Antti-Juhani Kaijanaho

LaTeX-tutkielmapohjan gradu3 käyttö

Tietotekniikan tutkielmapohjan esimerkki ja käsikirja

27. lokakuuta 2014

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Antti-Juhani Kaijanaho

Yhteystiedot: Ag C416.1, antti-juhani.kaijanaho@jyu.fi

Ohjaaja: Ohjaamaton työ

Työn nimi: LaTeX-tutkielmapohjan gradu3 käyttö

Title in English: Usage of the gradu3 document class for LaTeX theses

Työ: Tutkielmapohjan esimerkki ja käsikirja

Suuntautumisvaihtoehto: Kaikki suuntautumisvaihtoehdot

Sivumäärä: 34+2

Tiivistelmä: Tämä kirjoitelma on esimerkki siitä, kuinka gradu3-tutkielmapohjaa käytetään. Se sisältää myös käyttöohjeet ja tutkielman rakennetta koskevia ohjeita.

Tutkielman tiivistelmä on tyypillisesti lyhyt esitys, jossa kerrotaan tutkielman taustoista, tavoitteesta, tutkimusmenetelmistä, saavutetuista tuloksista, tulosten tulkinnasta ja johtopäätöksistä. Tiivistelmän tulee olla niin lyhyt, että se, englanninkielinen abstrakti ja muut metatiedot mahtuvat kaikki samalle sivulle.

Avainsanat: LaTeX, gradu3, pro gradu -tutkielmat, kandidaatintutkielmat, käyttöohje

Abstract: This document is a sample gradu3 thesis document class document. It also functions as a user manual and supplies guidelines for structuring a thesis document.

The English abstract of a thesis should usually say exactly the same things as the Finnish tiivistelmä.

Keywords: LaTeX, gradu3, Master's Theses, Bachelor's Theses, user's guide

Esipuhe

Tähän voit kirjoittaa tutkielmasi esipuheen. Tutkielmissa on harvemmin esipuheita, mutta jos sen kirjoitat, pidä se lyhyenä (enintään sivu).

Esipuheen tulisi kertoa ennemminkin tutkielmaprosessista kuin tutkielman sisällöstä. Esimerkiksi jos tutkielman aiheen valintaan tai tekemiseen liittyy jokin erikoinen sattumus, voit siitä kertoa esipuheessa. Tapana esipuheessa on myös kiittää nimeltä mainiten tärkeimpiä tutkielman tekemisessä auttaneita ihmisiä – ainakin ohjaajia, puolisoa ja lapsia. (Yleensä perhe on auttanut vähintään tukemalla ja kannustamalla.)

Esipuhe kannattaa kirjoittaa minä-muodossa. Tavanomaista on myös allekirjoittaa se.

Jyväskylässä 27. lokakuuta 2014

Tutkielman tekijä

Termiluettelo

Sovellusarkkitehtuuri	Donald Knuthin 1977–1989 laatima eräajotyyppinen ladontajärjestelmä (**knuth86:_texbook**).
MVC	TEXin (**knuth86:_texbook**) päälle rakennettu rakenteisten kirjoitelmien ladontaan tarkoitettu järjestelmä (**lamport94:_latex**). Siitä on nykyään käytössä versio LATEX 2_ε.
Sovelluskehys	TEXin (**knuth86:_texbook**) päälle rakennettu rakenteisten kirjoitelmien ladontaan tarkoitettu järjestelmä (**lamport94:_latex**). Siitä on nykyään käytössä versio LATEX 2_ε.

1 Johdanto

1.1 Tausta

1.2 Tutkimuskysymys

MVC-arkkitehtuuri on saanut paljon huomiota web-sovelluskehyksien toteutuksissa ja useat web-sovelluskehykset ovat luokiteltu MVC-pohjaisiksi sovelluskehyksiksi (Bestframeworks 2009). Se on kuitenkin alunperin tarkoitettu matalan tason käyttöliittymäsovellusten toteuttamiseen, jossa esimerkiksi hallitaan yksittäisiä näppäimistöltä tulleita syötteitä eikä sitä ole suoraan tarkoitettu käytettäväksi web-sovellusten ohjelmointiin. Alkuperäisen MVC:n toteutuksen soveltuvuutta web-ohjelmointiin onkin epäilty. Esimerkiksi Leff soveltaa artikkelissaan MVC:n käyttämistä web-sovelluksissa, mutta samalla esittelee alkuperäisen MVC:n toteuttamisen ongelmana. Tämä johtuu web-sovelluksen jakautumisesta asiakkaan (client) ja palvelimen (server) välille (Avraham 2001). Myös Pyramid-sovelluskehyksen tekijät kyseenalaistavat MVC-arkkitehtuurin toteutuksen Pyramidissa ja uskovat MVC:n olevan sellaisenaan sopimaton web-ohjelmointiin, vaikka Pyramidin toteutus onkin hyvin lähellä alkuperäistä MVC:tä (Pyramid 2010). Django on myös toteutettu MVC:n pohjalta, mutta se ei kuitenkaan toteuta MVC:tä sellaisenaan (Django 2005).

Tutkimuksen tarkoituksena on selvittää millä tavalla MVC-arkkitehtuuri on toteutettu web-sovelluskehyksissä ja miten se eroaa alkuperäisestä MVC:n toteutuksesta. Lisäksi laajennetaan Flask-sovelluskehys toteuttamaan alkuperäinen MVC niinkuin Krasner on sen määritellyt (Krasner 1988a). Havaintojen pohjalta selvitetään voidaanko MVC tuoda sellaisenaan sovelluskehykseen ja mitä mahdollisia ongelmia se tuo.

2 Oletus

Tässä oletetaan mitä tutkimuskysymykseen voisi tulla vastaukseksi

3 Tutkimuksen Rakenne

Tutkimus aloitetaan kirjallisuuskatsauksella, jossa tarkastellaan mitä aiempaa tutkimusta MVC:stä on tehty. Lisäksi käydään läpi mitä lähteitä löytyy Python-pohjaisista web-sovelluskehyksistä. Tämän jälkeen tutkitaan MVC:n historiaa sekä millä tavalla MVC on tarkoitettu toteutettavaksi. Tässä vaiheessa käydään läpi jokaisen MVC-komponentin tarkoitus sekä niiden keskinäisen kommunikaation rakentuminen. Lisäksi esitellään Dortmundin yliopistossa kirjoitettu esimerkkiohjelma Smalltalkilla siitä miten MVC:n toteutus tuodaan sovellukseen käytännössä.

MVC:n tarkastelun jälkeen esitellään tutkimuksessa käytetyt web-sovelluskehykset, joita käytetään apuna MVC:n tutkimisessa. Sovelluskehyksistä käydään läpi sen historia sekä yleisellä tasolla mihin käyttötarkoitukseen sovelluskehys on tarkoitettu. Tämän jälkeen verrataan MVC:n toteutusta erikseen jokaiseen sovelluskehykseen ja selvitetään millä tavalla niiden sovellusarkkitehtuuri mahdollisesti eroaa MVC:stä, Havaintojen perusteella pohditaan MVC:n mahdollisia ongelmia sovelluskehyksien toteutuksessa ja selvitetään löytyykö sovelluskehyksien arkkitehtuurista jotain yhtenäisiä piirteitä, mitkä ovat kytköksissä MVC:n toteutukseen. Saatujen tulosten pohjalta kirjoitetaan Flask-sovellus, joka toteuttaa MVC:n niinkuin se on alunperin tarkoitettu.

