärkeimmät ominaisuudet Woken metamallista ja sen perustana käytetyistä mallista.

Taulukko 3.1 *Yhteenveto eri työvuomallinnuskielien ominaisuuksista ja elementeistä.*

Ominaisuus	YAWL	BPMN2.0	UML2.x	Tilakaavio, UML1.x	Woke
Toimintamalli	Prosessi	Prosessi	Prosessi	Tila	Yhdistelmä
”Jokin on” (staattinen)	Ehto (ei vaadittu)	Tapahtuma	(Implisiittinen)	Tila / ulostulo	Tehtävä
”Jotain tapahtuu”	Tehtävä	Toiminta	Toiminta	(Implisiittinen)	Tehtävä
Relaatiot	Vuokaaret: Tehtävä -> Ehto Tehtävä -> Tehtävä Datakaaret	Vuokaaret: Tapahtuma -> Toiminta Tapahtuma -> Portti Toiminta -> Portti Viestikaaret	Vuokaaret: Toiminta -> Päätös Toiminta -> Toiminta Datakaaret	Siirtymä / ulostulo	Vuokaaret: Tehtävä -> Siirtymä Datakaaret
Liipaisu	Poletti, tehtävän jakautumis- ja yhtymis-ominaisuudet	Portti	Päätöselementti, erillinen jakautuminen ja yhtyminen	Siirtymä	Siirtymä (poletti, jakautuminen ja yhtyminen)

3.2 Lopullinen metamalli

Woken metamalli koostuu kahdesta peruselementistä, Tehtävät (Task, T) ja Siirtymät (Transition, R). Tehtäviä voisivat olla esimerkiksi mallinnus, simulointi ja synteesi. Siirtymät määräävät näiden tehtävien järjestyksen ja riippuvuudet niiden välillä. Kaksi peruselementtiä koettiin riittäväksi kuvaamaan erilaiset työvuot halutulla tarkkuudella, sillä varsinkin siirtymät ovat Woken metamallissa monipuolisempia kuin useassa muussa mallissa.

3.2.1 Tehtävät

Tehtävä kuvaa yhtä prosessin vaihetta. Woken tehtävät tarjoavat 12 valmiiksi määriteltyä ominaisuutta (attribute). Ominaisuudet mahdollistavat käyttäjälle näytettävän tiedon suodattamisen ja rajoittamisen ohjelman käyttöliittymässä.

Suurin osa ominaisuuksista on vapaamuotoisia tekstikenttiä, joten käyttäjä voi valita niiden tarkan tarkoituksen ja käyttötavan. Valmiit määritelmät yksinkertaistavat työkalun käyttöä, koska käyttäjän ei tarvitse itse määritellä mallia mallintaakseen työvuon. Tulevaisuudessa työkalu voisi antaa käyttäjälle mahdollisuuden luoda omia linjauksia, joka mahdollistaisi paremmat yhteenvedot ja yksinkertaisen tyyppitarkistuksen. Ominaisuudet, niiden tyypit ja kuvaukset on listattu taulukossa 3.2.

Taulukko 3.2 *Tehtävä-elementin sisältämät kentät.*

Kenttä	Nimi	Kuvaus
ID	Tunniste	Yksilöllinen tunniste, merkkijono. Esim. T.0, T.1, T.2
CO	Väri	Tehtävän taustaväri 12-bittisenä CSS-värikoodina muodossa #RGB
IC	Kuvake	Tehtävän kuvake PNG-muodossa
DE	Kuvaus	Tehtävän kuvaus, merkkijono
LI	Linkki	Lista ulkoisista tiedostoista merkkijonoina
RE	Resurssi	Tehtävän käyttämä resurssi, merkkijono
TL	Työkalu	Tehtävässä tarvittava työkalu, merkkijono
TP	Suunniteltu aika	Tehtävän käyttöön suunniteltu aika, esim. 1w:3d:5h
TS	Toteutunut aika	Tehtävään kulunut aika
TO	Poletti	Lista poleteista merkkijonoina
DI	Syötedata	Lista syötedatasta merkkijonoina (esim. tiedostonimi)
DO	Ulostulodata	Lista ulostulodatasta merkkijonoina

Jokaisella tehtävällä on pakollinen yksilöllinen tunniste (ID, identifier), joka koostuu etuliitteestä ja juoksevasta numerosta. Vakioarvo etuliittelle on T, joten uudet

tehtävät nimetään T.1, T.2, T.3 ja niin edelleen. Käyttäjä voi määritellä etuliitteen itse, mutta numerointi tehdään automaattisesti.

Tehtävällä on kaksi esitysasuun liittyvää ominaisuutta: esitysväri (CO, display color) ja kuvake (IC, icon). Käyttäjä voi määritellä kullekin tehtävälle yksilöllisen värin, tai väri voidaan määritellä etuliitekohtaisesti. Väri määritellään CSS-värikoodilla (Cascading Style Sheets). Etuliitekohtainen värin määrittely auttaa näkemään graafista nopeasti esimerkiksi tiettyyn prosessin vaiheeseen liittyvät tehtävät. Kullekin tehtävälle voidaan myös määritellä ikoni tunnistamisen helpottamiseksi. Ikoni annetaan PNG-muodossa (Portable Network Graphics).

Tehtävällä on kuusi perusominaisuutta. Kuvaus (DE, description) on vapaamuotoinen tekstikenttä johon voidaan tallettaa lisätietoa tehtävän suorituksesta. Linkki (LI, link) on lista viittauksia ulkoisiin tiedostoihin. Tiedosto voi olla esimerkiksi HTML- (Hypertext Markup Language) tai PDF (Portable Document Format) PDF-muotoinen dokumentti jossa tarjotaan lisäohjeita tehtävän suorittamiseen, tai jokin suoritettava koodi.

Resurssi- (RE, resource) ja työkalu-kentissä (TL, tool) listataan esimerkiksi tehtävästä vastaava henkilö, tehtävän suorittamiseen vaadittava tila tai tehtävässä vaadittava työkalu. Suunniteltu aika (TP, time planned) ja käytetty aika (TS, time spent) auttavat aikataulutuksessa ja analysoinnissa. Niiden arvot voidaan antaa joko aikayksiköinä tai geneerisinä yksiköttöminä arvoina.

Suorituksen liipaisevaa polettia (TO, flow trigger token) voidaan käyttää suorituksen ohjaamiseen siirtymissä. Siihen voidaan viitata siirtymien ehtolausekkeissa, jolloin siirtymän suorittaminen vaatii poletin valmistumisen tehtävässä.

Syötedataa (DI, data input) ja tulosdataa (DO, data output) voidaan käyttää datariippuvuuksien kuvaamiseen. Kukin tehtävä määrittelee datan jota se käyttää (syötedata), datan jota se tuottaa (tulosdata) ja datan jota se muokkaa (sekä syöte- että tulosdata). Datan muotoa ei ole erikseen määritelty, vaan se voi olla esimerkiksi globaali parametri, tiedostonimi tai tietokannan alkio.

3.2.2 Siirtymät

Metamallin toinen elementti, siirtymä, kuvaa yhtä tai useampaa lauseketta joiden mukaan tehtävien välillä siirrytään. Siirtymät ovat haaroja, esimerkiksi if-lauseita

joilla on ehto ja yksi tai useampia kohteita. Yksinkertaisimpana esimerkkinä lauseke `if T.1 then T.2` kuvaa suoraa siirtymää tehtävästä 1 tehtävään 2 tehtävän 1 valmistuttua.

Ehtolausekkeessa voi käyttää mitä tahansa tehtävän ominaisuutta tai binäärisiä muuttujia (tosi/epätosi tai on/ei ole). Siirtymä aiheuttaa aina vain suorituksen siirtymisen toisiin tehtäviin, se ei muokkaa ominaisuuksia eikä tuota tai muokkaa data.

Loogisten yhteyksien kuvaamiseen lausekkeissa on varattu kuusi sanaa: if, then, else, and, or ja xor. Avainsanat on koottu taulukkoon 3.3. Suoritus haarautuu kun lausekkeessa on useita kohteita. Tämä voi tapahtua vaihtoehtoisesti (`if t.X then t.Y else t.Z`) tai rinnakkaisesti (`if t.X then t.Y and t.Z`). Suoritus yhtyy kun ehtolausekkeessa on useita ehtoja, esimerkiksi `if t.X and t.Y then t.Z`.

Taulukko 3.3 *Siirtymän avainsanat ja niiden käyttötarkoitukset.*

Tarkoitus	Avainsanat	Esimerkki
Ehtolauseke	if, then, else	`if t.0 then t.1 else t.2`
Loogiset operaatiot	and, or, xor	`if t.0 or t.1 then t.2 and t.4`

3.2.3 Poletit

Woke-graafi koostuu siis kahdestatoista ominaisuudesta koostuvista tehtävistä ja siirtymistä jotka kuvaavat niiden suhteita. Malli on hyvin yksinkertainen kun sitä käytetään visualisointiin ja informaaliin analyysiin. Suoritettavan työvuokuvauksen on lisäksi kyettävä määrittelemään ja tunnistamaan tehtävien suorituksen valmistuminen. Kuten muissakin esitellyissä malleissa, Wokessa tähän käytetään poletteja (TO, token).

Poletit ovat binääriarvoisia. Aikakatkaisu vaatii poletin, ja kun Poletin arvo muuttuu todeksi siirtymä laukaistaan. Siirtymät eivät kuitenkaan voi esimerkiksi evaluoida muuttujia tai vertailla tehtävän ominaisuuden arvoa globaaliin kelloon, vaan tällaiset toiminnallisuudet pitää määritellä tehtävässä. Automaattinen poletin arvon laskenta määritellään linkki-ominaisuudella (link, LI). Yksinkertainen siirtymälauseke aikakatkaisulle olisi `if (t.X or t.X.5_min_timeout) then t.Y`. Tämä lauseke jatkaisi tehtävästä X tehtävään Y kun tehtävä X valmistuu tai sitä on ajettu 5 minuuttia.

4. GRAAFIN KÄSITTELY

Tässä luvussa esitellään graafin käsittelylle asetetut tavoitteet, sekä niiden pohjalta tehdyt suunnitteluratkaisut. Ensimmäisenä esitellään graafin rakennusprosessi. Tämän jälkeen esitellään graafin visualisointi ja selaaminen.

4.1 Graafin rakentaminen

Yhtenä Woken suunnittelun lähtökohtana oli graafin rakentamisen ja editoimisen sujuvuus pelkästään näppäimistöä käyttäen. Tämä on huomattava ero useisiin muihin vastaaviin työkaluihin verrattuna, jotka perustuvat pääosin hiiripohjaisiin graafisiin käyttöliittymiin. Koska näppäimistöä joka tapauksessa tarvitaan esimerkiksi tehtävien nimeämiseen ja ominaisuuksien muuttamiseen, myös korkean tason editointiominaisuuksien käyttäminen näppäimistöltä mahdollistaa nopean ja ergonomisen työskentelyn.

Käytännössä koko graafin rakennusprosessin haluttiin olevan mahdollista näppäimistöllä. Tämä kattaa uusien tehtävien ja siirtymien lisäämisen, tehtävien ja siirtymien poiston, tehtävien ja siirtymien sisällön muokkaamisen sekä siirtymisen näiden eri vaiheiden välillä.

Graafin rakentaminen perustuu pääosin siirtyminen muokkaamiseen. Siirtymän sisältöä voidaan muokata vapaamuotoisena tekstikenttänä, ja mahdolliset muutokset graafin rakenteeseeen toteutetaan siirtymän sisällön perusteella. Muokkaamisen avustamiseksi ja virheiden välttämiseksi käyttäjälle haluttiin tarjota siirtymää muokattaessa kontekstiriippuvaisia ehdotuksia esimerkiksi tehtävien nimistä sekä yksinkertainen syntaksin korostus tunnetuille sanoille.

Käyttäjä voi muokata siirtymän sisältöä vapaasti ja tehdyt muutokset astuvat voimaan vasta kun käyttäjä hyväksyy siirtymän sisällön. Tällöin siirtymän teksti parsitaan ja muutokset toteutetaan graafin rakenteeseen. Muutokset voivat olla yhteyk-

sien lisäämistä tai poistamista jo olemassaoleviin tehtäviin, tai sisältää viittauksia uusiin tehtäviin. Viittaus uuteen tehtävään toteutetaan viittaamalla tehtävään pelkällä etuliitteellä, jollloin uusi tehtävä lisätään graafiin automaattisesti. Myös siirtymän muokkauksen lopettaminen ilman muutoksien voimaantuloa haluttiin mahdollistaa.

Käyttäjälle haluttiin myös tarjota mahdollisuus palata graafin aikaisempaan tilaan muista sovelluksista tutulla peruutus-/uudelleensuoritus -konseptilla. Peruutus palauttaa graafin edellistä muutosta vastaavaan tilaan, ja uudelleensuoritus palauttaa graafin takaisin mahdollista peruutusta edeltäneeseen tilaan.

4.2 Graafin visualisointi

Myös graafin selaamisen sujuvuus oli yksi oleellisimmista tavoitteista työkalulle. Erityisesti haluttiin keskittyä selaamisen toimivuuteen kosketusnäyttöä käyttävillä laitteilla kuten taulutietokoneilla. Koska tavoitteena oli hyvä käyttökokemus graafin selaamiselle sovelluksen sisällä, graafin ulkonäölle esimerkiksi tulostettaessa ei asetettu suurta painoa.

4.2.1 Yhden sarakkeen näkymä

Graafin esittäminen yhdessä sarakkeessa oli ensimmäisiä päätöksiä sovelluksen suunnitteluvaiheessa. Yksiulotteinen vieritys todettiin selkeänä tapana selata isoa graafia varsinkin kosketusnäyttöä käyttävillä laitteilla. Vastaavaa tyyliä oli aiemmin käytetty Kactus2- sovelluksessa riippuvuuksien visualisointiin, jossa se oli todettu toimivaksi.

Yhden sarakkeen käyttö myös selkeyttää graafin lukemista. Tehtävien attribuutit on helppo löytää nopeasti, koska ne ovat jokaisessa tehtävässä samassa kohtaa. Myös graafien vertailun visualisointi on yksinkertaisempaa, kun graafin asettelu on vahvasti rajoitettu. Graafin rajoittaminen yhteen sarakkeeseen tekee graafin asettelusta vahvasti deterministisen, joten samaa graafia ei voi kuvata usealla eri tavalla.

