Muokattavien käyttöliittymien toteutus

Jani Lindgren

1.7.2004

Joensuun Yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Pro gradu -tutkielmassani esittelen toiminnallisen ytimen rakenteen sovellukselle, joka sallii käyttäjän määrätä vapaasti sovelluksen käyttöliittymän mieleisekseen. Käyttäjä määrittelee sovelluksen käyttöliittymän merkkauskielen ja ohjelmointikielen avulla. Merkkauskielestä viitataan ohjelmointikielellä tehtyihin määrittelyihin. Ohjelmointikielestä viitataan merkkauskielellä määriteltyihin komponentteihin. Tämän vuoksi kielet tulee kehittää tukemaan toisiaan.

Käyttäjä määrittelee sovelluksen käyttöliittymän rakenteen ja ulkoasun merkkauskielen avulla. Merkkauskieli koostuu merkitsimistä, joiden avulla käyttäjä määrittelee käyttöliittymässä esiintyvien komponenttien ominaisuudet. Esittelen käyttöliittymien määrittelyyn kehittelemäni merkkauskielen, jolla voidaan määritellä yksinkertaisia käyttöliittymiä. Merkkauskieli sisältää yleisimmin käytetyt käyttöliittymän rakenteet.

Käyttäjä määrittelee sovelluksen käyttöliittymän toiminnallisuuden ohjelmointikielen avulla. Keskeisessä asemassa on ajatus, jossa merkkauskielellä määriteltyyn käyttöliittymän komponenttiin kytketään toiminnallisuutta. Kytkennät esitetään aliohjelmakutsujen tasolla. Komponentti kytketään joko toiminnallisen ytimen operaatioon tai käyttäjän määrittelemään komentojonoon. Komentojonot voivat kutsua käyttöliittymäkirjaston tai toiminnallisen ytimen palveluita.

Havainnollistan ohjelman suorituksen etenemistä sovelluksessa, jossa on käyttäjän määrittelemä käyttöliittymä. Suorituksen etenemisen tarkastelu koostuu alustustoimenpiteistä ja ajonaikaisten syötteiden käsittelystä. Syötteiden käsittelyssä kantavana ajatuksena on ohjata ohjelman suoritus toiminnallisen ytimen operaatioihin tai käyttäjän määrittelemiin komentojonoihin käyttöliittymän sisäisen esitystavan perusteella. Alustustoimenpiteissä käsitellään merkkauskieliset ja ohjelmointikieliset määrittelyt, joiden pohjalta rakennetaan tarvittavat tietorakenteet syötteiden käsittelyä varten.

Avainsanat: Käyttöliittymä, sisäinen esitystapa, toiminnallinen ydin

Sisällysluettelo

1 Johdanto

Käyttöliittymien tutkimus on laajalle levinnyttä. Tutkimuksen tavoitteina ovat ohjelmiston käyttöönottamiseen liittyvä oppimiskynnyksen madaltaminen ja työtehtävien suorittaminen ohjelmistolla mahdollisimman vaivattomasti. Yleisesti käyttöliittymätutkimuksen tavoitteena on tehokkaampi työskentely.

Blom ja Monk esittävät tutkimuksen [3] lähtökohdaksi käyttäjien mielenlaadun. Eräs käyttäjän mielenlaatuun vaikuttava tekijä on tiheys, jolla sovellusta käytetään. Tässä tapauksessa käyttöliittymän muuttaminen johtaa tilanteeseen, jossa sovellus on vaivattomampi käyttää. Tässä pro gradu -tutkielmassa lähtökohtana käyttöliittymien muuttamiselle on työtehtävien yksinkertaistaminen.