Tutkimuksen lopuksi koostetaan havainnoista yhteenveto, jossa pohditaan saatuja tuloksia ja selvitetään pystytäänkö niiden perusteella vastaamaan tutkimuskysymykseen.

3.1 Aiheen rajaus

Tutkimus on rajattu tarkastelemaan MVC-arkkitehtuurin toteutusta Pythonilla kirjoitetuissa web-sovelluskehyksissä. Tarkasteltavat web-sovelluskehykset rajataan Pyramid-, Django-, Plone- sekä Flask-sovelluskehyksiin. Pyramid, Django ja Plone toteuttavat MVC:n kaltaisen sovellusarkkitehtuurin. Flask on sovelluskehys, joka tarjoaa vain välttämättömät kirjastot web-sovelluksen toteuttamiseen. Sitä käytetään tutkimuksessa työkaluna selvittämään miten MVC tulisi toteuttaa sovelluskehykseen. MVC:stä on olemassa erilaisia versioita, joten sen määrittely tulee rajata tarkasti. Kun puhutaan MVC:stä tarkoitetaan tällä Krasnerin artikke-

lissa esiteltyjä määrittelyitä MVC:n toteutuksesta (Krasner 1988a), jotka pohjautuvat Trygve Reenskaugin esittelemään MVC:n määritelmään (Parc 1979a).

Tarkasteltavat sovelluskehykset käydään ensiksi yleisellä tasolla läpi, jonka jälkeen niitä tarkastellaan MVC:n näkökulmasta. Yleisellä tasolla tarkoitetaan sovelluskehyksen historian ja käyttötarkoituksen esittelemistä. Sovelluskehyksien muihin teknisiin ominaisuuksiin ei oteta kantaa. Flask-osiossa MVC-arkkitehtuuri toteutaan niinkuin se on Krasnerin julkaisussa määritelty. Näiden havaintojen pohjalta pyritään vastamaan tutkimuskysymykseen.

4 Kirjallisuuskatsaus

4.1 Toteutus

Kirjallisuuskatsauksessa käydään läpi vaihe vaiheelta, miten lähdemateriaalia kerätään tutkimusta varten. Lähdemateriaalin haku toteutetaan hakukoneilla, jotka ovat tarkoitettu erityisesti tieteellisten artikkeleiden etsimiseen. Tässä tutkielmassa käytetyt hakukoneet ovat seuraavat: IEEE Xplore, ACM Digital Library, Google Scholar sekä joissakin tapauksissa Google:n yleinen hakukone. Yleistä hakukonetta on käytetty esimerkiksi sovelluskehyksien dokumentaatioiden etsintään.

Aluksi muodostetaan kokonaiskuva tuloksista, jolloin silmäillään läpi saatuja artikkeleita. Tässä vaiheessa tarkoitus ei ole vielä valita mitään pohjaksi tutkielmalle, vaan kerätä informaatiota siitä millainen lähdemateriaali on tarjolla kokonaisuudessaan. Saaduista tuloksista poimitaan artikkeleita, jotka sopivat tutkimuksen aihepiiriin. Seuraavaksi artikkeleista valitaan tutkielmalle pohjakirjallisuus. Tässä vaiheessa artikkelit luetaan huolellisesti läpi ja varmistutaan siitä, että ne ovat tieteellisesti päteviä tutkielmaa varten. Erityisesti kiinnitetään huomiota viittauksien määrän valittaessa tärkeimmät lähdemateriaalit. Tutkielmassa esiintyy myös satunnaisia viittauksia, joita ei ole kirjallisuuskatsauksessa mainittu. Tutkimuksen pääkirjallisuus kuitenkin käydään läpi kirjallisuuskatsauksessa. Haussa käytetään seuraavia hakutermejä: "MVC", "MVC Architecture", "frameworks", "web frameworks"ja "MVC- Architecture". Erityisesti artikkeleita löytyy MVC-arkkitehtuurin soveltamisesta erilaisissa tekniikoissa. Tarkasteltavat artikkelit rajataan kuitenkin niihin, jotka esittelevät suoraan MVC:tä itseään tai tarjoavat lähdemateriaalin sovelluskehyksien esittelyyn.

4.2 MVC

Google Scholarin tuloksista löytyy kolme artikkelia MVC:stä, jotka sopivat lähdemateriaaliksi tutkimukseen. Ensimmäinen artikkeleista on John Deaconin kirjoittama artikkeli, joka tarkastelee lyhyesti MVC:tä (Deacon 2009). Artikkeli on kuitenkin hyvin suppea, mutta selittää tiivistetysti MVC:n idean. Toinen artikkeli on Steve Burbeckin kirjoittama, joka käsit-

telee MVC:tä sellaisena kuin sitä käytettiin Smalltalkissa (Steve 1992). Burbeckin artikkeliin viitataan monissa MVC:tä käsittelevissä julkaisuissa, joten sen arvo tämän tutkielman pohjakirjallisuudessa on vahva. Viittausten määrä on katsottu hakemalla artikkelia Google Scholarin hakukoneessa. Viittauksia kyseiseen artikkeliin on kirjoitushetkellä 308. Seuraavaksi kartoitetaan pohjakirjallisuutta käyttäen ACM Digital Library sekä IEEE XPlore -hakukoneita. Kolmas artikkeli Glenn E. Krasnerin kirjoittama julkaisu, jossa esitellään MVC:n toteutusta erilaisissa Smalltalk-sovelluksissa. Julkaisusta löytyy useita versioita, joista tässä tutkielmassa käytetään molempia (Krasner 1988a) (**.**) Tähän artikkeliin on myös viitattu runsaasti, joten se on Burbeckin julkaisun kanssa tärkeimpiä lähteitä MVC:n pohjakirjallisuudessa. Kirjoitushetkellä viittauksia Krasnerin artikkeliin on 2263. Monien MVC-arkkitehtuuria soveltavien artikkeleiden lähdeviitteistä löytyy viittauksia Burbeckin ja Krasnerin artikkeleihin. Tämän perusteella pystytään toteamaan kyseisten artikkeleiden olevan tieteellisesti päteviä ja tarjoavan kattavan lähdemateriaalin MVC:n pohjaksi. Burbeckin ja Krasnerin kirjoittamien artikkeleiden taustalta löytyy MVC-arkkitehtuurin alkuperäinen kehittäjä Trygve Reenskaug, jonka omia julkaisuja sekä kotisivujen MVC-osiota käytetään myös lähteenä tutkielmassa (Parc 1978). Erityisesti Reenskaugin ja Adele Goldbergin julkaisu, jossa kerrotaan jokaisen MVC komponentin tehtävä....