4.2.2 Graafin piirtäminen

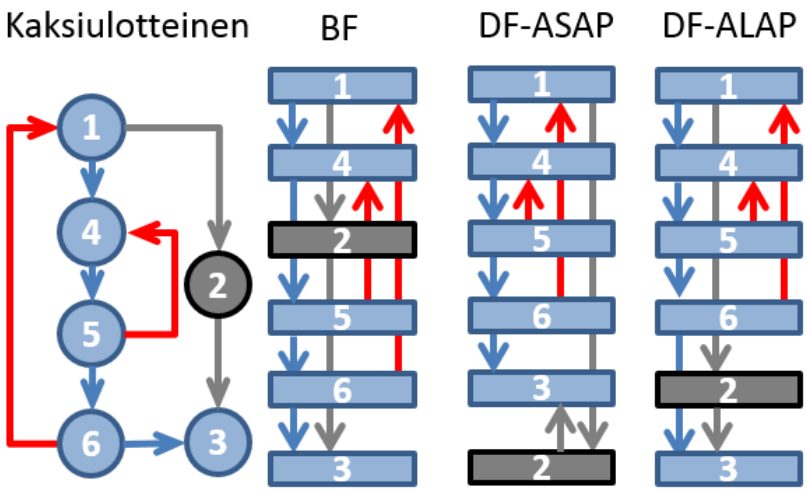
Varsinaisen visuaalisen graafin piirtämiseen metadatan perusteella on useita eri algoritmivaihtoehtoja. Työkalun suunnittelussa keskityttiin algoritmeista kahteen ka-

tegoriaan, syvyys ensin (BF, depth first) ja leveys ensin (DF, depth first). Kaikissa algoritmeissa graafin piirtäminen aloitetaan juurisolmusta.

Leveys ensin -algoritmissa ensin piirretään juurisolmu, ja sen jälkeen jokainen juurisolmun lapsista. Tämän jälkeen piirretään kunkin juurisolmun lapsen lapset ja niin edelleen, kunnes kaikki graafin solmut on piirretty. Tällä algoritmilla piirretyn graafin solmut on sijoitettu graafissa niin aikaisin kuin mahdollista (ASAP, as soon as possible).

Normaalissa syvyys ensin -algoritmissa juurisolmun jälkeen piirretään ensimmäinen sen lapsista. Seuraavaksi piirretään ensimmäinen tämän lapsisolmun lapsista, ja piirtämistä jatketaan vastaavasti kunnes saavutetaan solmu jolla ei ole lapsia, tai silmukan tapauksessa solmu jonka lapset on jo piirretty. Tämän jälkeen palataan piirretyissä solmuissa takaisin kunnes löydetään solmu jolla on piirtämättömiä lapsia, ja aloitetaan piirtoalgoritmi alusta.

Vaihtoehtoisessa syvyys ensin -algoritmissa DF-ALAP (as late as possible, niin myöhään kuin mahdollista) piirretään aluksi haara normaalilla syvyys ensin -algoritmilla. Tämän jälkeen rinnakkaiset haarat voidaan sijoittaa joko piirretyn haaran alapuolelle mikäli haarat eivät yhdy, tai jo piirretyn haaran keskelle mahdollisimman lähelle mahdollista yhtymäkohtaa.



Kuva 4.1 *Piirtoalgoritmien vertailu. Vertailu yksinkertaisesta graafista piirrettynä eri algoritmeilla.*

Kuvassa 4.1 on esitetty vertailu eri piirtoalgoritmien tuloksesta yksinkertaisen graafin piirrossa. Woke-työkalussa päädyttiin käyttämään normaalia syvyys ensin -algoritmia. Syvyys ensin ja leveys ensin -algoritmien kompleksisuus on käytännössä sama ja ne ovat helposti toteutettavissa. DF-ALAP -algoritmi on näitä monimutkaisempi, koska algoritmin on selvitettävä ja pidettävä kirjaa eri tehtävien liikkuvuudesta graafin sisällä. Syvyys ensin -algoritmin koettiin muutaman testigraafin perusteella rakentavan selkeämpiä graafeja kuin leveys ensin -algoritmi. Käyttäjä voi vaikuttaa graafin piirtojärjestykseen rakennusvaiheessa. Esimerkiksi siirtymän lausekkeet `if t.1 then t.2 and t.3` ja `if t.1 then t.3 and t.2` piirtävät siirtymän kaksi ulostulohaaraa eri järjestyksessä.

4.2.3 Näkyvyyden rajaaminen

Graafin yksinkertaistamiseksi sovellukseen haluttiin mahdollisuus rajoittaa graafissa olevan tiedon näkyvyyttä. Erityisesti graafin haarojen piilottamista pidettiin tärkeänä. Haarojen piilottaminen mahdollistaa tiedon rajaamisen esimerkiksi niin, että tietyssä roolissa olevalle henkilölle voidaan näyttää vain kyseisen henkilön rooliin liittyvät työtehtävät.

Myös tehtävien sisällön näkyvyyden rajoittaminen haluttiin mahdollistaa. Normaalisti graafissa näytetään kaikki tehtävän ominaisuudet, kuten kuvaus, resurssi ja työkalu. Käyttäjän tulisi kuitenkin olla mahdollista piilottaa näitä ominaisuuksia tarpeen mukaan.

Graafin selkeyttämiseksi myös siirtymiä tulisi olla mahdollista piilottaa. Esimerkiksi yksinkertaiset `if X then Y` -siirtymät voidaan piilottaa. Lisäksi käyttäjälle voidaan tarjota tila, jossa kaikkien siirtymien sisältö on piilotettu ja tehtävät on yhdistetty suoraan toisiinsa. Tämä voi selkeyttää graafin ulkoasua huomattavasti.

5. KÄYTTÖLIITTYMÄ

Tässä luvussa esitellään Woken lopullinen käyttöliittymä. Käyttöliittymä esitellään yksityiskohtaisesti esimerkkikuvien kanssa. Ensin käyttöliittymän päänäkymä esitellään yleisellä tasolla. Tämän jälkeen kukin käyttöliittymän elementti esitellään omassa alikohdassaan.

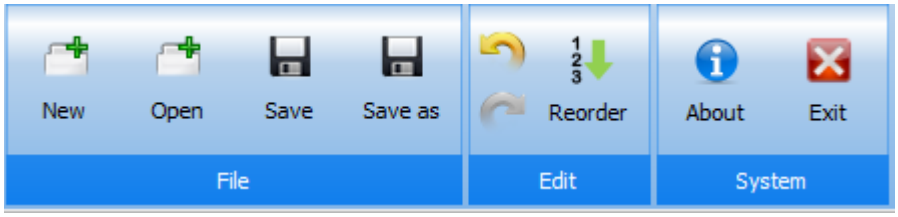
5.1 Käyttöliittymä yleisesti

Sovelluksen käyttöliittymän pohjana käytettiin Kactus2-sovellusta, jonka tyyli oli koettu toimivaksi. Käyttöliittymä koostu neljästä pääosasta: nauha (ribbon), tehtäväeditori, siirtymäeditori ja graafin visualisointi. Käyttöliittymän osat on merkitty kuvaan 5.1.

Kuva 5.1 *Sovelluksen käyttöliittymä. Käyttöliittymän osat on merkitty kuvaan punaisella värillä.*

5.2 Nauha

Nauha on esitetty kuvassa 5.2. Nauha tarjoaa käyttöliittymän sovelluksen korkean tason toiminnoille. Se koostuu kuvakerivistä, joka on ryhmitelty kolmeen osaan: tiedostoon liittyvät toiminnot (File), näkymään liittyvät toiminnot (View) ja järjestelmään liittyvät toiminnot (System). Mikäli jokin toiminto ei ole käytössä, esimerkiksi tallennus heti tallentamisen jälkeen, piirretään kyseistä toimintoa kuvaava ikoni harmaana.



Kuva 5.2 *Nauha. Uudelleensuoritustoiminto ei ole käytettävissä, joten sen kuvake on piirretty harmaana.*

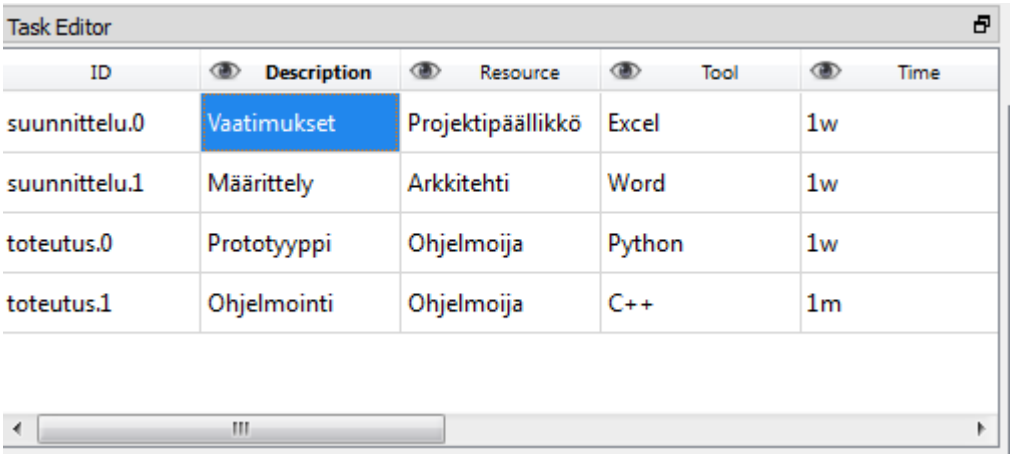
Tiedostoon liittyvät toiminnot mahdollistavat uuden graafin luomisen, graafin tallettamisen tiedostoksi ja graafin lataamisen tiedostosta. Järjestelmään liittyvät toiminnot tarjoavat tietoa sovelluksesta (About) ja painikkeen sovelluksen sulkemiselle (Exit). Mikäli graafia on muokattu sen viimeisimmän tallettamisen jälkeen, käyttäjältä kysytään sulkemisen yhteydessä halutaanko graafi tallettaa.

Näkymään liittyviä toimintoja käytetään graafin editointiin ja tarkasteluun. Käyttäjälle tarjotaan peruutus- (Undo) ja uudelleensuoritus-painikkeet (Redo) mahdollisten virheellisten syötteiden peruuttamiseksi tai muutosten tarkastelemiseksi. Lisäksi tarjotaan painikkeet graafin suurentamiseksi ja pienentämiseksi visualisointinäkymässä. Lopuksi tarjotaan uudelleenjärjestelypainike (Reorder), jolla graafi piirretään kokonaan uudestaan syvyys ensin -algoritmilla, samalla etuliitteellä alkavat tehtävät numeroidaan mahdollisesti uudelleen ja tehtävä- sekä siirtymäeditorien sisältö järjestellään graafin mukaiseksi.

5.3 Tehtäväeditori

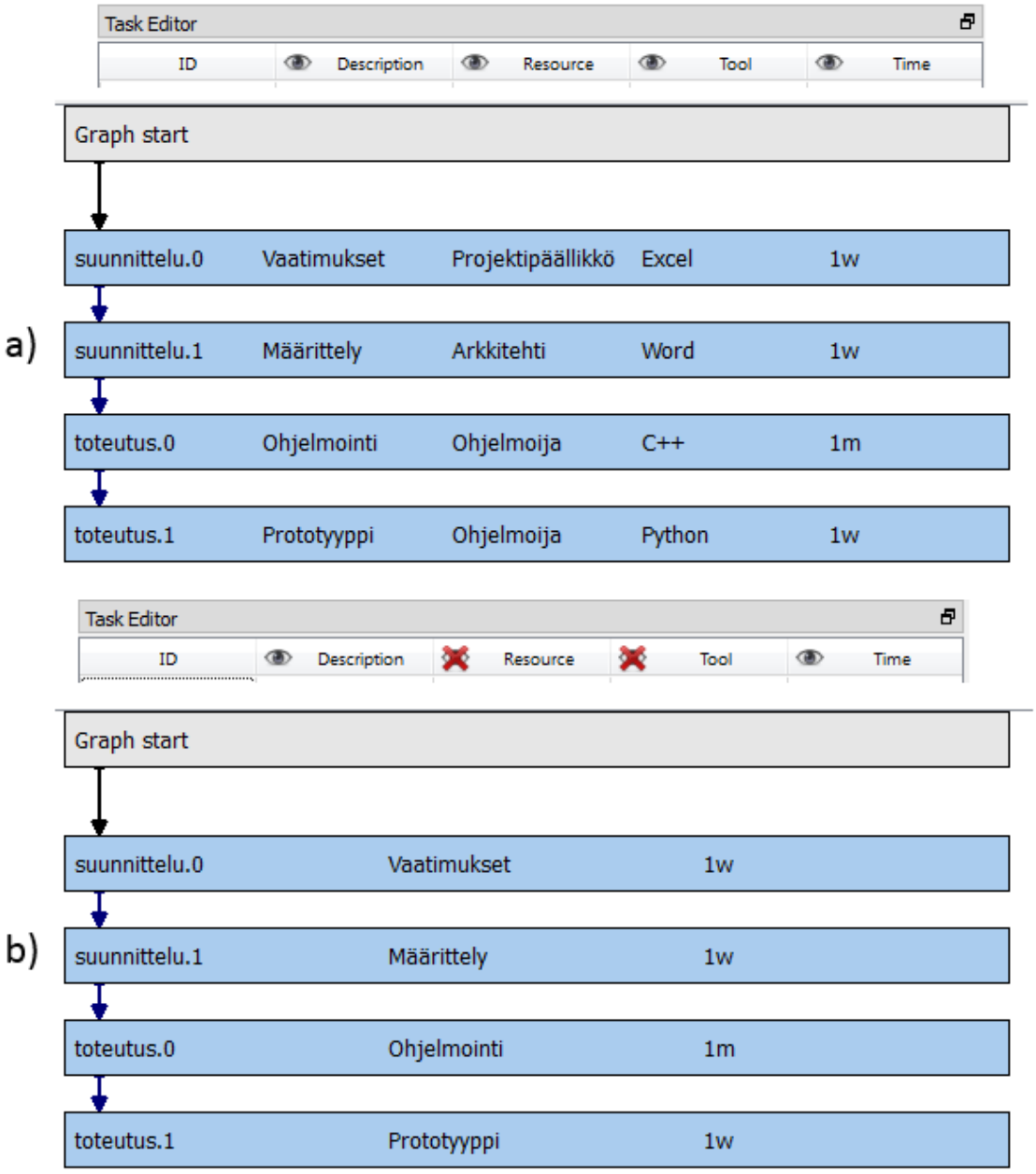
Tehtäväeditori on esitetty kuvassa 5.3. Se on tarkoitettu tehtävien sekä niiden ominaisuuksien tarkasteluun ja muokkaamiseen. Tehtäväeditori koostuu taulukkomuotoisesta listauksesta kaikista mallin tehtävistä. Taulukon rivi kuvaa yhtä tehtävää, ja

tehtävien ominaisuudet on jaettu taulukon sarakkeisiin. Sarakkeen otsikkoa hiirellä näpäyttämällä sen sisältö voidaan piilottaa graafissa kuvan 5.4 mukaisesti.



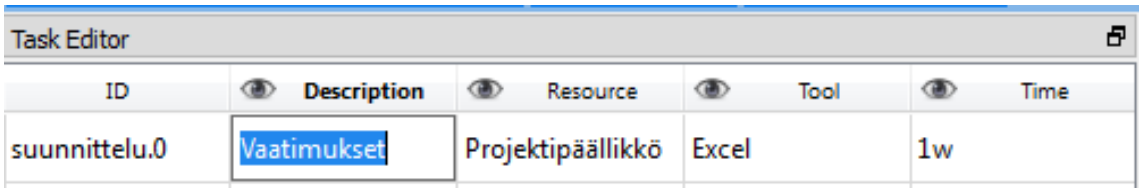
Kuva 5.3 *Tehtäväeditori perustilassa. Valittu kenttä on korostettu sinisellä taustavärillä. Osa sarakkeista ei mahdu editoriin, joten niiden näyttämiseksi on käytettävä alareunassa sijaitsevaa vierityspalkkia.*

Taulukkomuotoinen käyttöliittymä tarjoaa mahdollisuuden muokata tehtäviä nopeasti pelkän näppäimistön avulla. Käyttäjä voi liikkua taulukon kenttien välillä nuolinäppäinten avulla tai näpäyttämällä kenttää hiirellä. Valittu kenttä korostetaan sinisellä taustavärillä. Käyttäjä voi luoda taulukkoon uuden tehtävän painamalla Enter-näppäintä ja poistaa valitun tehtävän Delete-näppäimellä.