Schiaffino ja Amandi ovat tutkimuksessaan [9] perehtyneet käyttöliittymäagenttien toimintaan. Käyttöliittymäagentit ovat ohjelmia, jotka oppivat käyttäjän mieltymykset ja työskentelytavat sekä keskeyttävät käyttäjän antaessaan ohjeita, varoituksia ja korjausehdotuksia. Agenttien toiminta edellyttää tietojen keräystä vuorovaikutuksesta käyttäjän kanssa. Kerätyt tiedot taltioidaan käyttäjäprofiiliin. Siihen taltioidaan henkilökohtaisten tietojen lisäksi mm. mielenkiinnon kohteet, käyttäytymistoistuvuudet ja tavoitteet. Tutkimuksessa on keskitytty käyttäjien kokemuksiin tilanteissa, jossa he ovat olleet vuorovaikutuksessa käyttöliittymäagentin kanssa. Microsoftin tuotteissa on esiintynyt käyttöliittymäagentteja, jotka käyttäjien antaman palautteen vuoksi eivät ole oletusarvoisesti toiminnassa ohjelmistotalon tuotteiden uusimmissa versioissa.

Tässä pro gradu -tutkielmassa esitellään merkkaus- ja ohjelmointikieli, joilla käyttöliittymä on mukautettavissa vastamaan käyttäjän tarpeita. Tämä lähestymistapa johtaa käyttöliittymän ja sovelluksen lähdekoodien erottamiseen toisistaan. Merkkaus- ja ohjelmointikielellä laadittu käyttöliittymän määrittely on miellettävissä käyttäjäprofiiliksi. Tällainen profiili on laadittava ennen kuin käyttäjä aloittaa sovelluksen käytön.

Chiron-1 on arkkitehtuuri, joka esittää tavan ohjelman käyttöliittymän toteuttamiselle asiakas-palvelin -ratkaisuissa [10]. Arkkitehtuuri mahdollistaa käyttöliittymän ja sovelluksen lähdekoodin erottamisen toisistaan. Chiron-1 -arkkitehtuurissa tämä saadaan aikaiseksi hyödyntämällä *käyttöliittymäagentteja* (*interface agents*), joita kutsutaan *artisteiksi* (*artists*). Artisti on vastuussa abstraktin tietotyypin esittämisestä käyttäjälle ja vuorovaikutuksesta käyttäjän kanssa. Abstrakti tietotyyppi on

tässä yhteydessä nimi sovelluksen käsittelemälle tiedolle. Yhteen abstraktiin tietotyyppiin voi olla sidottuna yksi tai useampi artisti.

Chiron-1 -asiakasohjelma muodostuu sovelluksesta ja käyttöliittymästä. Asiakkaan käyttöliittymän koostumus määritellään ulkoisessa tiedostossa, jonka asiakasohjelma käsittelee alustustoimenpiteenä. Käyttöliittymän koostumus on muunneltavissa ilman, että asiakasohjelman lähdekoodi käännetään. En käsittele käyttöliittymän muuntamista asiakas-palvelin -ratkaisuissa. Jos pro gradu -tutkielmani sisältö suhteutetaan Chiron-1 -arkkitehtuuriin, käsittelen pro gradu -tutkielmassani asiakasohjelmaa.

Chiron-1 -arkkitehtuurissa sidos käyttöliittymän ja sovelluksen välille esitetään abstraktien tietotyypin tasolla. Pro gradu -tutkielmassani esitän sidoksen aliohjelmakutsujen tasolla. Aliohjelmien kutsut voidaan mieltää abstraktin tietotyypin operaatioiden kutsuiksi. Chiron-1 -arkkitehtuurissa yhteen abstraktiin tietotyyppiin voi olla sidottuna samanaikaisesti useampi kuin yksi artisti. Pro gradu -tutkielmassani esittämä arkkitehtuuri tukee samaa ajatusta.

Rendezvous-arkkitehtuuri ja ohjelmointikieli ovat suunniteltu tukemaan maantieteellisesti kaukana toisinaan sijaitsevien ihmisten vuorovaikutusta [7]. On vaikeaa toteuttaa sovelluksia, jotka mahdollistavat usean samanaikaisen käyttäjän muokkaavan samaa tietoa. Rendezvous-arkkitehtuuri ja ohjelmointikieli yksinkertaistavat tällaisten sovellusten toteuttamista.