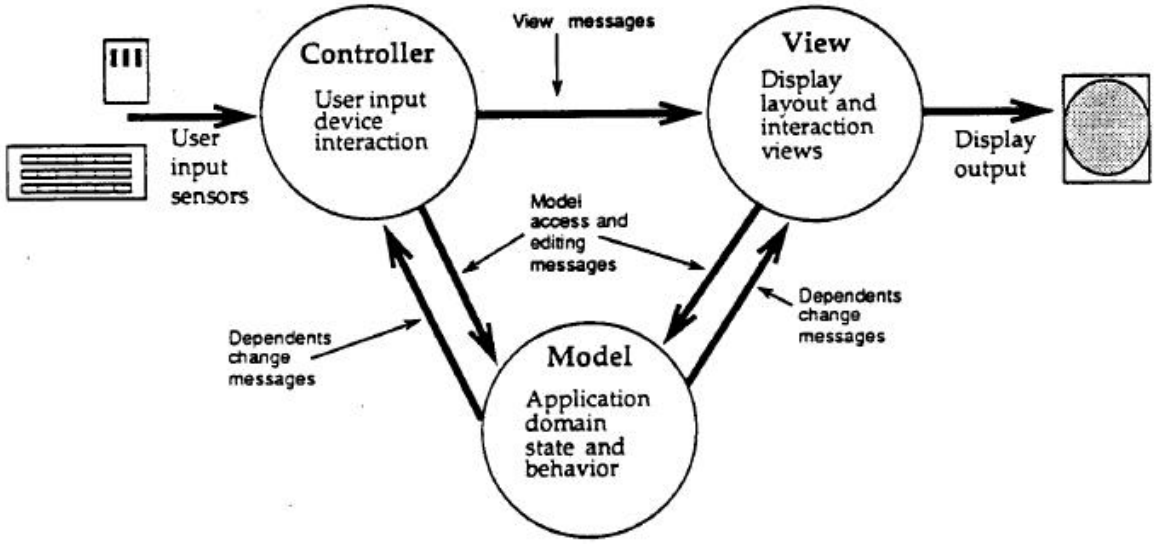
4.3 Web-sovelluskehykset

Web-sovelluskehyksistä löydetty kirjallisuus on hyvin suppea, eikä niiden varaan voida rakentaa kovinkaan perusteellista tieteellistä pohjaa. Tämän vuoksi tutkimuksessa joudutaan osaksi turvautumaan sovelluskehyksien omaan dokumentatioon täydentämään lähdemateriaalia. IEEE Xploren ja ACM Digital Libraryn avulla löytyy kolme julkaisua, joita käytetään tutkimuksen pohjana sovelluskehyksiä tarkastellessa. Ensimmäinen artikkeli on Okanovicin ja Mateljan kirjoittama artikkeli, jossa esitellään web-sovelluskehyksien suunnittelua (Ockanovic 2011). Se myös sivuuttaa lyhyesti MVC:tä. Toisena artikkelina käytetään ACM:stä tuloksena saatua Iwan Vosloon julkaisua, jossa käydään läpi yleisesti web-sovelluskehyksien rakennetta (Kourie 2008). Kolmanneksi hyödynnetään Ignacion artikkelia, jossa esitellään ketteriä web-sovelluskehyksiä sekä millä tavalla näitä tulisi vertailla (Fernández-Villamor 2008).

Google Scholarin hakutuloksista löytyi Liza Daly:n kirjoittama ja O'Reillyn julkaisema "Next Generation Web Frameworks in Python", joka sisältönsä puolesta sopii hyvin pohjaksi tutkimuksessa käsiteltävien sovelluskehyksien lähdemateriaaliksi (Daly 2007).

5 MVC

MVC-arkkitehtuurin perusajatus on erottaa käyttöliittymä sovelluslogiikasta ja näin tehdä sovelluksesta helposti ylläpidettävä kolmen eri komponentin avulla: Malli (Model), Näkymä (View) ja Ohjain (Controller). Jokainen komponentti on erikoistunut sovelluksessa johonkin tiettyyn tehtävään. Mallin tehtävänä on hallita sovelluksen tilaa ja vastata sen käsittelemästä datasta ohjaimelle ja näkymälle. Näkymän tehtävänä on taas näyttää sovelluksen käyttöliittymä ja sitä kautta mallin dataa. Ohjaimen tarkoitus on ottaa vastaan syötteitä käyttäjältä käskien mallia ja näkymää muuttumaan tarvittaessa.



Kuvio 1. Model-View-Controller State and Message Sending Krasner 1988b, s. 5

Jokaisella komponentilla on oma rajattu tehtävänsä ja ohjelmakoodi tulee jakaa näiden komponentien kesken. Jotta MVC:tä pystyttäisiin käyttämään tehokkaasti, tulee ymmärtää komponenttien työnjako sekä se kuinka komponentit kommunikoivat keskenään (Steve 1992).

Luodessamme MVC-arkkitehtuurin toteuttavia komponentteja, tulee ne periä jostakin abstraktista pohjaluokasta (Model, View tai Controller), joka määrittelee kyseisen komponentin käyttäytymisen MVC:ssä (Krasner 1988b, s. 5). Tässä kappaleessa käydään jokaisen komponentin toteutus erikseen läpi käyttäen ohjelmointikielenä Smalltalkia. Lähteenä käytetään Krasnerin julkaisua (Krasner 1988b).

Yleisesti MVC-komponenttien toimintaa kuvaavassa esimerkissä käyttäjältä tulee jokin syöte, jonka sillä hetkellä aktiivinen ohjain ottaa vastaan. Syötteen perusteella ohjain lähettää mallille viestin. Malli puolestaan tekee sille määrättyjä operaatioita muuttaen tilaansa ja lähettää edelleen viestin muutoksestaan kaikille siihen liitetyille riippuvuuksille (näkymät ja ohjaimet). Näkymät voivat tämän jälkeen kysyä mallilta sen nykyistä tilaa ja päivittää itsensä, jos siihen on tarvetta. Ohjaimet voivat myös muuttaa tilaansa riippuen mallin tilasta (Krasner 1988b, s. 4).

Suurin merkitys MVC:llä on luoda silta ihmismielen hahmottamalle mallille ja tietokoneessa esiintyvälle mallille. Oikein toteutettuna MVC:n avulla luodaan illuusio siitä, että käyttäjä kommunikoi suoraan mallin kanssa. Todellisuudessa kuitenkin ohjain ja näkymä muodostavat yhdessä rajapinnan sille, miltä malli näyttää ulospäin ja miten sitä käsitellään. Ohjain huolehtii syötteiden vastaanottamisesta ja käsittelemisestä. Näkymä taas huolehtii mallin graafisesta puolesta (Reenskaug 2003, s. 11-12).