Kuva 5.4 *Ominaisuuksien näkyvyyden muuttaminen. Ylemmässä kuvassa a) kaikki tehtävän ominaisuudet ovat näkyvillä. Kuvassa b) resurssi- ja työkaluominaisuudet on piilotettu.*

Kentän muokkaaminen voidaan aloittaa joko suoraan kirjoittamalla kentän ollessa valittuna, kaksoisnäpäyttämällä kenttää hiirellä tai näppäimistön F2-näppäimellä. Tekstimuotoisten kenttien muokkaus tapahtuu taulukon sisällä kuvan 5.5 mukaisesti.

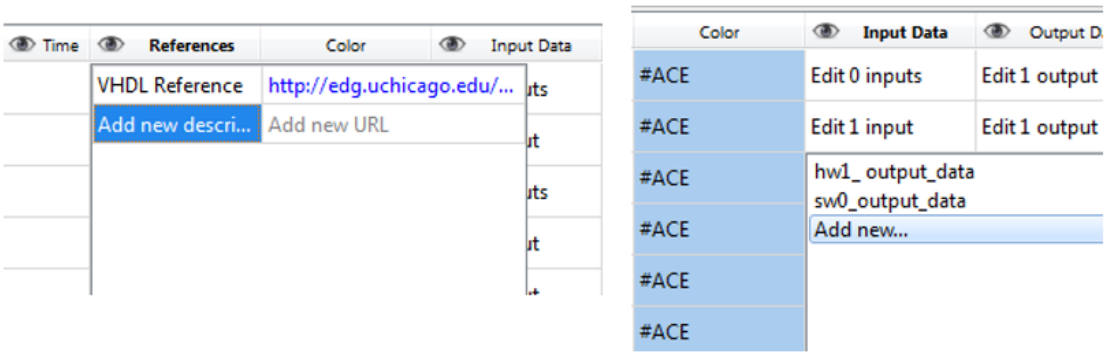


Kuva 5.5 *Tehtävän tekstikentän muokkkaminen. Kuvassa muokataan tehtävän suunnittelu.0 kuvauskenttää.*

Kun halutut muutokset on tehty, käyttäjä voi lopettaa kentän muokkaamisen Enter-näppäimellä jolloin kentän sisältö hyväksytään. Vastaavasti muokkaaminen voidaan lopettaa Esc-näppäimellä, jolloin tehdyt muutokset hylätään.

5.3.1 Listamuotoiset kentät

Listamuotoisten kenttien kohdalla editointitilaan siirtyminen avaa erityisen listaeditorin. Listamuotoisia kenttiä ovat linkki, syötedata ja ulostulodata. Esimerkit listaeditoreista on esitetty kuvassa 5.6. Linkkien muokkaamiseen käytettävässä editorissa on kaksi saraketta (kuvaus ja linkki), sisään- ja ulostulodatan muokkaamiseen käytettävässä editorissa yksi (nimi).



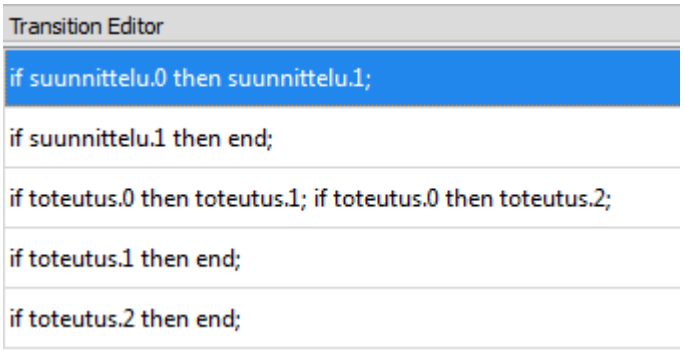
Kuva 5.6 *Listaeditorit. Kuvassa vasemmalla on linkkien muokkaamiseen käytettävä listaeditori. Oikealla on sisääntulodatan muokkamiseen käytettävä editori.*

Kun käyttäjä on avannut listaeditorin, liikkuminen nuolinäppäimillä rajoittuu kyseisen editorin sisälle. Käyttäjä voi aloittaa kentän muokkaamisen tehtäväeditorin

tavoin F2-näppäimellä tai suoraan kirjoittamalla. Muutokset hyväksytään Enter-näppäimellä. Kun listaeditorin sisältö on täytetty, tehdyt muutokset voidaan hyväksyä Tab-näppäimellä, jolloin listaeditori sulkeutuu. Esc-näppäimellä editori sulkeutuu ja tehdyt muutokset hylätään.

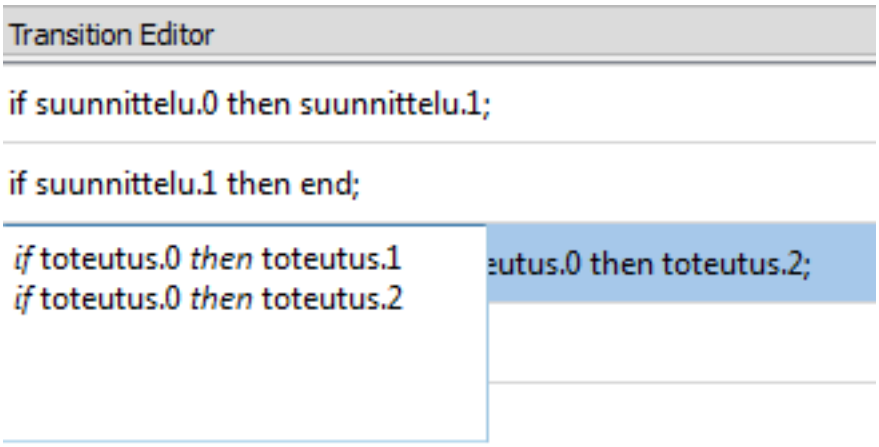
5.4 Siirtymäeditori

Siirtymäeditori (kuva 5.7) on tarkoitettu siirtymien muokkaamisen. Siirtymäeditori sisältää rivitetyn listan kaikista graafin siirtymistä, ja listan rivien välillä voidaan liikkua nuolinäppäimillä. Valittu siirtymä korostetaan sinisellä taustavärillä.



Kuva 5.7 *Siirtymäeditorin perustila. Valittu siirtymä on korostettu sinisellä taustavärillä.*

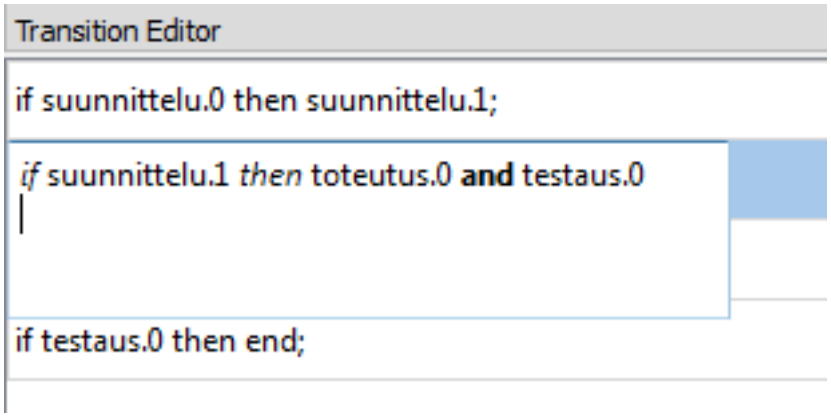
Tehtäväeditorin tavoin siirtymän muokkaus voidaan aloittaa kirjoittamalla näppäimistöllä, hiiren kakoisnäpäytyksellä tai F2-näppäimellä. Listasta voidaan poistaa valittu siirtymä Delete-näppäimellä. Vakiotilassa siirtymän koko sisältö esitetään yhdellä rivillä, jolloin lausekkeet on erotettu puolipisteellä.



Kuva 5.8 *Siirtymäeditorin muokkaustila. Muokattavassa siirtymässä on kaksi lauseketta, jotka on eritelty omille riveilleen. Huomaa kursiivilla korostetut avainsanat.*

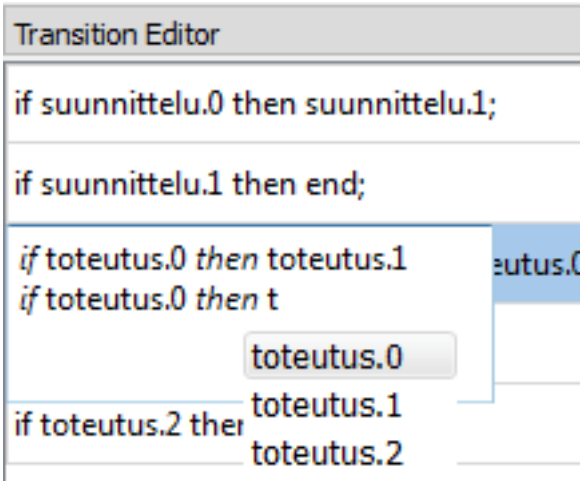
Muokkaustilassa editorin päälle avautuu erityinen muokkausikkuna, joka on esitetty kuvassa 5.8. Muokkausikkunan koko muuttuu automaattisesti tekstisisältöä muokattaessa niin että siirtymän sisältö mahtuu ikkunaan. Käyttäjä voi vapaasti muokata tekstiä muokkausikkunassa. Jokainen tekstikentän rivi kuvaa yhtä siirtymän lauseketta.

Käyttäjän kirjoittaessa tekstiä tunnetut avainsanat korostetaan tekstissä kuvan 5.9 mukaisesti. Vuota ohjaavat avainsanat `if`, `then` ja `else` korostetaan *kursiivilla*, kun taas loogiset operaattorit `or`, `and` ja `xor` **lihavoinnilla**.



Kuva 5.9 *Syntaksin korostus. Vuota ohjaavat avainsanat korostetaan kursiivilla, loogiset operaattorit lihavoinnilla.*

Lisäksi käyttäjälle esitetään ehdotuksia tehtävien nimistä kuvan 5.10 mukaisesti. Kun ehdotuslistaus on näkyvillä, käyttäjä voi valita halutun ehdotuksen nuolinäppäimillä ja vahvistaa valinnan Enter-näppäimellä.

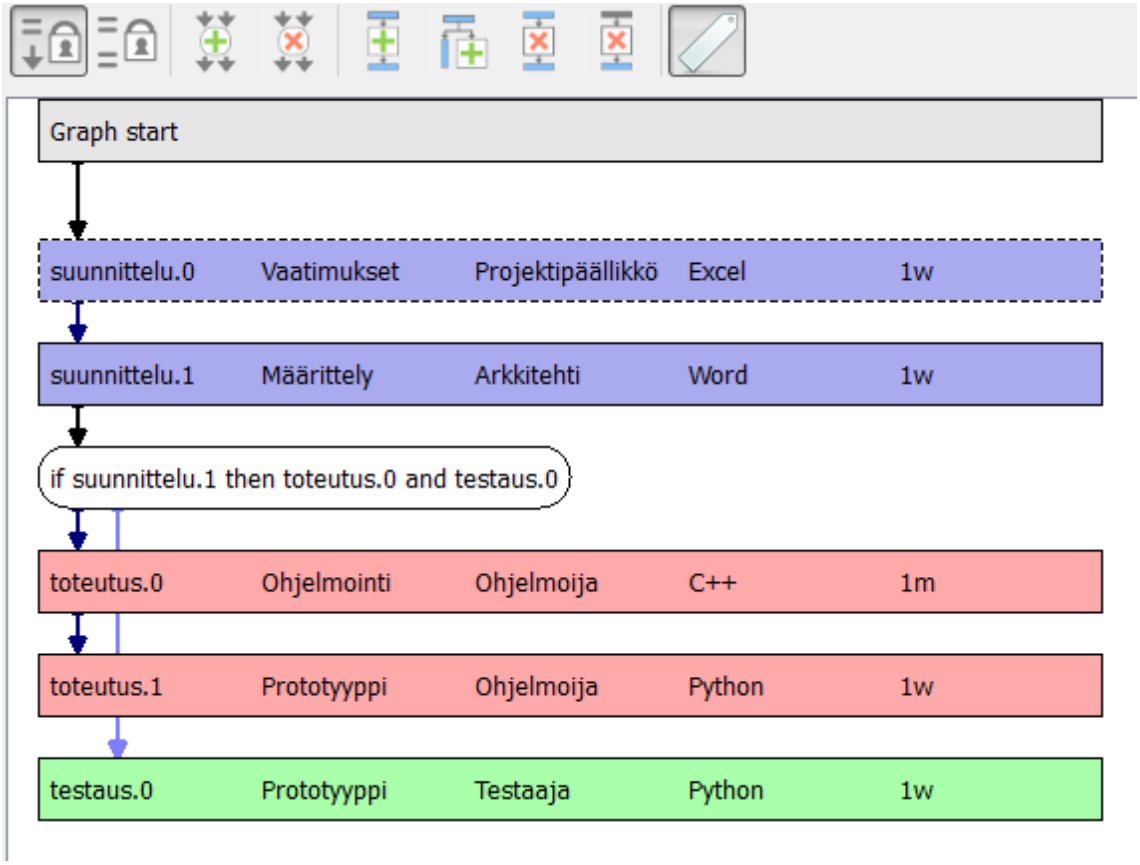


Kuva 5.10 *Tehtävänimien ehdotusvalikko. Käyttäjälle tarjotaan lista olemassaolevista tehtävistä muokkaamisen nopeuttamiseksi.*

Käyttäjä voi poistua muokkaustilasta Tab-näppäimellä, jolloin siirtymän sisältö vahvistetaan ja mahdolliset muutokset tallennetaan malliin. Mikäli käyttäjä haluaakin hylätä tehdyt muutokset, muokkaustilasta voi poistua Esc-näppäimellä.

5.5 Graafi

Graafi rakennetaan ja näytetään käyttäjälle ikkunan oikeassa reunassa sijaitsevassa visualisointinäkymässä. Graafin yläreunassa on aina graafin aloitusta ilmaiseva solmu (Graph start). Graafi piirretään tästä alkaen syvyys ensin -algoritmilla niin, että tehtävät joihin ei tulla yhdestäkään siirtymästä käsitellään aloitussolmun lapsina. Esimerkki graafi-ikkunasta on esitetty kuvassa 5.11.