Arkkitehtuuri tukee ajatusta *monen käyttäjän sovelluksesta* (*multiuser program*). Monen käyttäjän sovelluksessa usealle käyttäjälle näytetään sama tieto ja käyttäjien tulee olla mahdollista muokata esitettävää tietoa samanaikaisesti. Käyttäjille on mahdollista näyttää sama tieto, mutta täysin eri muodossa. Esimerkki Rendezvous-arkkitehtuuria hyödyntävästä sovelluksesta on jätkänshakki. Kahdelle pelaajalle näytettään sama pelitilanne, mutta sovellusten käyttöliittymät saattavat olla hyvinkin erilaisia.

Rendezvous-arkkitehtuurissa esiintyy ajatus, jota suunnittelijat kutsuvat *vuorovaikutus riippumattomuudeksi* (*dialog independence*). Tämä tarkoittaa sovelluksen jakamista kahteen osaan: käyttöliittymään ja sovellukseen. Jako tapahtuu sekä suunnittelun että ajon aikana. Suunnitteluvaiheessa käyttöliittymäasiantuntijat suunnittelevat käyttöliittymän ja ohjelmointiasiantuntijat suunnittelevat sovelluksen. Jako esiintyy myös ajonaikana, jolloin kahden suunnittelijajoukon ohjelmakoodit ovat olemassa rinnakkain. Tässä pro gradu -tutkielmassa toissijainen tavoite on ollut käyttöliittymän ja sovelluksen käyttöliittymän lähdekoodien erottaminen.

Dewan ja Choudhary [4] ovat kehittäneet mallin käyttöliittymien sitomiselle sovellukseen monen käyttäjän sovelluksen tapauksessa. Malli pohjautuu ajatukseen, jossa sovellus mielletään tiedon *toimitusohjelmaksi* (*editor of data*). Malli koostuu *semanttisesta mallista* (*semantic model*), *kuvausmallista* (*specification model*) ja *toteutusmallista* (*implementation model*). Semanttinen malli määrää esimerkiksi tilanteet, joissa käyttäjän tekemät muutokset usean käyttäjän jakamaan tietoon päivitetään muille käyttäjille. Kuvausmalli määrää tavan, jolla käyttöliittymä sidotaan jaettuihin tietoihin. Toteutusmalli ottaa kantaa tapaan, jolla käyttöliittymien sidonta jaettuun tietoon toteutetaan käytännössä.

Dewan ja Choudbary havainnollistavat mallin hyödyntämistä järjestelmässä, jonka nimi on Suite. He esittelevät tutkimuksessaan yhteneväisyyksiä kehittelemänsä mallin ja Rendevous-arkkitehtuurin välillä. Pro gradu -tutkielmani sisältö koostuu asioista, jotka vastaavat pääosin kuvausmallia ja toteutusmallia. Luvut 2-5 kuvaavat kuvausmallin. Luku 6 koostuu toteutusmallista.

Pro gradu -tutkielmani pyrkii antamaan vastauksen ohjelmistosuunnittelijan näkökulmasta siihen, millaista lähestymistapaa noudattamalla voidaan tehdä mahdolliseksi käyttäjän muokattavissa oleva käyttöliittymä. Tämän vuoksi *käyttöliittymien ylläpitojärjestelmät* (*user interface management systems*) ovat tutkimusalueena lähimpänä pro gradu -tutkielmani aihepiiriä [4, 7, 10]. Tavoitteena on käyttöliittymän muuttaminen ilman, että sovelluksen lähdekoodia käännetään. Pro gradu -tutkielmani tavoite on viedä muokattavuus tasolle, jossa sovelluksen käyttäjä voi vapaasti muokata käyttöliittymän mieluisekseen. Toissijainen tavoite on selkeyttää käyttöliittymän ja toiminnallisen ytimen välistä eroa.