5.1 Historia

MVC:n esitteli Norjalainen Trygve Reenskaug ollessaan mukana Xerox PARC -tutkimushankkeessa. Ensimmäinen julkaisu MVC:stä kirjoitettiin vuonna 1978 samassa tutkimuskeskuksessa. Tuolloin julkaisussa esiteltiin kolmen komponentin sijasta neljä komponenttia: Malli (Model), Näkymä(View), Ohjain(Controller) sekä Muokkaaja(Editor). Muokkaaja on väliaikainen komponentti, jonka näkymä luo itsensä ja syötelaitteiden välille. Muokkaaja-komponentista kuitenkin luovuttiin käsitteenä ja se sisällytettiin näkymään ja ohjaimeen (Parc 1978). Alkuperäinen Xerox PARC:n tuottama raportti MVC:stä oli Reenskaugin vuonna 1979 kirjoittama THING-MODEL-VIEW-EDITOR (Parc 1979b). Raportti esitteli MVC:n komponentteja käyttäen hyväksi esimerkkejä Reenskaugin omasta suunnittelutyöstä. Thing-komponentilla mallinnettiin jotakin isompaa kokonaisuutta, joka hallitsee pienempiä kokonaisuuksia. Sitä voidaan ajatella eräänlaisena suurena mallina,joka on jaettu useisiin pienempiin malleihin. Editor-komponentti luo rajapinnan käyttäjän ja yhden tai useamman näkymän välille. Se tarjoaa käyttäjälle sopivan komento-rajapinnan kuten esimerkisi valikon, joka vaihtuu sisällön muuttuessa (Parc 1979b). Reenskaug hylkäsi kuitenkin Editor- ja Thing-komponentin ja päätyi Adele Goldbergin avustuksella termeihin Models-Views-Controllers julkaisten saman

vuoden lopulla raportin, jossa määritellään lyhyesti jokaisen komponentin tehtävä (MODELS-VIEWS-CONTROLLERS) (Parc 1979a). Koska MVC:n historia ja suurin osa MVC:n alkuperäisistä julkaisuista pohjautuvat Smalltalk-ohjelmointikieleen, esitellään myös tässä tutkielmassa MVC:n totetusta Smalltalkilla. Tämä ei kuitenkaan rajoita tarkastelua, koska arkkitehtuurin idea pysyy täysin samana riippumatta ohjelmointikielestä.

5.2 Malli (Model)

Malli pitää yllä sovelluksen tilaa sekä vastaa sovelluksen tallentamasta datasta. Se voi olla esimerkiksi kokonaislukumuuttuja laskuri-sovelluksessa, merkkijono-olio tekstinkäsittelyohjelmassa tai mikä tahansa monimutkainen olio (Krasner 1988b, s. 3). Kaikkein yksinkertaisimmassa tapauksessa mallin ei tarvitse kommunikoida ollenkaan ohjaimen ja näkymän kanssa, vaan toimia passiivisena säiliönä datalle. Tällaisesta tilanteesta on hyvä esimerkki yksinkertainen tekstieditori, jossa teksti nähdään juuri sellaisena kuin se olisi paperilla. Tässä tapauksessa mallin ei tarvitse ottaa vastuuta kommunikoinnista näkymälle, koska muutokset tekstiin tapahtuvat käyttäjän pyynnöstä. Tällöin ohjain ottaa vastaan käyttäjän syötteet ja voi esimerkiksi ilmoittaa näkymälle muutoksesta, jolloin näkymä päivittää mallin. Ohjain voi myös päivittää mallin ja ilmoittaa tästä näkymälle, jolloin näkymä voi pyytää mallin sen hetkistä tilaa. Kummassakaan tapauksesssa mallin ei tarvitse tietää ohjaimen ja näkymän olemassaolosta (Steve 1992).

Malli ei kuitenkaan aina voi olla täysin passiivinen. Se voi myös muuttua ilman, että se tarvitsee ohjaimen tai näkymän käskyä. Otetaan esimerkiksi malli, joka muuttaa tilaansa satunnaisin väliajoina. Koska malli muuttaa itseään, täytyy sillä olla jokin yhteys näkymään, jotta se voi antaa tiedon muutoksestaan (Steve 1992). Datan kapseloinnin ja ohjelmakoodin uudelleen käytön kannalta ei ole kuitenkaan järkevää, että malli on suoraan yhteydessä näkymään ja ohjaimeen. Ohjaimen ja näkymän tulee siis olla riippuvaisia mallista, mutta ei toisinpäin. Näin mahdollistetaan myös se, että mallilla voi olla useita näkymiä ja ohjaimia (Krasner 1988b, s. 4).

Yleensä mallin tila muuttuu ohjaimista tulleiden käskyjen kautta. Tämän muutoksen tulisi heijastua kaikkiin näkymiin, jotka ovat sidottuja malliin. Tällaisia tilanteita varten kehitettiin

riippuvuudet (*dependents*). Riippuvuuksilla tarkoitetaan listaa niistä ohjaimista ja näkymistä, jotka ovat sidottuja malliin. Mallilla tulee siis olla lista riippuvuuksista ohjaimiin ja näkymiin sekä myös kyky lisätä ja poistaa niitä. Malli ei siis tiedä mitään yksittäisistä riippuvuuksista, mutta pystyy kuitenkin lähettämään itsestään muutosviestejä (*change messages*) listassa oleville ohjaimille ja näkymille. Mallin tuottamat muutosviestit voivat olla minkä tyyppisiä tahansa, joten ohjaimet ja näkymät reagoivat niihin omalla määritellyllä tavallaan (Krasner 1988a, s.2-3).

Mallille määritellään pääluokka *Model* ja tälle viitemuuttuja *dependents*, joka viittaa yhteen riippuvaan komponenttiin tai listaan riippuvista komponenteista. Kaikki uudet mallit tulee periä niiden pääluokasta, jotta saavutetaan sama toiminnallisuus kaikkiin mallikomponentteihin. Komponenttien tieto mallin muutoksista tukeutuu täysin mallin riippuvuusmekanismiin. Kun jokin komponentti luodaan, se rekisteröi itsensä malliin riippuvuudeksi ja samalla tavalla se myös poistaa itsensä (Steve 1992). Näkymät käyttävät riippuuvuusmekanismia päivittääkseen itsensä mallin muutoksien perusteella. Esimerkiksi mallin muuttuessa lähetätetään *changed*, jonka pohjalta jokainen riippuvuus saa *update* -viestin.Viestillä voi olla myös erilaisia parametrejä, joiden perusteella viestiä pystytään tarkentamaan. Esimerkiksi mallin, johon on liitetty useita näkymiä, ei välttämättä tarvitse lähettää kaikille näkymille viestiä muutoksestaan. Se voi välittää viestin mukana parametrina tiedon muutoksesta, jonka perusteella jokainen vastaanottaja voi päättää miten toimia (Steve 1992).

Alkuperäinen *update* -metodi on peritty *Object* -luokasta, eikä se tuolloin tee vielä yhtään mitään. Useimmilla näkymillä se on kuitenkin toteutettu näyttämään näkymä uudestaan kutsuttaessa. Tämä *changed/update* -mekanismi valittin toimimaan kommunikaatiokanavana mallien ja näkymien välille, koska se aiheuttaa vähiten rajoituksia ja esteitä (Steve 1992). Burbeck esittelee myös

5.2.1 Näkymä (View)

Näkymän tehtävänä on huolehtia graafisesta puolesta MVC:ssä. Näkymä pyytää yleensä mallilta datan ja tämän pohjalta näyttää käyttäjälle käyttöliittymän sovellukseen. Toisinkuin malli, jota pystytään rajoittamattomasti yhdistelemään moniin näkymiin ja ohjaimiin, jokai-

nen näkymä on liitetty yhteen ohjaimeen. Näkymä siis sisältää viitteen ohjaimeen ja ohjain sisältää viitteen näkymään. Kuten ohjain, näkymä on myös rekisteröity mallin riippuvuuksiin. Kummatkin sisältävät siis myös viitteen siihen malliin, johon ne on rekisteröity (Steve 1992). Jokaisella näkymällä on tasan yksi malli ja yksi ohjain (Krasner 1988b, s. 7).