Kuva 5.11 *Graafinäkymä. Esimerkki yksinkertaisesta graafista. Valittu tehtävä suunnittelu.0 on reunustettu katkoviivalla.*

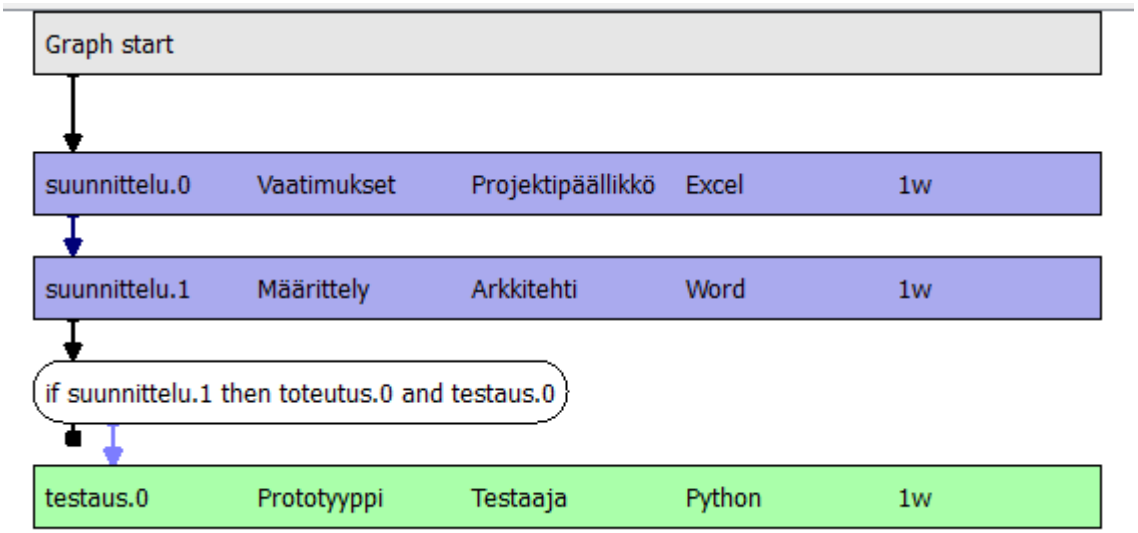
Graafin tehtävät piirretään teräväkulmaisina suorakulmioina, joiden taustaväri riippuu kyseisen tehtävän väriominaisuudesta. Siirtymät ovat suorakulmioita joiden kulmat ovat pyöristettyjä, ja niiden taustaväri on aina valkoinen. Tehtävät ja siirtymät yhdistetään mallin mukaisesti nuolilla. Kustakin tehtävästä osoittaa nuoli jokaiseen siirtymään jossa tehtävää käytetään siirtymän ehtona, ja siirtymistä piirretään nuoli jokaiseen siirtymän ulostulotehtävään.

Käyttäjä voi valita graafinäkymästä tehtävän tai siirtymän näpäyttämällä sitä hiirellä. Valittu siirtymä korostetaan piirtämällä sen ääriviivat katkoviivalla (kuva 5.11.

Graafin yksinkertaistamiseksi yksinkertaisin mahdollinen siirtymälauseke, `if X then Y`, piilotetaan aina kokonaan graafinäkymästä. Tällöin yhdistävä nuoli piirretään suoraan tehtävien välille. Kuvassa 5.11 näkyy sekä automaattisesti piilotettuja

siirtymiä että näkyviin jääneitä siirtymiä.

Käyttäjä voi manuaalisesti piilottaa graafin haaroja kaksoisnäpäyttämällä haaran aloittavaa nuolta hiirellä. Tällöin nuoli korvataan pienellä symbolilla, jota uudestaan kaksoisnäpäyttämällä haara piirretään taas graafiin. Kuvassa 5.12 on esimerkki piilotetusta haarasta.



Kuva 5.12 *Haaran piilotus. Siirtymästä tehtävään toteutus.0 osoittava haara on piilotettu.*

5.5.1 Graafinäkymän työkalupalkki

Graafinäkymän yläreunassa on erillinen ikoneista koostuva työkalupalkki (kuva 5.13). Tähän työkalupalkkiin on koottu graafin käsittelyyn käytettäviä toimintoja. Erillisessä työkalupalkissa toiminnot erottuvat selkeästi nauhaan sijoitetuista korkeamman tason toiminnoista. Taulukossa 5.1 on esitelty työkalupalkin painikkeet ja niiden toiminnallisuudet.

Kuva 5.13 *Graafinäkymän työkalupalkki. Graafinäkymän yläreunassa oleva työkalupalkki sisältää graafin käsittelyyn käytettäviä toimintoja.*

Taulukko 5.1 *Graafinäkymän työkalupalkin painikkeet ja niiden selitykset.*

Painike	Toiminto	Selitys
	Synkronoi editori ja graafi	Synkronoi selaamisen editorissa ja graafissa. Katso alikohta 5.6.
	Synkronoi editorit	Synkronoi selaamisen tehtävä- ja siirtymäeditoreissa. Katso alikohta 5.6.
	Lisää siirtymä	Lisää malliin uuden tyhjän siirtymän.
	Poista siirtymä	Poistaa valitun siirtymän mallista.
	Lisää tehtävä keskelle	Lisää malliin uuden tehtävän ketjumaisesti valitun tehtävän jälkeen.
	Lisää tehtävä haarana	Lisää malliin uuden tehtävän uutena haarana valittua tehtävää seuraavaan siirtymään.
	Poista tehtävä yhdistäen	Poistaa valitun tehtävän mallista, yhdistäen seuraavan tehtävän siirtymiin joissa poistettua tehtävää käytettiin.
	Poista tehtävä	Poistaa valitun tehtävän mallista. Tehtävästä jatkuva vuo yhdistetään graafin juureen uutena haarana.
	Piilota siirtymät	Piilottaa kaikki siirtymät graafista sisällöstä riippumatta.

5.6 Näkymien synkronointi

Käyttöliittymän osien välille on toteutettu synkronointiominaisuuksia, jotka auttavat käyttäjää löytämään saman tiedon eri näkymistä. Synkronointi voidaan aktvioida graafin yläpuolella olevan tehtäväpalkin ikoneilla.

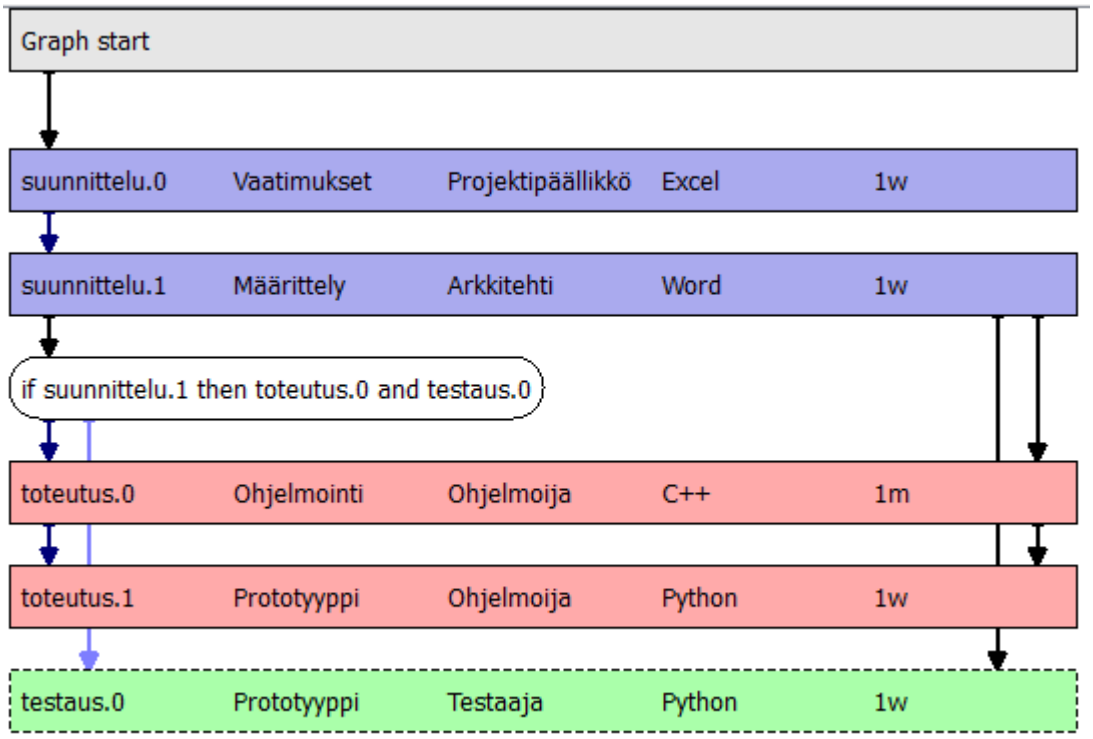
Editorin osat voidaan synkronoida keskenään, jolloin siirtyminen tehtävien välillä vaihtaa valittua siirtymää ja toisin päin. Kun käyttäjä valitsee tehtävän, siirtymäeditori siirtyy näyttämään ensimmäistä siirtymää jossa kyseistä tehtävää käytetään. Vastaavasti kun käyttäjä valitsee siirtymän, tehtäväeditori siirtyy näyttämään ensimmäistä siirtymässä mainittua tehtävää.

Myös graafin selaaminen voidaan synkronoida editoriin. Synkronoinnin ollessa päällä kun graafissa valitaan tehtävä tai siirtymä, vastaava rivi valitaan myös editorissa.

Synkronointi toimii myös toisin päin, eli kun editorissa valitaan tehtävä tai siirtymä, siirtyy graafinäkymä näyttämään valittua objektia.

5.7 Datavuo

Datavuo visualisoidaan graafin oikeassa reunassa. Nuolet kuvaavat datan siirtymistä tuottajatehtävien ja kuluttajatehtävien välillä. Jokaisesta tehtävästä, jolle on määritelty ulostulodataa, piirretään nuoli jokaiseen tehtävään joka käyttää kyseistä dataa. Datavuo ei vaikuta graafin piirtojärjestykseen. Tämän vuoksi yksittäisten datanuolien piilotusta ei koettu tarpeelliseksi, joten työvuosta poiketen datavuon osia ei voi piilottaa nuolia hiirellä näpäyttämällä. Kuvassa 5.14 esitetään yksinkertainen esimerkki datavuon visualisoinnista.



Kuva 5.14 *Datavuon visualisointi. Tehtävä suunnittelu.1 tuottaa dataa, jota sekä toteutus.0 että testaus.0 käyttävät. Lisäksi Toteutus.0 tuottaa dataa jota toteutus.1 käyttää.*

6. SOVELLUKSEN TOTEUTUS

Tässä luvussa käsitellään sovelluksen varsinainen toteutus. Ensin esitellään käytetty kehitysympäristö. Tämän jälkeen esitellään sovelluksen rakenne korkealla tasolla. Seuraavaksi esitellään toteutuksen yksityiskohdat. Lopuksi selostetaan toteutustyön eteneminen ja esitetään statistiikkaa toteutuksesta.

6.1 Kehitysympäristö

Sovelluksen toteutuskieleksi valittiin C++ [8]. Valintaperusteena oli aikasempi kokemus kielestä Kactus2-projektissa [24]. Projektin puitteissa ei koettu tarpeelliseksi selvittää muiden kielten käyttöä sovelluksen toteutukseen. Sovellus toteutettiin olio-ohjelmoinnin periaatteita noudattaen.

Woke rakennettiin Qt-ohjelmistokehyksen [21] päälle. Perusteena Qt:n valitsemiselle oli aikaisempi kokemus ympäristön käytöstä Kactus2-projektissa. Qt tarjoaa alustariippumattoman rajapinnan graafisen käyttöliittymän toteuttaville sovelluksille. Tämä mahdollistaa saman sovelluskoodin käyttämisen esimerkiksi Windows-, Linux- ja Mac OS X -käyttöjärjestelmillä. Graafisten käyttöliittymäkomponenttien lisäksi Qt tarjoaa esimerkiksi oleellisimmat tietorakenteet. Qt-kehyksestä käytettiin kehitysaikaan viimeisintä versiota, Qt 5.2.0 [22].

Sovelluksen kehitys tapahtui Microsoft Visual Studio 2010 -ohjelmointiympäristössä. Visual Studio tarjoaa C++-kääntäjän Windows-käyttöjärjestelmälle sekä esimerkiksi syntaksin korostuksen ja muita ohjelmointia helpottavia toimintoa. Myös Visual Studio valittiin käyttöön Kactus2-projektin kokemuksen perusteella. Ohjelmakoodissa ei käytetä mitään alustakohtaisia rajapintoja. Sovellusta ei ole käännetty muille käyttöjärjestelmille kuin Windowsille, mutta sen pitäisi olla mahdollista kunhan alustalle on saatavilla C++-kääntäjä sekä Qt-kirjasto.

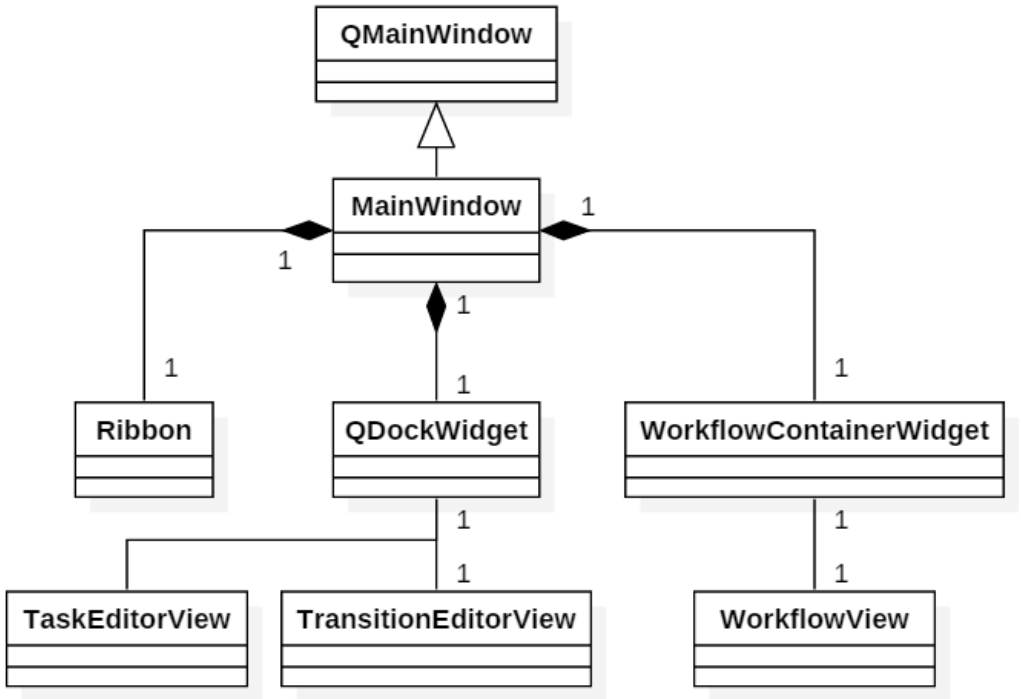
Versionhallintana projektissa käytettiin Apache Subversionia (SVN) [6]. Versionhal-

linta oli ylläpidettynä ilmaisessa SourceForge-palvelussa [2]. Versionhallinnan lisäksi SourceForge tarjoaa myös muita kehitystyössä avustavia työkaluja. Näitä ovat esimerkiksi virheiden seuranta, sovelluksen jakelu sekä dokumentointi.