Pro gradu -tutkielma pyrkii tarjoamaan vastauksia ohjelmistosuunnittelijalle, joka on syystä tai toisesta kiinnostunut muokattavissa olevan käyttöliittymän ohjelmoimisesta sovellukseen tai käyttöliittymäkirjastoon. Tutkielman tarkoitus ei ole esitellä mitä kaikkea on teoriassa mahdollista muokata sovelluksen käyttöliittymässä. Pääpaino on käytännön ohjelmoinnissa. Pro gradu -tutkielmani sisältöä tukemaan on kirjoitettu ohjelma, joka havainnollistaa eräitä tutkielmassa esiintyviä ajatuksia. Ohjelma on kirjoitettu C++ -ohjelmointikielellä.

Luku 2 perehdyttää lukijan käyttöliittymiin. Luvussa käsitellään eräiden sovellusten käyttöliittymiä, joiden pohjalta on johdettavissa tärkeimmät ajatukset käyttöliittymien rakenteesta. Luvussa annetaan esimerkkejä tilanteissa, joissa käyttöliittymän muokkaus saattaa olla mielekästä. Muiden lukujen sisältö nojautuu luvun sisältöön.

Luvussa 3 johdetaan käyttöliittymien sisäinen esitystapa. Käsiteltäville komponenteille johdetaan sisäinen esitystapa muokattavien kohteiden perusteella. Tämä lähestymistapa johtaa *puu*-rakenteeseen, joka on tapa esittää sovelluksen käyttöliittymä tietokoneen keskusmuistissa. Luvun sisältö perustuu luvussa 2 käsiteltyihin asioihin.

Luvussa 4 esitellään merkkauskieli, jolla käyttäjä määrittelee sovelluksen käyttöliittymän rakenteen ja ulkoasun. Luvussa esitelty merkkauskieli on vaihtoehtoinen tapa esittää käyttöliittymän sisäinen esitystapa. Merkkauskielinen määrittely sijaitsee massamuistissa muodossa, jota käyttäjä voi vapaasti muokata.

Luvussa 5 esitellään ohjelmointikieli, jolla määritellään käyttöliittymän toiminnallisuus. Luvussa esitelty ohjelmointikieli perustuu Pascal -ohjelmointikieleen. Ohjelmointikielellä laadittu toiminnallisuuden määrittely sijaitsee massamuistissa muodossa, jota käyttäjä voi vapaasti muokata.

Luku 6 käsittelee sovelluksen toiminnallista ydintä. Toiminnallinen ydin koostuu ohjelmakomponenteista, jotka käsitellään vuorollaan. Luvussa esitelty toiminnallisen ytimen rakenne on oma ehdotukseni. Lisäksi luku kertoo miten sovelluksen suoritus etenee käyttäjän antaessa syötteitä oheislaitteilta.

Luvussa 7 esitetään pro gradu -tutkielmani keskeinen asia uudelleen kertauksena. Lisäksi luvussa pohditaan miten pro gradu -tutkielmassani esitettyjä asioita on mielekästä kehittää eteenpäin. Luvussa esitetään myös tilanteita, joissa pro gradu -tutkielmassani esitetyistä asioista on hyötyä.

2 Käyttöliittymät

Tämän luvun tavoite on perehdyttää lukija käyttöliittymien rakenteeseen. Tavoitteeseen päästään tutkimalla esimerkkejä sovellusten käyttöliittymistä. Luvussa pyritään kuvien avulla havainnollistamaan yhteneväisyyksiä ja eroavaisuuksia käyttöliittymissä esiintyvien ratkaisujen välillä, ja esittelemään myöhemmissä luvuissa esiintyvää terminologiaa. Lukijan tulee ymmärtää käyttöliittymien rakenne ja terminologia ennen kuin myöhemmät luvut ovat ymmärrettävissä. Seuraavassa luvussa tehdään johtopäätöksiä, miten käyttöliittymien muokattavuus on toteutettavissa.