Näkymä vastaa myös MVC-komponenttien sisäisestä kommunikaatiosta MVC-kolmikon luontivaiheessa. Näkymä rekisteröi itsensä riippuvuudeksi malliin, asettaa viitemuuttujansa viittamaan ohjaimeen ja välittää itsestään viestin ohjaimelle. Viestin avulla ohjain rekisteröi näkymän omaan viitemuuttujaansa. Näkymällä on myös vastuu poistaa viitteet sekä rekisteröinnit (Steve 1992).

Näkymä ei sisällä ainoastaan komponentteja datan näyttämiseen ruudulla, vaan se voi sisältää myös useita alanäkymiä (*subviews*) ja ylänäkymiä (*superviews*). Tästä muodostuu hierarkia, jossa ylänäkymä hoitaa aina jonkun suuremman kokonaisuuden, kuten esimerkiksi näytön pääikkunan. Alanäkymä taas huolehtii jostain pienemmästä yksityiskohdasta pääikkunassa. Näkymillä on myös viite erilliseen transformaatioluokkaan, joka hoitaa kuvan sovittamisen ja yhdistämisen alanäkymien ja ylänäkymien välillä. Jokaisella näkymällä tulee siis olla toteutus, jolla hoidetaan alanäkymien poistaminen sekä lisääminen. Samalla tulee määritellä ominaisuus, jolla sisäiset transformaatiot tuodaan transformaatioluokalle. Tämä helpottaa näkymän ja sen alanäkymien yhdistämistä (Krasner 1988b, s. 8). Burbeck havainnollistaa Smalltalkilla kirjoitetulla esimerkillä kuinka MVC-kolmikko luodaan. Esitetyssä esimerkissä on yksinkertaistettu versio MVC-kolmikon luonnista siten, että mukana on myös ylä- ja alanäkymien toteutus.

```
1  openListBrowserOn: aCollection label: labelString initialSelection: sel
2    "Create and schedule a Method List browser for
3    the methods in aCollection."
4    | topView aBrowser |
5    aBrowser ← MethodListBrowser new on: aCollection.
6    topView ← BrowserView new.
7    topView model: aBrowser; controller: StandardSystemController new;
8                        label: labelString asString; minimumSize: 300@100.
9  topView addSubView:
```

```
10      (SelectionInListView on: aBrowser printItems: false oneItem: false
11      aspect: #methodName change: #methodName: list: #methodList
12      menu: #methodMenu initialSelection: #methodName)
13      in: (0@0 extent: 1.0@0.25) borderWidth: 1.
14  topView addSubView:
15      (CodeView on: aBrowser aspect: #text change: #acceptText:from:
16      menu: #textMenu initialSelection: sel)
17      in: (0@0.25 extent: 1@0.75) borderWidth: 1.
18      topView controller open
```

Seuraavaksi käydään rivi kerrallaan läpi mitä yllä esitetyssä ohjelmakoodissa tapahtuu. Mallin luonnin jälkeen [5] luodaan viite uudelle *BrowserView* -luokan instanssille [6]. *BrowserView* on peritty *StandardSystemView* -luokasta. Seuraavaksi määritellään malli ja ohjain sekä muuttujat näkymän otsikolle ja koolle [7]. Jos ohjainta ei määritellä erikseen, käytetään näkymän *defaultController* metodia. Riveillä [7-11] luodaan alanäkymä *SelectionInListView* ja riveillä [12-15] luodaan toinen alanäkymä *CodeView*. Lopuksi [16] avataan ohjain, joka käynnistää ikkunoiden piirtämisprosessin.

Näkymät saattavat tarvita myös oman protokollan itsensä näyttämiseen. Kun malli ilmoittaa muutoksestaan, *update* -metodi näkymässä kutsuu *display*, joka puolestaan kutsuu *displayBorder*, *displayView* ja *displaySubviews*. Jos näkymä tarvitsee erityistä käyttäytymistä itsensä näyttämiseen, se toteutetaan edellämainituissa metodeissa. Muuten käytetään pääluokasta perittyjä ominaisuuksia (Steve 1992). Monet näkymät käyttävät myös erilaisia transformaatioinstansseja, joilla hallitaan esimerkiksi näkymän skaalausta ruudulla. Tähän ei kuitenkaan perehdytä sen enempää, koska ne menevät tutkimuksen rajojen ulkopuolelle.

5.2.2 Ohjain (Controller)

Ohjaimen tehtävänä on ottaa vastaan syötteitä sekä koordinoida malleja ja näkymiä saatujen syötteiden perusteella. Sen tulee myös kommunikoida muiden ohjaimien kanssa. Teknisesti ohjaimessa on kolme viitemuuttujaa: malli, näkymä ja sensori (sensor). Sensorin tehtävänä on toimia rajapintana syötelaitteiden sekä ohjaimen välillä. Sensori mallintaa syötelaitteiden

käyttäytymistä ja muuttaa ne ohjaimen ymmärtämään muotoon.

Ohjaimien tulee käyttäytyä siten, että vain yksi ohjain ottaa vastaan syötteitä kerrallaan. Esimerkiksi näkymät pystyvät esittämään informaatiota rinnakkain monen näkymän kautta, mutta käyttäjän toimintoja tulkitsee aina vain yksi ohjain. Ohjain on siis määritelty käyttäytymään siten, että se osaa tietyn signaalin perusteella päättää tuleeko sen aktivoida itsensä vai ei. Teknisesti ohjaimen käyttäytymisen määrittelee seuraavat metodit, joiden avulla ohjaimet viestivät (Krasner 1988b, s. 9):

isControlWanted - Tuleeko ohjaimen ottaa hallinta.
isControlActive - Onko ohjain aktiivinen.
controlToNextLevel - Luovutetaan hallinta seuraavalle ohjaimelle.
viewHasCursor - Onko ohjaimen näkymässä hiiren kursori.
controlInitialize - Kun ohjain on saanut hallinnan, alustetaan se.
controlLoop - Lähettää *controlActivity* -viesteja niin kauan, kuin ohjaimella on hallinta.
controlTerminate - Lopettaa ohjaimen hallinnan.