6.2 Sovelluksen rakenne

Sovellus on jaettu korkealla tasolla käyttöliittymän (kuva 5.1) mukaisesti neljään osaan: nauha, tehtävien käsittely, siirtymien käsittely ja graafi. Ohjelman pääikkuna vastaa näiden osien sijoittelusta ja alustamisesta ohjelman käynnistyksessä.

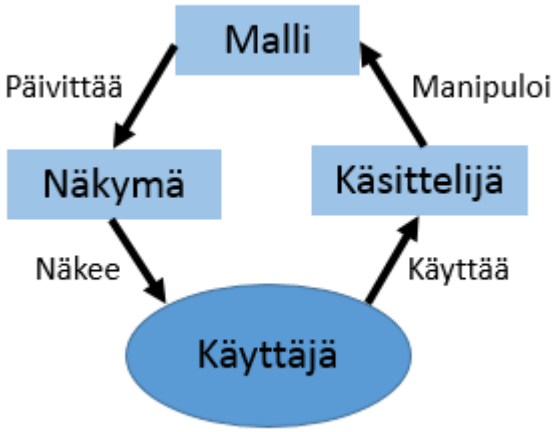
Qt-sovellukset rakentuvat käyttöliittymän sijoittelun määrittävistä widget-komponenteista ja niihin liitetyistä näkymistä. Woke-sovelluksen pääikkuna sisältää kolme widget-komponenttia, joihin tehtäväeditori, siirtymäeditori ja graafi on yhdistetty. Kuvassa 6.1 on esitetty luokkakaavio sovelluksen päänäkymään liittyvistä komponenteista ja niiden relaatioista. Seuraavissa alikohdissa esitellään tarkemmin editorien ja graafinäkymän sisäiset toteutukset.



Kuva 6.1 *Päänäkymän luokkakaavio. Esittää sovelluksen päänäkymään käytettävien luokkien relaatioita. Q-alkuiset luokat ovat Qt:n tarjoamia komponentteja, joista sovelluksen käyttämät luokat on periytetty.*

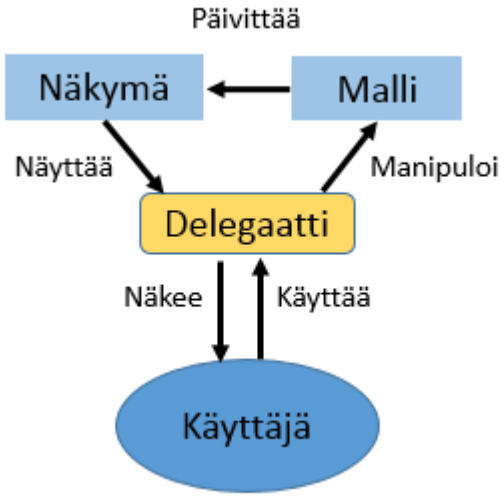
6.2.1 Editorit

Qt-sovelluskehyksen käyttöliittymäkomponentit on suunniteltu käytettäväksi MV-arkkitehtuurin (model/view, malli/näkymä) mukaisesti [20]. MV-arkkitehtuuri on yksinkertaistus MVC-arkkitehtuurista (model-view-controller, malli-näkymä-käsittelijä) [13]. MVC-arkkitehtuuri jakaa sovelluksen kolmeen osaan: malli, näkymä ja käsittelijä. Kuvassa 6.2 on esitetty MVC-arkkitehtuurin toimintaperiaate. Malli toteuttaa sovelluksen sisäisen tietorakenteen. Näkymä toteuttaa käyttöliittymän rakenteen ja näyttää tietorakenteen sisällön käyttäjälle. Käsittelijä ottaa vastaan käyttäjän komennot ja välittää nämä mallille, jolloin malli päivittyy käyttäjän komentojen mukaisesti.



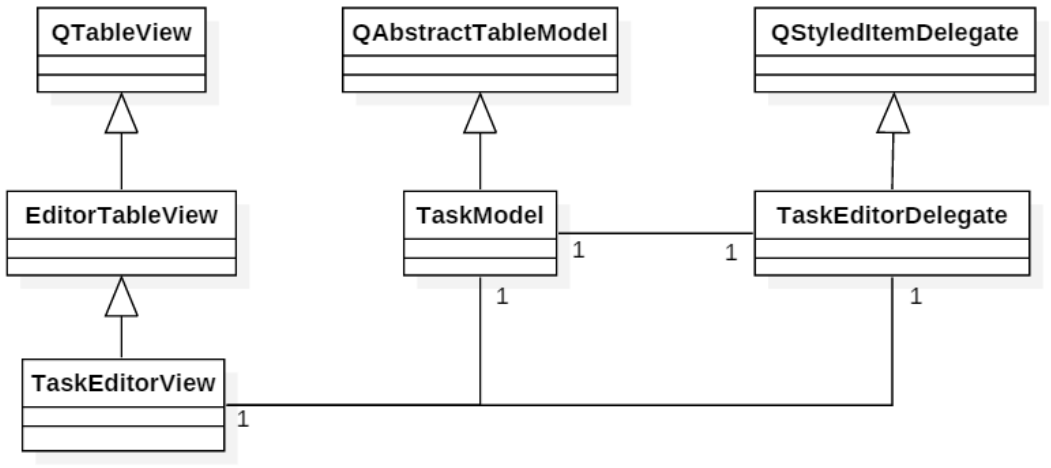
Kuva 6.2 *MVC-arkkitehtuuri. Kuvaa MVC-arkkitehtuurin eri osien sekä käyttäjän relaatioita.*

MV-arkkitehtuurissa varsinainen käsittelijäkomponentti poistuu. Sen tilalla arkkitehtuurissa on avustava lisäkomponentti delegaatti (delegate). MVC-arkkitehtuurissa komponentit ovat keskenään samanarvoisia, delegaatti sen sijaan on näkymän hallinnoima. Delegaatti sisältää osia sekä MVC:n käsittelijästä että näkymästä. Delegaatin tehtävänä on määrittää yksittäisen tietoalkion ulkoasu ja välittää käyttäjän alkioon tekemät muutokset mallille. MV-arkkitehtuurin toimintaperiaate on esitetty kuvassa 6.3.



Kuva 6.3 *MV-arkkitehtuuri. Kuvaa MV-arkkitehtuurin eri osien sekä käyttäjän relaatioita.*

Sovelluksessa on kaksi merkittävää malli/näkymä -paria. Tehtäviä kuvaa luokka `TaskModel` ja tämän sisältämä data näytetään käyttäjälle tehtäväeditorin toteuttavassa `TaskEditorView`-luokassa. Vastaavasti luokka `TransitionModel` sisältää työvuomallin siirtymät ja luokka `TransitionEditorView` toteuttaa siirtymäeditorin. Delegaatteina näille pareille toimivat luokat `TaskEditorDelegate` ja `TransitionEditorDelegate`. Kuvassa 6.4 on esitetty tehtävien käsittelyyn käytettävien komponenttien luokkakaavio. Siirtymien käsittelyssä rakenne on täysin vastaava.

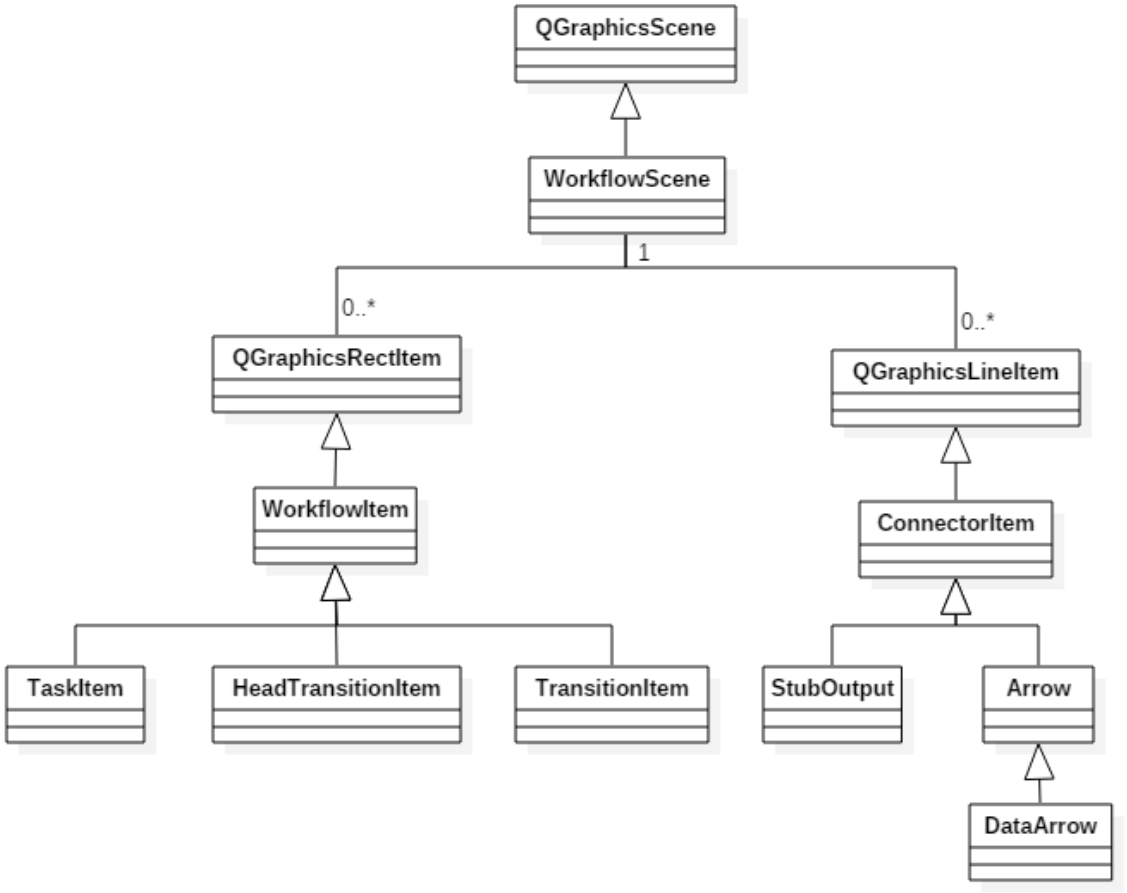


Kuva 6.4 *Tehtäväeditorin luokkakaavio. Esittää tehtäväeditorin toteuttamiseen käytettävien luokkien relaatioita. Q-alkuiset luokat ovat Qt:n tarjoamia komponentteja, joista sovelluksen käyttämät luokat on periytetty.*

6.2.2 Graafinäkymä

Visuaalisen graafin piirtämiseen ei käytetä MV-arkkitehtuuria. Graafi piirretään piirtoalgoritmin mukaisesti kuvana Qt:n grafiikkakirjastojen avulla. Piirtämiseen käytetään Qt:n sisältämiä yksinkertaisia graafisia elementtejä, kuten suorakulmioita ja viivoja, joiden tarkempi ulkonäkö on määritelty omassa luokassaan jokaiselle elementtityypille. Grafiikkakirjastot mahdollistavat suoraan myös graafin tallentamisen PNG-muotoisena kuvana.

Kuvassa 6.5 on esitetty luokkakaaviona graafinäkymän piirtämiseen käytettävien luokkien relaatiot. `QGraphicsItem`-tyyppisistä luokista perityt luokat, kuten `TaskItem`, `TransitionItem` ja `Arrow` kuvaavat graafin elementtejä. Näitä elementtejä lisätään `WorkflowScene`-luokan olioon. `WorkflowScene`-luokan yliluokka `QGraphicsScene` mahdollistaa graafisten elementtien vapaan sijoittelun 2D-avaruudessa ja toteuttaa graafin piirtämisen näytölle sijoitteun mukaisesti `WorkflowScene`-luokka toteuttaa sovelluskohtaiset lisäominaisuudet, kuten graafin leveyden muuttamisen ja tehtävien ominaisuuksien piilottamisen.



Kuva 6.5 *Graafinäkymän luokkakaavio. Esittää graafinäkymän piirtämiseen käytettävien luokkien relaatioita. Q-alkuiset luokat ovat Qt:n tarjoamia komponentteja, joista sovelluksen käyttämät luokat on periytetty.*

6.3 Työvuomallin tallentaminen

Työvuomalli tallennetaan levylle XML-tiedostona (Extensible Markup Language) [26]. XML-formaatti tarjoaa yksinkertaisen tekstipohjaisen rakenteen jota voidaan tarvittaessa muokata käsin tai ulkoisella sovelluksella. Näin esimerkiksi tallennetun työvuon integroiminen muihin sovelluksiin on suoraviivaista. XML-formaatista käytetään hyväksi hyvin pelkistettyä osajoukkoa. Kaikki tietoalkiot on tallennettu erillisiin elementteihin attribuuttien sijaan.

XML-tiedoston ylimpänä tasona toimii XML-formaatin vaatima juurielementti `rootElement`. Juurielementti sisältää kaksi lapsielementtiä, `tasks` ja `transitions`.

`Tasks`-elementti sisältää kaikki mallin tehtävät `task`-tyyppisinä elementteinä. `Task`-elementti sisältää erillisinä lapsielementteinä kaikki tehtävän sisältämät kentät. Alla olevassa listauksessa on esitetty esimerkki `task`-tyyppisestä elementistä.

```
<task>
 <name>toteutus</name>
 <id>0</id>
 <description>Ohjelmointi</description>
 <resource>Ohjelmoija</resource>
 <tool>C++</tool>
 <time>1m</time>
 <references/>
 <color>##FAA</color>
 <outputData>
  <dataName>Sovellus</dataName>
 </outputData>
 <inputData>
  <dataName>Spesifikaatio</dataName>
 </inputData>
</task>
```

`Transitions`-elementti sisältää mallin siirtymät `transition`-tyyppisinä elementteinä. `Transition`-elementti sisältää lapsielementit `content` ja `activeOutputs`. `Content`-elementti sisältää siirtymän sisällön lausekeittain talletettuna (`line`), `activeOutputs`-elementti graafinäkymässä aktiivisena olevat siirtymän ulostulohaarat (`output`). Alla olevassa listauksessa on esitetty esimerkki `transition`-elementistä.