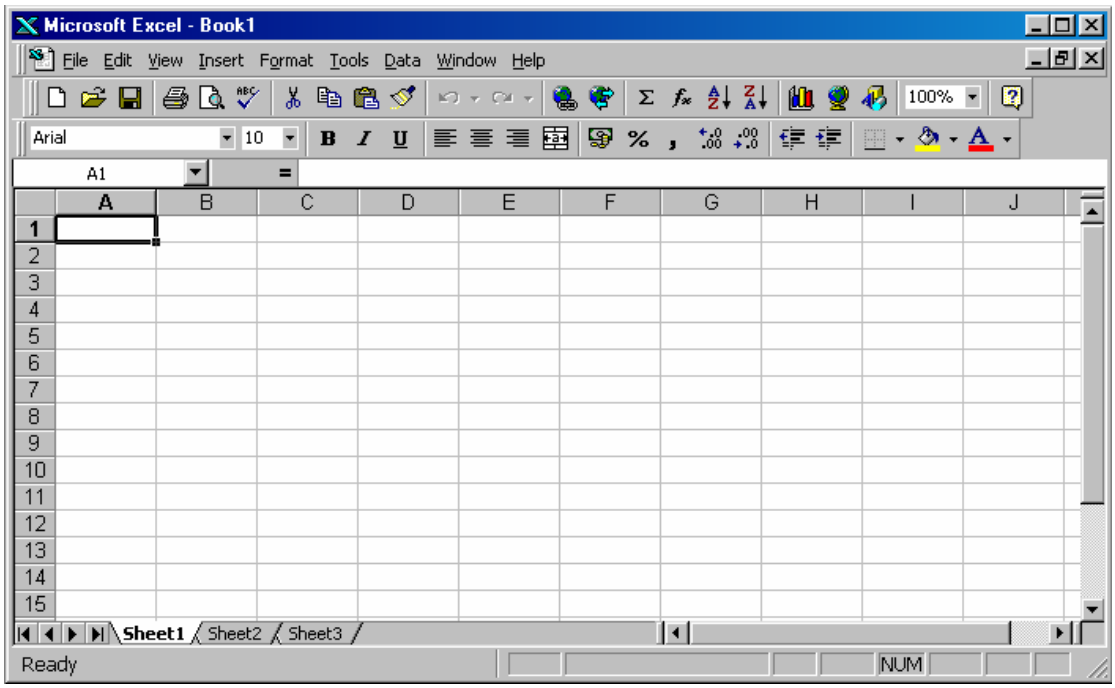
Käyttöliittymien rakenteen selvittämiseksi tutkin useiden sovellusten käyttöliittymissä esiintyviä ratkaisuja. Lähes kaikista sovelluksista on olemassa uudempikin versio, mutta tämä ei tarkoita olennaista muutosta sovelluksen käyttöliittymässä tai sen rakenteessa. Sovellusten uudemmissa versioissa on yleensä enemmän toimintoja. Tutkitut sovellukset ovat:

- Microsoft Excel 97
- Microsoft Word 97
- Microsoft PowerPoint 97
- Microsoft Visual C++ 5.0
- Borland C++ Builder 5.0
- Blender Creator
- Acrobat Reader
- Creative Mixer
- Creative PlayCenter

Edellä mainitut sovellukset jakautuvat mielestäni kolmeen eri ryhmään käyttäjäkunnan perusteella: *toimistosovellukset*, *ammattilaissovellukset* ja *multimediasovellukset*. Edellä mainittujen sovellusryhmien käyttäjillä on erilaiset tarpeet käyttöliittymän käytettävyydelle. Toimistosovelluksia käyttävät henkilöt hyödyntävät tietokoneita työssään enemmän tai vähemmän säännöllisesti. Ammattilaissovelluksia käyttävät henkilöt työskentelevät pääsääntöisesti tietokoneen ääressä. Multimediasovelluksia käyttävät lähinnä harrastajat. Tutkimalla jokaisen ryhmän sovelluksia todennäköisesti löydetään mahdollisimman paljon erilaisia ratkaisuja käyttöliittymien suunnittelussa.