Kun ohjain saa hallinnan itselleen, kutsuu se *startUp* -metodia, joka puolestaan kutsuu seuraavia metodeja: *controlInitialize*, *controlLoop* ja *controlTerminate*. Metodit voidaan ylikirjoittaa, jolloin saavutetaan jokin haluttu ominaisuus kyseisessä vaiheessa. Esimerkiksi *controlInitialize* ja *controlTerminate* määräävät mitä tehdään, kun ohjain saa hallinnan tai luovuttaa sen eteenpäin. Ohjaimen hallinnan aikana kutsutaan *controlLoop* -metodia, joka taas kutsuu *controlActivity* -metodia niin kauan kuin ohjaimella on hallinta. Metodi *controlActivity* määrää ohjaimen toiminnan hallinnan aikana **Krasner:desc**

5.3 Esimerkkiohjelma

6 Web-sovelluskehykset

6.1 Tausta & Teoria

6.2 Pyramid

6.3 Django

6.4 Plone

6.5 Flask

7 MVC:n toteutus web-sovelluskehyksissä

7.1 Yleistä

7.2 Pyramid

7.3 Django

7.4 Plone

8 MVC and Flask

8.1 Esimerkkiohjelma

8.2 Pohdinta

8.3 Yhteenveto

9 Johtopäätökset

10 Tutkielman rakenne

10.1 Teoriaosa

10.2 Teorian jälkeen

11 Lähteiden käyttö

11.1 Lähdeviittaukset

Tämä tehdään komennolla \parencites, jolle annetaan kutakin lähdettä kohti samat argumentit kuin yksittäiselle \parencite-komennolle. Komento on hyvä (mutta ei pakko) päättää \relax-komentoon, jotta yllätyksiltä vältyttäisiin.

```
\parencites%
  [ks.][luku~3.7]{biblatex-manual}%
  [ks.~lähteiden käytöstä yleisesti myös][luku~5.3.2]%
    {biblatex-chicago-manual}%
\relax.
```

Jos jaat \parencites-komennon usealle riville, päätä rivit kommenttimerkillä (kuten yllä), jotta tulokseen ei ilmaantuisi ylimääräisiä välilyöntejä.

11.2 Lähdetietokanta

Lähteet lisätään erilliseen BIBTEX-tiedostomuodossa olevaan lähdetietokantaan. Sen laatimisessa voit käyttää apuna monia lähteidenhallintajärjestelmiä, mutta sen voi laatia myös käsin. Tietokannan nimi kirjoitetaan \addbibresource-komennon argumentiksi.

BIBTEX-muotoinen lähdetietokanta on erityisellä tavalla muotoiltu tekstitiedosto. Se koostuu tietueista, jotka alkavat @-merkillä ja sitä seuraavalla tietuetyypin nimellä. Muu osa tietueesta kirjoitetaan aaltosulkeiden sisään. Esimerkiksi edellä mainittu kääntäjäkirja (Aho ym. 2007) voidaan esittää seuraavanlaisena tietueena:

```
@Book{aho-compilers,
  author =       {Alfred V. Aho and Monica S. Lam and Ravi Sethi and
                  Jeffrey D. Ullman},
  title =        {Compilers},
  subtitle =     {Principles, Techniques, \& Tools},
  publisher =    {Pearson Addison Wesley},
  year =         2007,
```

```
  address =      {Boston},
  edition =      2
}
```

Tämän tietueen tyyppi on book, joka tarkoittaa luonnollisestikin kirjaa. Aaltosulkeiden sisällä oleva ensimmäinen sana on tietueen koodi, jota käytetään \textcite- ja \parencite-komennoissa. Sen jälkeen tulee pilkku ja joukko nimettyjä kenttiä kuten kirjan kirjoittaja (author), nimi (title), alaotsikko (subtitle) ja julkaisija (publisher). Kenttien sisällöt laitetaan aaltosulkeisiin, tosin pelkkiä numeroita sisältävät kentät voi kirjoittaa ilmankin.

Kirjoittajien nimet kirjoitetaan tietuekenttään pääosin täysin tavanomaisella tavalla. Vaihtoehtoisesti nimi voidaan esittää myös muodossa sukunimi-pilkku-etunimi (Aho, Alfred V.), ja joissakin erityistapauksissa (esimerkiksi moniosainen väliviivaton sukunimi) se on myös pakko tehdä niin. Jos kirjoittajia on useita, heidän nimensä erotetaan sanalla and (jota ei pidä suomentaa!). Jos kaikkia kirjoittajia ei luetella, laitetaan viimeisen nimen perään (ilman lainausmerkkejä) ”and others”.

Jos lähteen tekijäksi on merkitty jokin organisaatio, sen nimi pitää kirjoittaa ylimääräisiin aaltosulkeisiin (esim. Unicode Consortium 2012):

```
@Book{unicode620,
  author =       {{Unicode Consortium}},
  title =        {The Unicode Standard, Version 6.2.0},
  year =         {2012},
  url =          {http://www.unicode.org/versions/Unicode6.2.0/},
  urldate =      {2013-01-29}
}
```

Jos lähteellä ei jostain syystä ole lainkaan mimettyä tekijää, tulee author-kenttä jättää kokonaan pois, jolloin lähdeviitteeseen tulee tekijän tilalle otsikko (esim. *O* 2011):

```
@Book{presidential-novel,
  title =        {O},
  subtitle =     {A Presidential Novel},
  publisher =    {Simon \& Schuster},
  year =         {2011},
}
```

Tieteellinen lehtiartikkeli (esim. Strachey 2000) kirjoitetaan esimerkiksi seuraavanlaiseksi tietueeksi:

```
@Article{strachey-fundamentals,
  author =       {Christopher Strachey},
  title =        {Fundamental Concepts in Programming Languages},
  journal =      {Higher-Order and Symbolic Computation},
  year =         2000,
  volume =       13,
  number =       {1--2},
  pages =        {11--49},
  doi =          {10.1023/A:1010000313106}
}
```

Huomaa erityisesti kenttä doi, johon voi kirjoittaa artikkelin digitaalisen tunnisteen (Digital Object Identifier, DOI). Se on yleensä parempi valinta kuin mikään URL, koska DOI on pysyvä artikkelin tunnistetieto. Useimmat DOIt on lisäksi muutettavissa URLiksi lisäämällä sen alkuun `http://dx.doi.org/`.

Jos netissä olevan lähteen DOI ei ole tiedossa (tai sitä ei ole lainkaan), voi käyttää url-kenttää ja sen kaverina urldate-kenttää, jolla ilmaistaan (muodossa VVVV–KK–PP) verkossa olevan lähteen viittauspäivä. Linkki kannattaa valita huolella siten, että se on mahdollisimman tarkka ja mahdollisimman pitkään voimassa – jos sivulla on erikseen osoitettu pysyvä linkki (engl. *permanent link*), sitä on syytä käyttää.