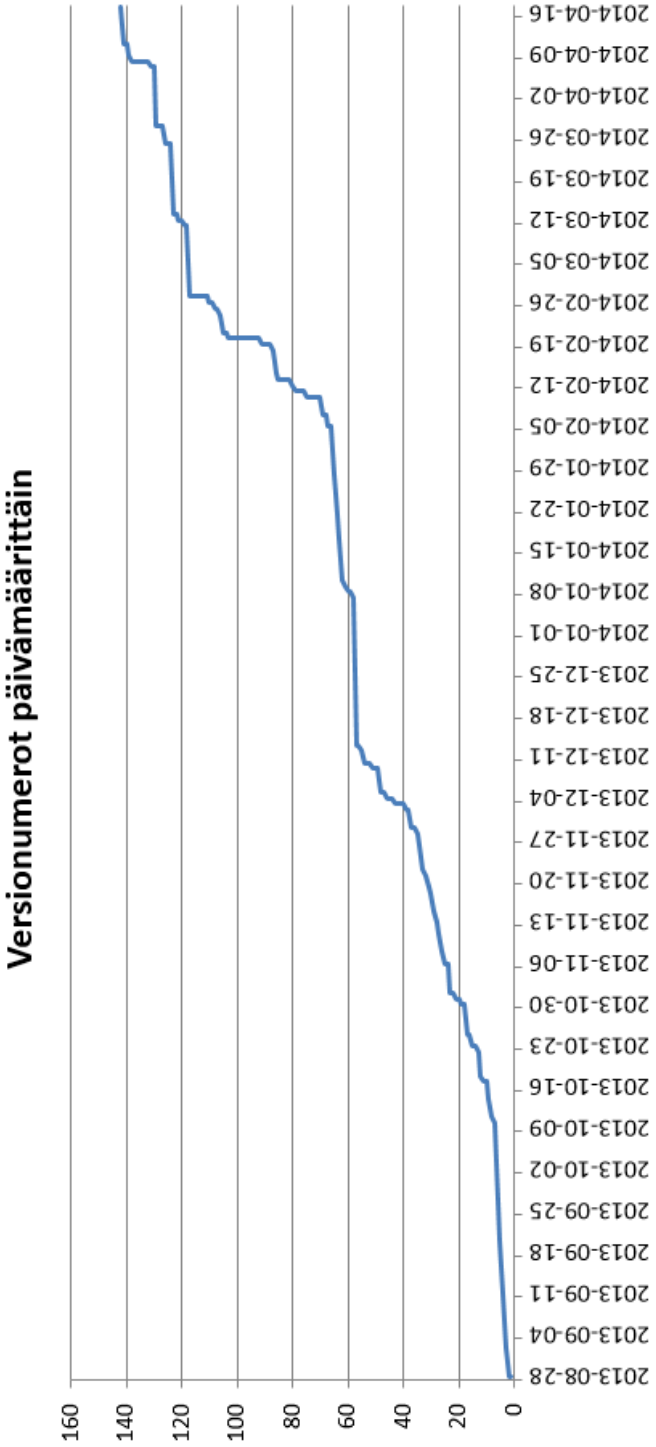
```
<transition>
 <content>
  <line>if suunnittelu.1 then toteutus.0 and testaus.0</line>
  <line></line>
 </content>
 <activeOutputs>
  <output>testaus.0</output>
  <output>toteutus.0</output>
 </activeOutputs>
</transition>
```

Liitteessä 1 on esimerkki kokonaisesta XML-tiedostosta. Tiedoston tuottama graafi on esitetty kuvassa 5.11.

6.4 Ohjelmointityö

Projektin ohjelmointityö sijottui pääosin puolen vuoden ajalle välillä syyskuu 2013 – helmikuu 2014. Sovelluksen versio 1.0.0 julkaistiin 27. helmikuuta 2014 [4]. Kehitystä jatkettiin vielä saman vuoden maaliskuussa ja huhtikuussa. Versionhallinnan viimeisin versio on päivämäärällä 17. huhtikuuta 2014 [3].

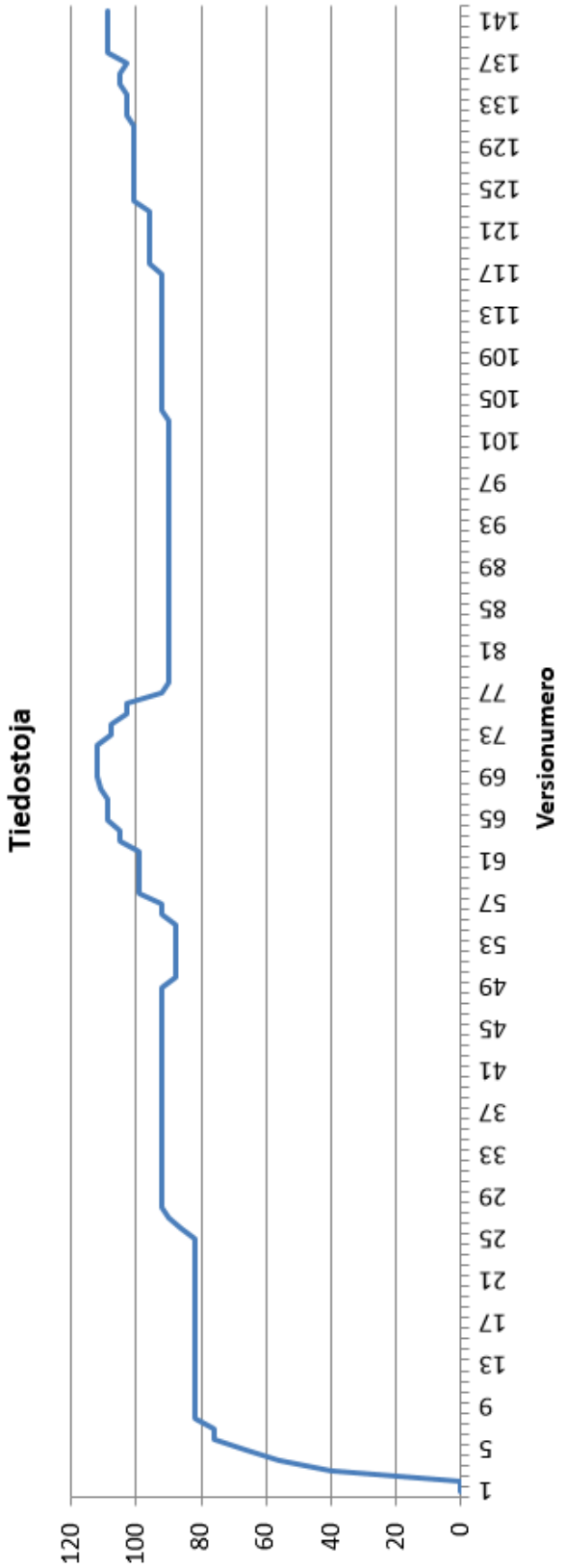
Kuvaan 6.6 on koottu versionumeroiden eteneminen projektin aikana. Versioiden etenemisessä on nähtävillä piikit joulukuussa 2013 ja helmikuussa 2014. Joulukuussa ohjelmointi eteni nopeasti sovelluksen vaatimusten ja arkkitehtuurin tarkennuttua. Helmikuussa kehitys keskittyi havaittujen ongelmien ja pienten käytettävyysparannusten tekoon. Tämä johti useisiin pieniä muutoksia sisältäviin versioihin.



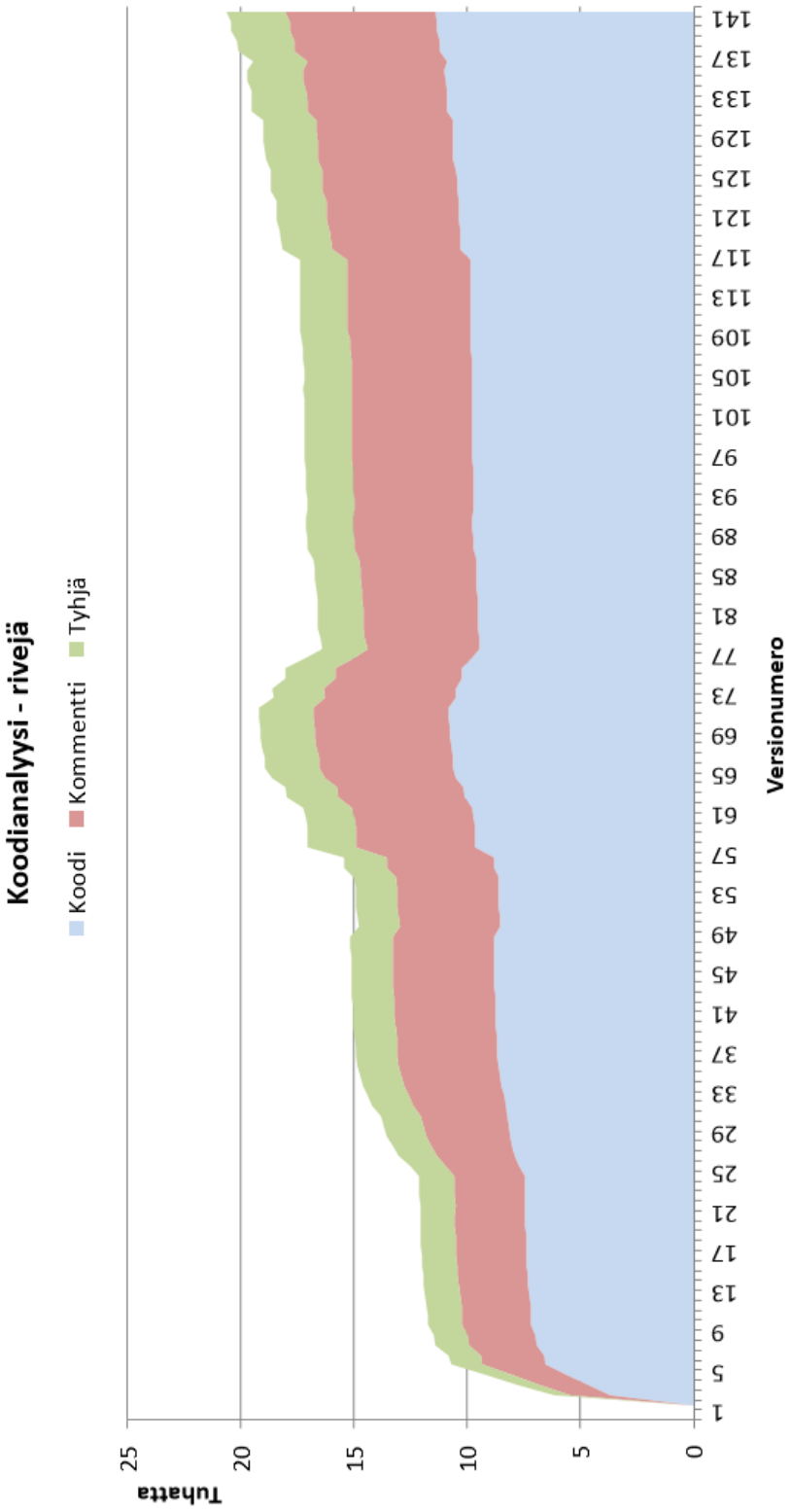
Kuva 6.6 *Versionumerot päivämäärittäin. Esittää versionhallinnan versionumeroiden etenemistä projektin aikana.*

Kuvassa 6.7 on esitetty sovelluksen tiedostomäärän kehitys versionumeroiden mukana. Kuvasta nähdään, että tiedostomäärä pysyy alun kasvun jälkeen tasaisena koko projektin ajan. Koska jokainen luokka on omassa tiedostossaan, pysyi luokkien määrä siis hyvin lähellä alkuperäistä suunnitelmaa. Isommat muutokset johtuvat lähinnä projektin siivoamisesta, esimerkiksi tarpeettomien projektin pohjaksi kopioitujen tiedostojen poistosta.

Kuvaan 6.8 on koottu projektin rivimäärän kehitys. Rivimäärän kehitys vastaa melko tarkasti tiedostomäärän kehitystä. Yksittäisten tiedostojen rivimäärä pysyi siis koko projektin ajan tasaisena. Melko suurta kommenttirivien ja tyhjien rivien osuutta voidaan pitää yhtenä hyvän ohjelmointityylin mittarina.



Kuva 6.7 *Tiedostomäärän kehitys. Kuvaa projektin tiedostomäärän kehitystä projektin aikana.*



Kuva 6.8 *Rivimäärän kehitys. Kuvaa projektin rivimäärän kehitystä projektin aikana. Varsinaisen ohjelmakoodin lisäksi listattuna on kommenttien ja tyhjien rivien määrä.*

7. ARVIOINTI JA VERTAILU

Tässä luvussa kehitettyä sovellusta arvioidaan sille asetettuihin vaatimuksiin ja tavoitteisiin. Lisäksi sovellusta verrataan olemassaoleviin graafinpiirtosovelluksiin esimerkiksi käytettävyyden ja graafin ulkoasun osalta. Lopuksi pohditaan sovelluksen jatkokehitysmahdollisuuksia.

7.1 Sovelluksen arviointi

Sovellusta arvioidaan sille työn alussa asetettuihin tavoitteisiin. Arvioitavana on sekä metamalli (alikohta 3.1) että graafinen sovellus (alikohdat 4.1 ja 4.2). Arviointia ei tehdä formaalisti, vaan pohdintatyylisesti.

Lopullinen metamalli on asetettujen tavoitteiden mukaisesti suppea, mutta silti ilmaisukykyinen. Merkittävä mallin suppeana pitävä ominaisuus on peruselementtien rajaaminen pelkästään tehtäviin ja siirtymiin. Elementtityyppien määrä ei kuitenkaan rajoita mallin mahdollistamien graafityyppien avaruutta. Tehtävien osalta malli on vahvasti rajattu käyttäjälle tarjottavien datakenttien lukumäärän ja muodon avulla.

Myös sovelluksen käyttöliittymä tavoittaa sille asetetut tavoitteet. Graafin rakentamisen osalta tärkein tavoite eli graafin rakentaminen pelkän näppäimistön avulla täyttyy. Tämä on mahdollista taulukkotyyppisen käyttöliittymän ja näppäinoikoteiden käyttämisen avulla. Graafin visualisoinnille ja selaamiselle asetettiin tavoitteeksi sujuvuus kosketusnäyttöä käyttävillä laitteilla. Tähän lopullinen yhden sarakkeen näkymä soveltuu hyvin, eikä selaamisessa tarvita lainkaan näppäimistöä.

Sovelluksen toteutus on pääosin toimiva. Sovellukseen on kuitenkin jäänyt jonkin verran toiminnallisia virheitä, jotka voivat aiheuttaa ongelmia sovelluksen käytössä. Suurin osa virheistä aiheuttaa vain käyttäjän korjattavissa olevia virheitä graafiin, mutta pahimmassa tapauksessa sovellus voi kaatua kokonaan kadottaen käyttäjän

tekemät muutokset.

7.2 Vertailu

Vertailun esimerkkitapauksena käytetään FPGA-pohjaista SoC-suunniteluvuota, joka alunperin on esitetty UML2.0 -aktivitettiikaaviossa [11]. Malli kuvaa laitteisto-, ohjelmisto- ja järjestelmäsuunnittelijoiden tehtäviä, näiden tehtävien järjestystä ja tehtävien välisiä datariippuvuuksia. Graafien selkeyttämiseksi kaikkia datariippuvuuksia ei ole mallinnettu. Yhteensä alkuperäisessä mallissa on 21 tehtävää, 2 jakautumista ja yhtymistä, 23 vuokaarta sekä 25 datakaarta.

Vertailussa mitataan mallin rakentamisen ja muokkaamisen monimutkaisuutta sekä näiden vaatimaa aikaa. Lisäksi tarkastellaan tulosgraafien luettavuutta ja ymmärrettävyyttä. Sovellusta verrataan kahteen saatavilla olevaan graafinpiirtotyökaluun: YAWL [5] ja Graphviz [1]. Näistä YAWL (alikohta 2.2.4) edustaa monipuolista mutta myös monimutkaista työkalua, kun taas Graphviz yksinkertaista tekstisyötteistä graafinpiirtotyökalua.