Viitattaessa WWW-sivuun, joka ei ole kirja tai artikkeli tai muukaan julkaisu, voidaan käyttää online-tietuetyyppiä (esim. “Debian Social Contract” 2004):

```
@Online{debian-social-contract,
  title =        {Debian Social Contract},
  year =         {2004},
  url =          {http://www.debian.org/social_contract.en.html},
  urldate =      {2013-01-29}
}
```

Jotkin lähteet ovat toimitettuja kokoomateoksia, jotka koostuvat itsenäisistä artikkeleista. Yleensä tällöin viitataan johonkin sen osa-artikkeliin (esim. Prechelt ja Petre 2011) eikä ko-

ko kokoomateokseen. Tällöin sekä teos että viitatut artikkelit lisätään tietokantaan omina tietueinaan, ja kussakin artikkelitietueessa viitataan kokoomateokseen käyttäen crossref-kenttää:[1]

```
@Collection{making-software,
  editor =       {Andy Oram and Greg Wilson},
  title =        {Making Software},
  subtitle =     {What Really Works, and Why We Believe It},
  publisher =    {O'Reilly},
  year =         2011
}
@InCollection{prechelt-credibility,
  author =       {Lutz Prechelt and Marian Petre},
  title =        {Credibility, or Why Should I Insist on Being
                  Convinced},
  crossref =     {making-software},
  pages =        {17--34}
}
```

Huomaa, kuinka kokoomateoksella on toimittajia (editor) eikä tekijöitä (author).

Tarkempia tietoja lähdetietokannan rakenteesta löytyy BIBTEXin manuaalista (Patashnik 1988), BIBLATEXin manuaalista (**biblatex-manual**) sekä BIBLATEX-Chicagon manuaalista (**biblatex-chicago-manual**). Lisää esimerkkejä löydät myös tämän oppaan lähdekoodista.

11.3 Lähdeluettelo

Lähdetietokanta muutetaan lähdeluetteloksi apuohjelmalla biber. Se on varsin uusi, joten se puuttuu useimmista koneista, joiden TEX-asennus ei ole aivan ajantasalla. Yliopiston suorakäyttökoneista se löytyy tällä hetkellä vain charra.it.jyu.fi-koneesta. Ubuntu-asennuksiin se on saatavissa versiosta 12.10 alkaen ja Debian-asennuksiin Wheezystä alkaen. Windowsiin se on asennettavissa MikTEX-pakettina miktex-biber-bin.[2]

1. Sallittua on myös yhdistää artikkeli ja kokoomateos yhdeksi InCollection-tietueeksi, esimerkiksi jos kokoomateoksesta viitataan vain yhteen artikkeeliin. Tällöin kokoomateoksen nimi tulee booktitle-kenttään eikä crossref-kenttää käytetä.

2. Valitettavasti tämä paketti on tällä hetkellä saatavissa vain 32-bittiseen MikTEXiin.

Komentoriviltä biberin käyttö on yksinkertaista. Kun LaTeX (tai pdfLaTeX) on kerran ajettu, ajetaan biber parametrinaan dokumentin nimi. Tämän jälkeen ajetaan LaTeX (tai pdfLaTeX) vähintään kerran (kunnes edellisen ajon lopussa ei enää pyydetä uutta ajoa). Esimerkiksi näin:

```
$ pdflatex malliopas
[...]
Package biblatex Warning: Please (re)run Biber on the file:
(biblatex)                malliopas
(biblatex)                and rerun LaTeX afterwards.
[..]
Output written on malliopas.pdf (18 pages, 96855 bytes).
Transcript written on malliopas.log.
$ biber malliopas
INFO - This is Biber 0.9.9
[...]
INFO - Output to malliopas.bbl
$ pdflatex malliopas
[...]
LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.
[...]
Output written on malliopas.pdf (21 pages, 107373 bytes).
Transcript written on malliopas.log.
$ pdflatex malliopas
[...]
Output written on malliopas.pdf (21 pages, 107509 bytes).
Transcript written on malliopas.log.
```

11.4 Tiedossa olevat ongelmat

Lähdeluettelon ja lähdeviitteiden toiminta ei ole toistaiseksi aivan virheetöntä.

Jos artikkelilla ei ole tekijää, lähdeluettelossa kyseisen artikkelin merkintä alkaa vuosiluvulla. Tämä vika on korjattu BibLaTeX-Chicagon versiossa 0.9.9c (julkaistu 15.3.2013).

Jos lähdetietokantaan kirjoittaa urldate-päiväyksen, tulee se lähdeluetteloon virheellisessä

muodossa. Tämä vika on korjattu BIBLATEX-Chicagon versiossa 0.9.9b (julkaistu 6.12.2012).

12 Tutkielmapohjan erityispiirteet

Pääsääntöisesti gradu3 käyttäytyy kuten LaTeXin mukana tuleva report-kirjoitelmaluokka. Eroja kuitenkin on:

- Sinun ei tarvitse ladata inputenc-, fontenc- eikä babel-pakettia.
  - Käyttämäsi merkistö sinun pitää ilmoittaa \documentclass-komennon optiona. Nykyään utf8 on yleensä sopiva valinta, joskin joissakin tilanteissa latin1 tai latin9 voi tulla myös kyseeseen.
  - Jos tutkielmasi on englanninkielinen, ilmoita se \documentclass-komennon optiolla english.
- Jos tutkielmasi on kandidaatintutkielma, käytä \documentclass-komennon optiota bachelor.
- Ilmoita tutkielmasi metatiedot taulukossa 1 esitetyillä komennoilla. Ne tulee antaa ennen \maketitle-komentoa.
- Voit \maketitle-komennon jälkeen halutessasi kirjoittaa esipuheen. Sen otsikon saat komennolla \preface.
- Mahdollisen esipuheen jälkeen voit kirjoittaa termiluettelon käyttämällä thetermlist-ympäristöä. Sen sisällä voit käyttää \item[*termi*]-komentoa merkitsemään määriteltävän termin.
- Käytä \maketitle-komennon ja mahdollisten esipuheen ja termiluettelon jälkeen \mainmatter-komentoa. Se laatii automaattisesti tarvittavat sisällys-, kuvio- ja taulukkoluettelot.
- Komentoja \subsubsection, \paragraph ja \subparagraph ei tueta.
- Liitteet eivät ole lukuja (\chapter) vaan alilukuja (\section).
- Lähdeluettelon ja lähdeviitteiden tekemisestä kerrottiin edellisessä luvussa.

Komento	Tarkoitus
\title	Työn otsikko (älä käytä \thanks-komentoa)
\translatedtitle	Suomenkielisen työn englanninkielinen otsikko, englanninkielisen työn suomenkielinen otsikko
\studyline	Suuntautumisvaihtoehtosi
\tiivistelma	Suomenkielinen tiivistelmä
\abstract	Englanninkielinen abstrakti
\avainsanat	Suomenkieliset avainsanat
\keywords	Englanninkieliset avainsanat
\author	Kirjoittajan nimi (jos useita, anna kukin omana komentonaan – \and-komentoa ei tueta)
\contactinformation	Kirjoittajan yhteystiedot
\supervisor	Tutkielman ohjaaja (jos useita, anna kukin omana komentonaan)

Taulukko 1. Metatietojen ilmoituskomennot

13 Yhteenveto

Tutkielman viimeinen luku on Yhteenveto. Sen on hyvä olla lyhyt; siinä todetaan, mitä tutkielmassa esitetyn nojalla voidaan sanoa johdannon väitteen totuudesta tai tutkimuskysymyksen vastauksesta. Yhteenvedossa tuodaan myös esille tutkielman heikkoudet (erityisesti tekijät, jotka heikentävät tutkielman tulosten luotettavuutta), ellei niitä ole jo aiemmin tuotu esiin esimerkiksi Pohdinta-luvussa. Tässä luvussa voidaan myös tuoda esille, mitä tutkimusta olisi tämän tutkielman tulosten valossa syytä tehdä seuraavaksi.