Taulukkoon 7.1 on koostettu vertailussa tehdyt havainnot. Seuraavissa alikohdissa kuvataan lyhyesti käyttökokemus kullakin vertailun työkalulla. Lisäksi esitetään työkalujen tuotoksena syntyneet graafit.

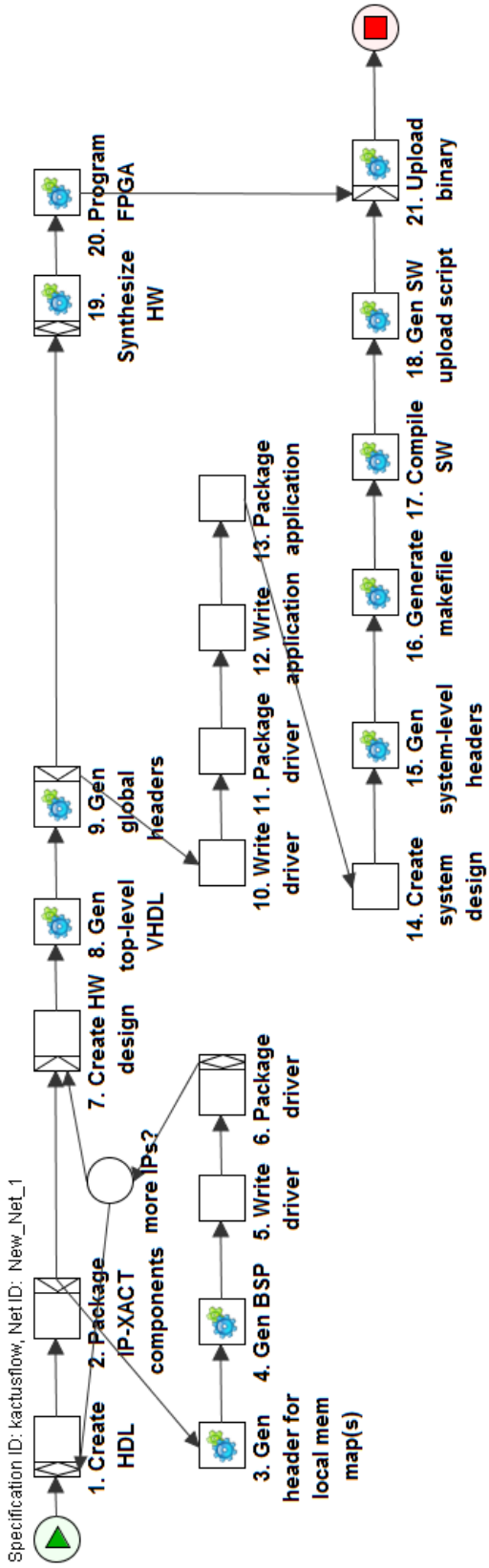
Taulukko 7.1 *Yhteenveto työkaluvertailusta.*

Ominaisuus \ Työkalu	**YAWL**	**Graphviz**	**Woke**
Mallin luontiaika [minuutteja]	20	15 (+45)	10 + 10
Asettelu	Manuaalinen	Automaattinen + manuaalinen	Automaattinen
Tehtävän luonnin kompleksisuus	Piirtäminen + teksti	Teksti	Teksti
Yleisnäkymä	Jakaantunut useaan näkymään	Jakaantunut yhdessä näkymässä	Listamainen
Muutosten visualisointi	Manuaalinen	Manuaalinen	Manuaalinen, automaattinen suunniteltu

7.2.1 YAWL

Vertailussa YAWL-työkalua käytettiin vain piirtämiseen tarkemman mallinnuksen sijaan. Aloittelija loi graafin työkalulla noin 20 minuutissa ilman suurempia vaikeuksia. Gaafin piirrossa noudatettiin alkuperäisen UML-graafin ulkoasua. Kaikki tieto kirjoitetaan näppäimistöllä, mutta hiirtä tarvitaan solmujen luomiseen ja kopioimiseen. Tämä hidastaa sovelluksen käyttöä jonkin verran.

YAWL-työkalussa vaakasuuntainen graafi todettiin toimivammaksi. YAWL-työkalulla piirretty graafi on esitetty kuvassa 7.1. Graafista puuttuvat kaikki datariippuuvuudet, samoin tehtävien yksityiskohdat.



Kuva 7.1 *Graafi YAWL-työkalulla. Esimerkkigraafi piirrettynä YAWL-työkalulla ilman datariippuvuuksia.*

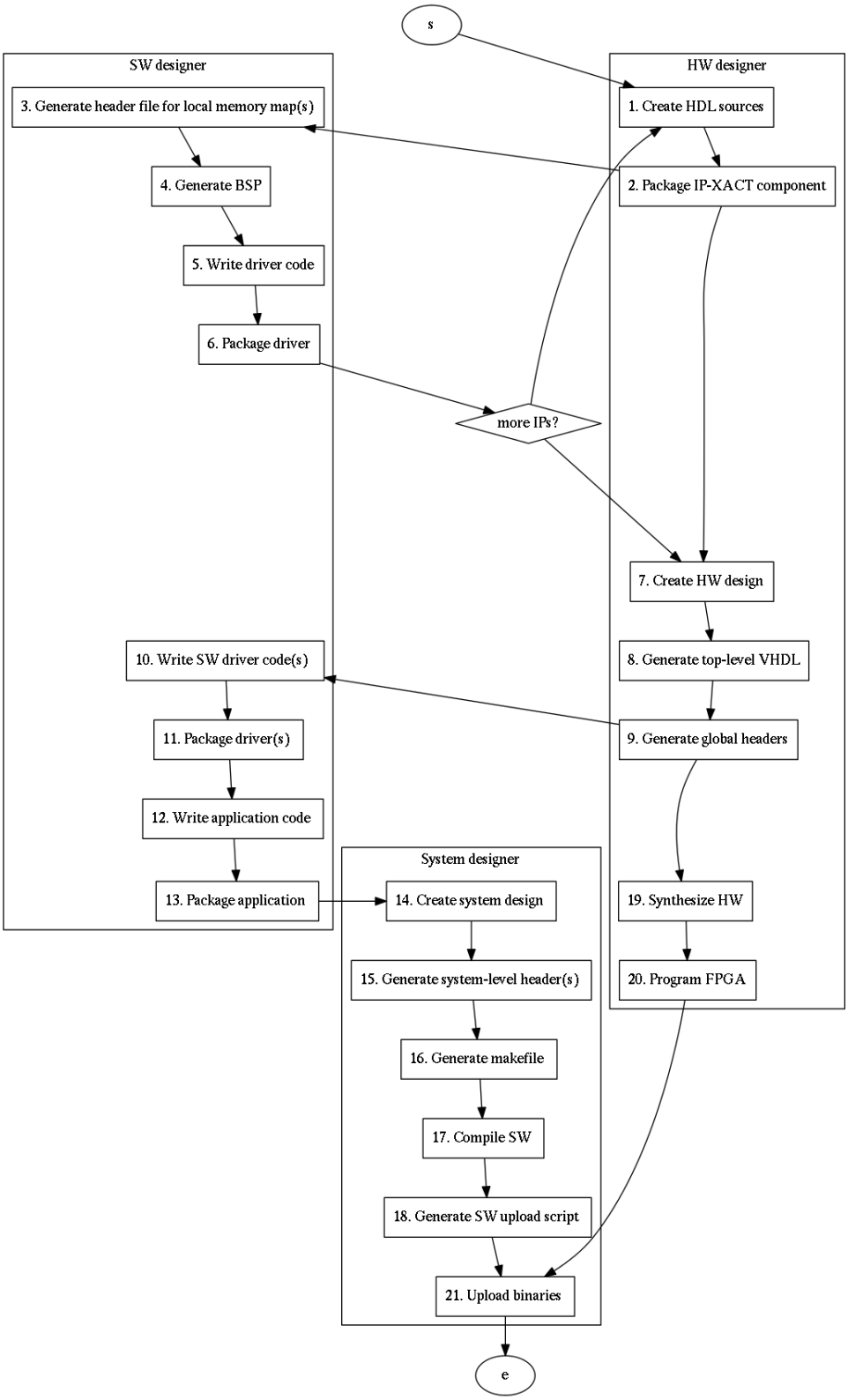
7.2.2 Graphviz

Kuvasssa 7.2 on esitetty esimerkkigraafi Graphviz-työkalulla piirrettynä ilman datariippuvuuksia. Työkalussa graafi rakennetaan siihen tarkoitetulla dot-kuvauskielellä [9]. Malli luotiin tekstieditorissa ja prosessoitiin komentorivillä graafimuotoon.

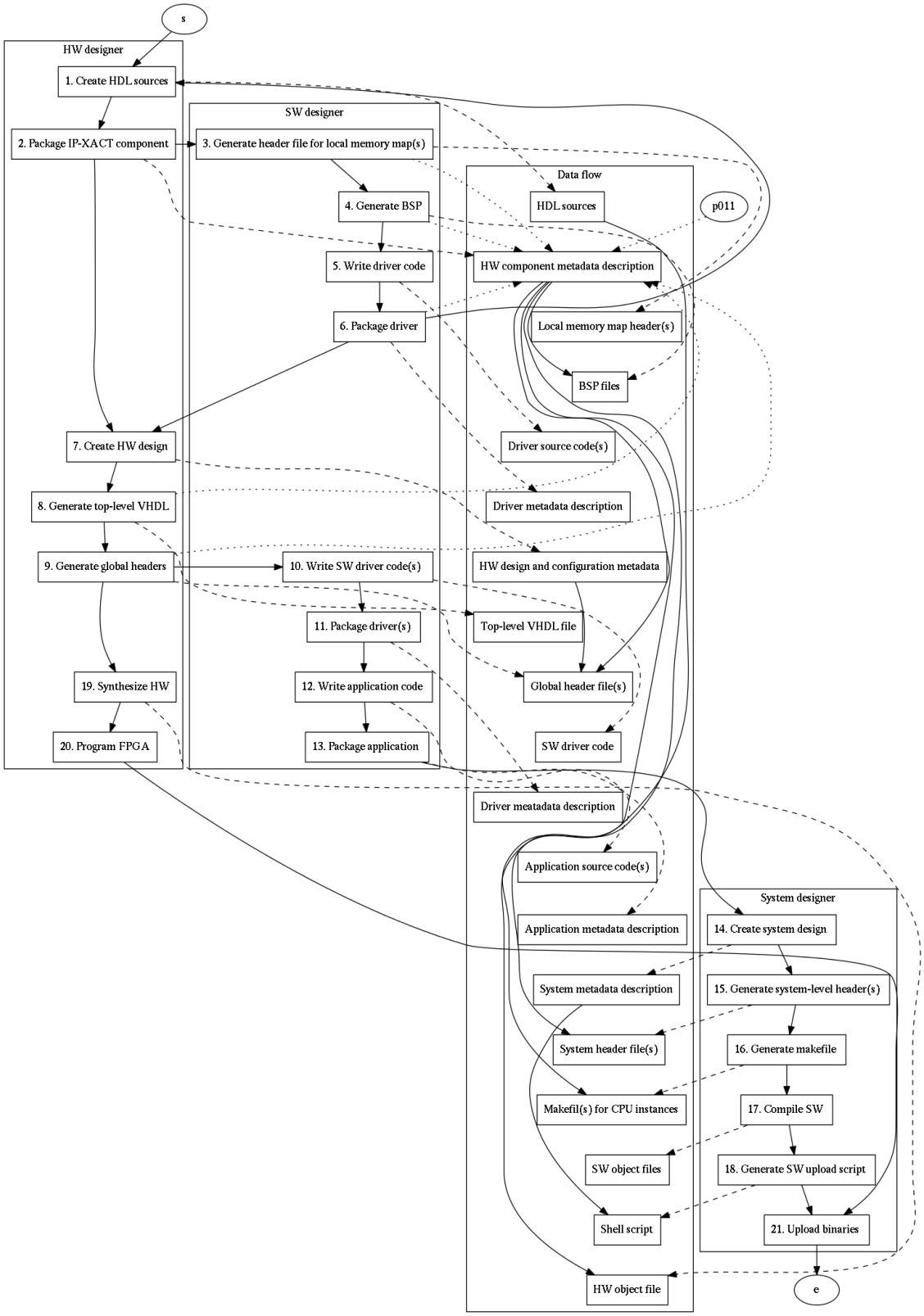
Graphivizilla luodut graafit esitetään pääosin ylhäältä alas -tyylisesti kuten Wokessa. Muutoksia on helppo tehdä, mutta graafin ulkoasu voi muuttua paljon muutoksia tehdessä. Tästä johtuen graafin rakentaminen oli melko nopeaa, 15-20 minuuttia, kun taas graafin muokkaaminen selkeämmän ulkoasun saamiseksi kesti ainakin 45 minuuttia.

Puhdas tekstimuotoinen syöte oli hyödyllistä, sillä se mahdollistaa kommenttien lisäämisen sekä tehtävien tai liitoksien väliaikaisen poistamisen. Lisäksi esimerkiksi etsintä- ja korvausoperaatiot toimivat hyvin. Tekstimuotoisen graafin prosessointi kuvamuotoon kesti vain joitakin sekunteja. Työkalu mahdollistaa useita erilaisia muotoja ja nuolityylejä.

Datatiedostojen lisääminen muuttaa kuitenkin graafin ulkoasua paljon, kuten kuvasta 7.3 nähdään. Graafi pysyy edelleen pääosin luettavana, mutta jotkin datariippuvuudet sotkevat koko graafia.