Jos Yhteenveto alkaa pitkittyä, se kannattaa jakaa kahtia niin, että tulosten tulkinta otetaan omaksi Pohdinta-luvukseen, jolloin Yhteenvedosta tulee varsin lyhyt ja lakoninen.

Yhteenvedon jälkeen tulee \printbibliography-komennolla laadittu lähdeluettelo ja sen jälkeen mahdolliset liitteet.

Lähteet

Aho, Alfred V., Monica S. Lam, Ravi Sethi ja Jeffrey D. Ullman. 2007. *Compilers: Principles, Techniques, & Tools.* 2. painos. Boston: Pearson Addison Wesley.

Avraham, Avraham Leff James T. Rayfield. 2001. "Web-Application Development Using the Model/View/Controller Design Pattern: 1.2, Web.Applications and the MVC Design". http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=950428.

Bestframeworks. 2009. *List of Python MVC-frameworks by bestwebframeworks.com.* http://www.bestwebframeworks.com/compare-web-frameworks/python/.

Daly, Liza. 2007. *Next-Generation Web Frameworks in Python.* O'Reilly Media.

Deacon, John. 2009. "Computer Systems Development, Consulting and Training Model-View-Controller (MVC) Architecture".

"Debian Social Contract". 2004. Viitattu 29. tammikuuta 2013. http://www.debian.org/social_contract.en.html.

Django, Software Foundation. 2005. *FAQ: General.* Django Software Foundation.

Fernández-Villamor, José Ignacio. 2008. "A comparison model for agile web frameworks". delivery.acm.org/10.1145/1630000/1621101/a14-ignacio.pdf.

Kourie, Iwan Vosloo & Derrick G. 2008. "Server-Centric Web Frameworks: An Overview". http://dl.acm.org/citation.cfm?id=1348246.1348247&coll=DL&dl=ACM&CFID=509430230&CFTOKEN=54230385.

Krasner, Glenn E. Krasner & StephenT.Pope. 1988a. "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80". http://www.ics.uci.edu/~redmiles/ics227-SQ04/papers/KrasnerPope88.pdf.

———. 1988b. "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System". http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf.

O: A Presidential Novel. 2011. Simon & Schuster.

Ockanovic, T. Mateljan, V. Okanovic'. 2011. "Designing a New Web Application Framework". http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5967262.

Parc, Xerox. 1978. "MVC Xerox Parc". http://heim.ifi.uio.no/trygver/themes/mvc/mvc-index.html.

———. 1979a. "Xerox Parc THE Original MVC reports". http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf.

———. 1979b. "Xerox Parc THING-MODEL-VIEW-EDITOR". http://heim.ifi.uio.no/trygver/1979/mvc-1/1979-05-MVC.pdf.

Patashnik, Oren. 1988. BIBTEX*ing.* 8. helmikuuta. Viitattu 29. tammikuuta 2013. http://mirror.ctan.org/biblio/bibtex/base/btxdoc.pdf.

Prechelt, Lutz, ja Marian Petre. 2011. "Credibility, or Why Should I Insist on Being Convinced". Teoksessa *Making Software: What Really Works, and Why We Believe It,* toimittanut Andy Oram ja Greg Wilson, 17–34. O'Reilly.

Pyramid. 2010. *Pyramid introduction.* http://www.kemeneur.com/clients/pylons/docs/pyramid/narr/introduction.html.

Reenskaug, Trygve. 2003. "The Model-View-Controller (MVC) Its Past and Present". http://heim.ifi.%20uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf.

Steve, Burbeck. 1992. "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)". http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Strachey, Christopher. 2000. "Fundamental Concepts in Programming Languages". *Higher-Order and Symbolic Computation* 13 (1–2): 11–49. doi:10.1023/A:1010000313106.

Unicode Consortium. 2012. *The Unicode Standard, Version 6.2.0.* Viitattu 29. tammikuuta 2013. http://www.unicode.org/versions/Unicode6.2.0/.

Liitteet

A Siirtyminen gradu2:sta gradu3:een

Keskeneräisen tutkielman siirtäminen gradu2:sta gradu3:een ei ole kovin vaikeata. Aluksi on totta kai vaihdettava \documentclass-komennossa gradu2 gradu3:ksi. Komennon optioista suurin osa on poistettava, koska niitä ei enää tueta; ainoastaan merkistön ilmoittava optio jää jäljelle. Mahdollinen kandi-optio vaihdetaan optioksi bachelor.

Taulukossa 2 on lueteltu tarvittavat komentovaihdokset. Viiva tarkoittaa, ettei vastaavaa komentoa ole lainkaan. Huomaa erityisesti uudet komennot.

gradu2	gradu3
—	\maketitle
—	\supervisor
\acmccs	—
\aine	\subject
\copyrightowner	—
\fulltitle	—
\laitos	\department
\license	—
\linja	\studyline
\paikka	—
\setauthor	\author
\termlist	thetermlist-ympäristö
\tyyppi	\type
\yhteystiedot	\contactinformation
\yliopisto	\university
\ysa	—

Taulukko 2. Komentomuutokset gradu2:sta gradu3:een

Isoin työ voi aiheutua lähdeluettelon laatimistekniikan muuttumiseen sopeutumisesta.

B Harvemmin tarvittavat ominaisuudet

Aiemmin esiteltyjen lisäksi gradu3 tarjoaa seuraavat lisäominaisuudet:

- LaTeX 2_ε:n vakio-optiot draft ja final toimivat.
- Vaikka tutkielman suomenkielisyyttä ei tarvitse erikseen mainita, finnish-optio toimii.
- \university-komennolla voit ilmoittaa tutkielman kotiyliopistoksi jonkin muun kuin Jyväskylän yliopiston.
- \department-komennolla voit ilmoittaa tutkielman kotilaitokseksi jonkin muun kuin Tietotekniikan laitoksen.
- \subject-komennolla voit ilmoittaa tutkielman oppiaineeksi jonkin muun kuin tietotekniikan. Huomaa, että oppiaine tulisi suomenkielisissä tutkielmissa kirjoittaa genetiivimuodossa ja isolla alkukirjaimella (”Tietotekniikan”), englanninkielisissä tuktkielmissa in-preposition kanssa (”in Information Technology”).
- \type-komennolla voit ilmoittaa tutkielman tyypin, jos se on jokin muu kuin pro gradu (oletus) tai kandidaatintutkielma (optiolla bachelor).
- \setdate-komennolla voit asettaa päivämäärän haluamaksesi. Anna komennolle kolme parametria – päivä, kuukausi ja vuosi – numeerisessa muodossa.
- Ympäristöllä chapterquote voit laittaa luvun alkuun mietelauseen. Sillä on yksi pakollinen parametri (lainauksen attribuutio).
- Komento \graduclsdate sisältää käytössä olevan gradu3:n julkaisupäivämäärän ja \graduclsversion sen versionumeron.