Kuva 7.2 *Graafi Graphviz-työkalulla. Esimerkkigraafi piirrettynä Graphviz-työkalulla ilman datariippuvuuksia.*

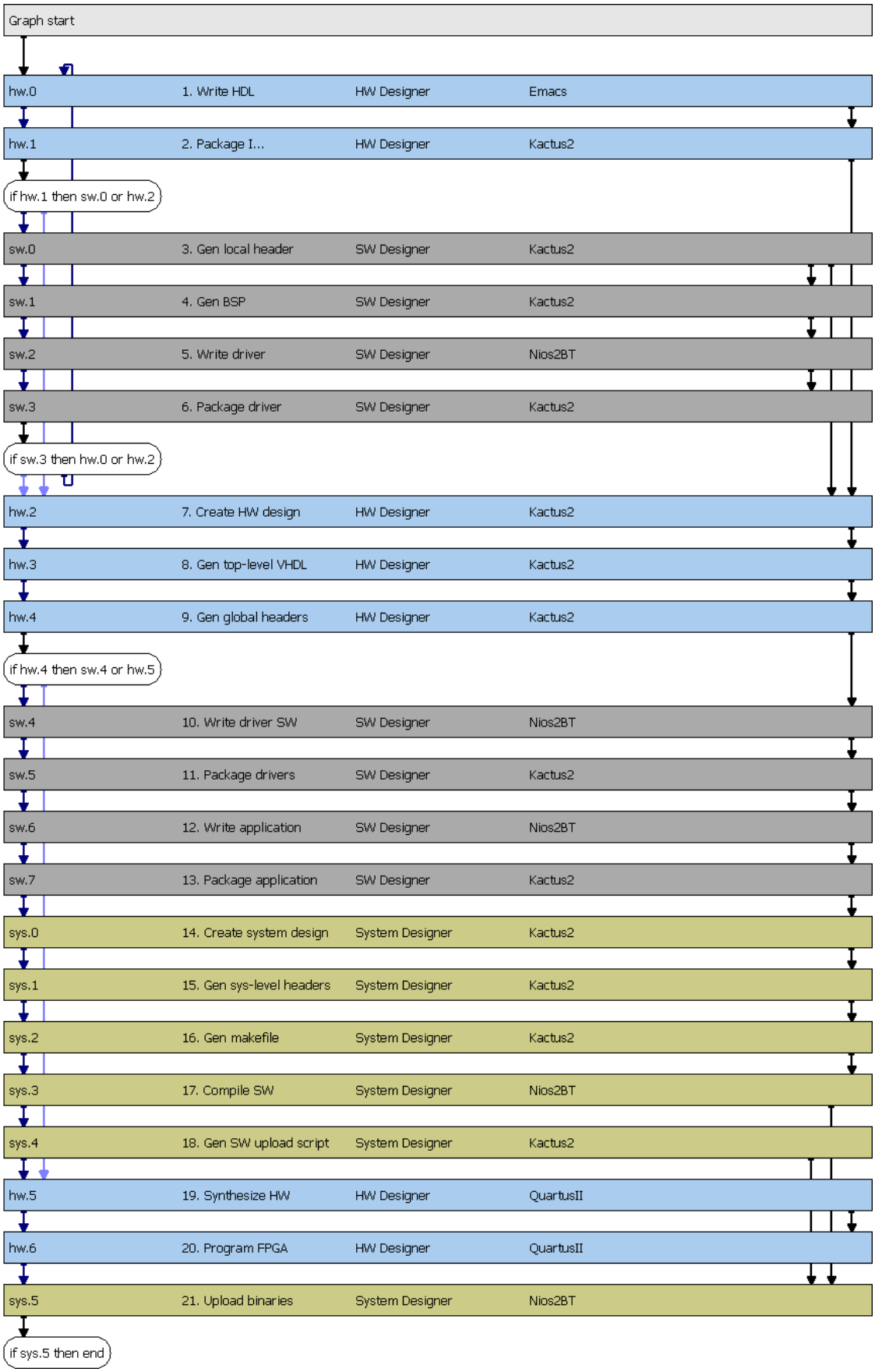


Kuva 7.3 *Datariippuvuudet Graphviz-työkalulla. Esimerkkigraafi piirrettynä Graphviz-työkalulla datariippuvuuksien kanssa.*

7.2.3 Woke

Mallin luonti Wokella on nopeaa, noin 10-15 minuuttia tehtävien ja siirtymien syöttämiseen sekä 10 minuuttia datariippuvuuksien lisäämiseen. Kuvassa 7.4 on esimerkkigraafi piirrettynä Woke-työkalulla datariippuuuksien kanssa.

Datariippuvuudet näkyvät graafin oikeassa reunassa. Datariippuvuudet näytetään graafissa vain korkealla tasolla, esimerkiksi datan nimeä ei näytetä graafissa. Lisäksi tehtävissä listataan vastuuhenkilöt ja käytetyt työkalut. Yhden sarakkeen asettelu sallii useita tietokenttiä kompaktissa ja luettavassa muodossa.



Kuva 7.4 *Graafi Woke-työkalulla. Esimerkkigraafi piirrettynä Woke-työkalulla datariippuvuuksien kanssa.*

7.3 Jatkokehitysmahdollisuudet

Valmiissa työkalussa on useita jatkokehitysmahdollisuuksia. Jatkokehitystä voitaisiin tehdä sekä nykyisten ominaisuuksien parantelemiseksi että kokonaan uusien ominaisuuksien lisäämiseksi.

Luonnollinen kehityskohde on sovellukseen jääneiden virheiden korjaaminen. Virheiden korjaaminen parantaisi käyttäjän käyttökokemusta huomattavasti. Lisäksi käyttökokemusta voitaisiin parantaa pienillä muutoksilla käyttöliittymän selkeyttämiseksi.

Yksi jo työn alkuvaiheessa suunniteltu jatkokehitysmahdollisuus on mahdollisuus lukea ja tuottaa muiden mallien tiedostomuotoja. Tämä mahdollistaisi yhteensopivuuden muiden mallinnustyökalujen kanssa, sekä työkalun käyttämisen vaikka jokin muu työkalu vaatisi tiettyä tiedostomuotoa. Tuki voidaan toteuttaa käyttämällä Qt-kehyksen tarjoamaa liitännäisrajapintaa.

Graafin piirtojärjestys voitaisiin optimoida graafin selkeyttämiseksi käyttämällä nykyistä monimutkaisempaa DF-ALAP -algoritmia. Sovellus voisi myös tarjota käyttäjälle mahdollisuuden valita usean erilaisen piirtoalgoritmin välillä, jolloin käyttäjä voisi itse päättää kullekin graafille parhaiten sopivan tyylin.

Myös graafin visualisointiin littyviä ominaisuuksia voitaisiin edelleen parantaa, erityisesti tarjoamalla lisäominaisuuksia näkyvyyden rajoittamiseen. Käyttäjä voisi esimerkiksi rajata graafin näyttämään vain tiettyyn resurssiin tai työkaluun liittyvät tehtävät. Työkalu osaisi edelleen piirtää rajatusta näkymästä loogisen, oikeaa työvuota vastaavan graafin.

8. YHTEENVETO

Diplomityössä suunniteltiin kokonaan uusi työkalu työvuon mallintamiseen. Tämä tarkoitti mallintamiseen käytettävän metamallin suunnittelua, mallin muokkaamiseen ja esittämiseen käytettävän graafisen sovelluksen suunnittelua sekä graafisen C++-sovelluksen toteuttamista. Toteutuksen jälkeen sovellusta verrattiin olemassaoleviin vastaavaan käyttöön tarkoitettuihin työkaluihin.

Työn aluksi selvitettiin muiden työvuon mallinnukseen käytettävien mallien ominaisuuksia. Malleista haluttiin poimia hyväksi koettuja ominaisuuksia, ja samalla mietittiin millaisilla ominaisuuksilla mallinnusta voitaisiin tehostaa näihin malleihin verrattuna. Tehtyjen havaintojen pohdalta suunniteltiin työvuomallin rakenne ja sen sisältämä tieto. Työvuomallin tallennusmuodoksi valittiin XML-tiedosto.

Seuraavaksi pohdittiin työkalua käytettävyyden näkökulmasta. Työkalun tavoitteiksi valittiin jo työn alussa nopea muokattavuus näppäimistön avulla, sekä erityisesti taulutietokoneiden käyttöliittymälle soveltuva visualisointi. Muokkaaminen ja selaaminen eriytettiin selkeästi erillisiksi käyttötapauksiksi: muokkaaminen tapahtuisi tavallisella tietokonella hiiren ja näppäimistön avulla, visualisointi ja selaaminen taulutietokoneen kosketusnäytöllä. Näin käyttöliittymä voitiin suunnitella kunkin tilanteen ja alustan vahvuuksien perusteella.

Kun mallin rakenne ja käyttöliittymän tavoitteet olivat selvillä, voitiin aloittaa itse sovelluksen toteutus. Sovellus toteutettiin aikaisemmista projekteista tutulla tavalla C++-kielellä Qt-ohjemistokehyksen päälle. Koska varsinainen toteutusalusta oli jo työn alkuvaiheessa pääosin tuttu, toteutusvaihe sujui hyvin. Toteutuksen aikana sekä mallissa että alkuperäisessä käyttöliittymäsuunnitelmassa havaittiin puutteita ja parannusmahdollisuuksia, joihin voitiin puuttua välittömästi.

Työn lopuksi sovellusta verrattiin olemassaoleviin sovelluksiin. Sovellus osoittautui vertailussa onnistuneeksi. Graafin rakentaminen oli yhtä nopeaa tai nopeampaa kuin vertailun muilla työkaluilla ja graafi pysyi helposti luettavana.

Työn suurimmaksi haasteeksi osottautui suurehkon ohjelmistoprojektin suunnitteleminen ja toteuttaminen yksin. Mahdolliset heikot suunnittelu- ja toteutusratkaisut havaittiin vasta myöhäisessä vaiheessa, jolloin niiden korjaamiseksi vaadittiin huomattavaa työtä. Usean henkilön projekteissa vastaavat ongelmat havataan usein aikaisemmassa vaiheessa.

Työkaluun jäi useita jatkokehitysmahdollisuuksia. Näitä ovat esimerkiksi tuki muiden työvuokuvaukseen käytettyjen mallien tiedostomuodoille, sekä erilaisten järjestelyalgoritmien tarjoaminen graafin piirrossa.

LÄHTEET

[1] *Graphviz - Graph Visualization Software*, [Online] http://www.graphviz.org/.

[2] *SourceForge*, [Online] https://sourceforge.net/.

[3] *Woke Code*, [Online] https://sourceforge.net/p/woke/code/HEAD/tree/.

[4] *Woke Files*, [Online] https://sourceforge.net/projects/woke/files/.

[5] *YAWL Foundation*, [Online] http://www.yawlfoundation.org/.

[6] *Apache Subversion*, Apache Software Foundation, [Online] https://subversion.apache.org/.

[7] D. Blaza and A. Wolfe, "2013 embedded market study," 2013, Saatavissa: http://presentations.ubmdesign.com/events/san-jose/2013.

[8] *C++ Standard*, C++ Standards Committee, Saatavissa: https://isocpp.org/std/the-standard.

[9] E. Gansner, E. Koutsofios, and S. North, *Drawing graphs with dot*, 2015, Saatavissa: http://www.graphviz.org/pdf/dotguide.pdf.

[10] M. Honkonen, L. Matilainen, E. Salminen, and T. D. Hämäläinen, "Woke: A novel workflow model editor," International Symposium on System-on-Chip, Tampere, Finland, October 28-29, 2014.

[11] A. Kamppi, L. Matilainen, J.-M. Määttä, E. Salminen, and T. D. Hämäläinen, "Extending ip-xact to embedded system hw/sw integration," International Symposium on System-on-Chip, Tampere, Finland, October 23-24, 2013.

[12] L. Matilainen, A. Kamppi, J.-M. Määttä, E. Salminen, and T. D. Hämäläinen, "KACTUS2: IP-XACT/IEEE1685 compatible design environment for embedded MP-SoC products," *TUT Report 37, ISBN 978-952-15-2625-1*, 2011.

[13] *MVC architecture*, Mozilla Foundation, [Online] https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture.

[14] *Unified Modeling Language Specification Version 1.1*, Object Management Group, 1997, Saatavissa: http://www.omg.org/cgi-bin/doc?ad/97-08-11.

[15] *Unified Modeling Language Specification Version 1.5*, Object Management Group, 2005, Saatavissa: http://www.omg.org/spec/UML/1.5/.

[16] *Unified Modeling Language Specification Version 2.0*, Object Management Group, 2005, Saatavissa: http://www.omg.org/spec/UML/2.0/.

[17] *Unified Modeling Language Specification Version 2.5*, Object Management Group, 2005, Saatavissa: http://www.omg.org/spec/UML/2.5/.

[18] *BPMN 2.0 by Example*, Object Management Group, 2011, Saatavissa: http://www.omg.org/spec/BPMN/2.0/.

[19] *Business Process Model and Notation (BPMN) Version 2.0*, Object Management Group, 2011, Saatavissa: http://www.omg.org/spec/BPMN/2.0/.

[20] *Model/View Programming*, The Qt Company, [Online] http://doc.qt.io/qt-5/model-view-programming.html.

[21] *Qt*, The Qt Company, [Online] https://www.qt.io/.

[22] *Qt 5.2.0*, The Qt Company, [Online] https://download.qt.io/official_releases/qt/5.2/5.2.0/.

[23] M. Richardson, "Differences Between UML 1.x and UML 2.0," [Online] Saatavissa: http://www.michael-richardson.com/processes/rup_for_sqa/core.base_rup/guidances/supportingmaterials/differences_between_uml_1_x_and_uml_2_0_CA70F2E6.html.

[24] *Kactus2*, Tampereen teknillinen yliopisto, [Online] http://funbase.cs.tut.fi/.

[25] W. van der Aalst and A. ter Hofstede, *YAWL: Yet Another Workflow Language (Revised version)*, 2003.

[26] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, World Wide Web Consortium, 2008, Saatavissa: https://www.w3.org/TR/2008/REC-xml-20081126/.

[27] *YAWL release 4.0*, YAWL Foundation, 2016, Saatavissa: https://github.com/yawlfoundation/yawl/releases/tag/v4.0.

LIITE 1. XML-LISTAUS

```
<!DOCTYPE savedata>
<rootElement>
 <tasks>
  <task>
   <name>suunnittelu</name>
   <id>0</id>
   <description>Vaatimukset</description>
   <resource>Projektipäällikkö</resource>
   <tool>Excel</tool>
   <time>1w</time>
   <references/>
   <color>#AAE</color>
   <outputData/>
   <inputData/>
  </task>
  <task>
   <name>suunnittelu</name>
   <id>1</id>
   <description>Määrittely</description>
   <resource>Arkkitehti</resource>
   <tool>Word</tool>
   <time>1w</time>
   <references/>
   <color>#AAE</color>
   <outputData/>
   <inputData/>
  </task>
  <task>
   <name>testaus</name>
   <id>0</id>
   <description>Prototyyppi</description>
   <resource>Testaaja</resource>
   <tool>Python</tool>
```

```
   <time>1w</time>
   <references/>
   <color>#AFA</color>
   <outputData/>
   <inputData/>
  </task>
  <task>
   <name>toteutus</name>
   <id>0</id>
   <description>Ohjelmointi</description>
   <resource>Ohjelmoija</resource>
   <tool>C++</tool>
   <time>1m</time>
   <references/>
   <color>#FAA</color>
   <outputData/>
   <inputData/>
  </task>
  <task>
   <name>toteutus</name>
   <id>1</id>
   <description>Prototyyppi</description>
   <resource>Ohjelmoija</resource>
   <tool>Python</tool>
   <time>1w</time>
   <references/>
   <color>#FAA</color>
   <outputData/>
   <inputData/>
  </task>
 </tasks>
 <transitions>
  <transition>
   <content>
    <line>if suunnittelu.0 then suunnittelu.1</line>
    <line></line>
```

```
   </content>
   <activeOutputs>
    <output>suunnittelu.1</output>
   </activeOutputs>
  </transition>
  <transition>
   <content>
    <line>if suunnittelu.1 then toteutus.0 and testaus.0</line>
    <line></line>
   </content>
   <activeOutputs>
    <output>toteutus.0</output>
    <output>testaus.0</output>
   </activeOutputs>
  </transition>
  <transition>
   <content>
    <line>if toteutus.0 then toteutus.1</line>
    <line></line>
   </content>
   <activeOutputs>
    <output>toteutus.1</output>
   </activeOutputs>
  </transition>
 </transitions>
</rootElement>
}
```