2.1 Microsoft Excel

Microsoft Excel on taulukkolaskentasovellus, joka on yksi Office -toimistosovelluspaketin sovelluksista. Kuva 2.1 esittää sovelluksen käyttöliittymän. Microsoftin sovelluksien toiminnallisuus jakautuu käyttöliittymässä selkeästi neljään osaan: *valikkoriviin* (*menubar*), *työkalurivialueeseen* (*toolbar area*), *tilariviin* (*statusline*) ja *työnäkymään* (*work view*). Edellä mainitut alueet ovat esimerkkejä käyttöliittymän muodostavista *komponenteista* (*component*). Komponentilla on tiedonkäsittelyä edistävä tarkoitus käyttöliittymässä. Myös myöhemmin esiteltävien sovellusten käyttöliittymissä esiintyvät vastaavat alueet, mutta niiden sijoittelu käyttöliittymässä saattaa olla erilainen.



Kuva 2.1 Microsoft Excel -sovelluksen käyttöliittymä

2.1.1 Valikkorivi

Valikkorivi sijaitsee ikkunan ylälaidassa välittömästi otsikkorivin alapuolella. Kuvassa 2.2 on esitetty Microsoft Excel -sovelluksen valikkorivi. Valikkorivi sisältää sovelluksen päävalikon, jonka kautta lähes jokaisen sovelluksen tapauksessa on mahdollista suorittaa jokainen sovelluksen toiminnoista. Se, miten sovelluksen toiminnallisuus on jaettu *valikoihin* (*menu*), on sovelluskohtaista. Solujen muotoileminen on esimerkki sovelluskohtaisesta toiminnosta Microsoft Excel-sovelluksen tapauksessa.

Kuva 2.2 Microsoft Excel -sovelluksen valikkorivi

2.1.2 Työkalurivialue

Työkalurivialue sijaitsee välittömästi valikkorivin alapuolella. Kuvassa 2.3 on esitetty Microsoft Excel -sovelluksen työkalurivialue. Työkalurivialue sisältää yhden tai useamman *työkalurivin* (*toolbar*). Työkalurivi on ensisijaisesti oikotie useasti suoritettaviin operaatioihin. Microsoft Excel -sovelluksen tapauksessa työkalurivit ovat jaettu kahdelle riville.

Kuva 2.3 Microsoft Excel -sovelluksen työkalurivialue

Työkalurivialue voi sisältää useamman kuin yhden työkalurivin päällekkäin, kuten Microsoftin sovelluksien käyttöliittymissä on tapana. Ainakin Microsoftin sovelluksissa on mahdollista muokata työkalurivialueen sisältöä mieleisekseen raahaamalla työkalurivejä uuteen paikkaan. Sovelluksissa esiintyy toisinaan valikkoja, joista voi valita työkalurivialueella näytettävät työkalurivit.

Kuva 2.3 havainnollistaa myös miten työkalurivit koostuvat komponenteista. Kuvan työkaluriveissä esiintyy esimerkiksi *alasvetovalikkoja* (*dropdown menu*), joilla valitaan kirjasinlaji. *Tilanvalitsimilla* (*state button*) määrätään onko toiminto käytössä. Esimerkkejä tällaisista toiminnoista ovat kirjasinlajin lihavointi, kursivointi ja alleviivaus. Työkaluriveissä esiintyy myös *nappeja* (*button*), jotka suorittavat jonkin sovelluksen toiminnan kannalta mielekkään toiminnon.

2.1.3 Tilarivi

Tilarivi sijaitsee lähes poikkeuksetta ikkunan alalaidassa. Kuvassa 2.4 on esitetty Microsoft Excel -sovelluksen tilarivi. Tilarivin tarkoitus on esittää käyttäjälle tärkeää tietoa sovelluksen tilasta tai käsiteltävästä tiedosta. Tilarivissä esitettävän tiedon laatu on sovelluskohtaista. Työnäkymässä käsiteltävän tiedon tallentamisen tarve on esimerkki tilarivissä esitettävästä tiedosta.

Kuva 2.4 Microsoft Excel -sovelluksen tilarivi

2.1.4 Työnäkymä

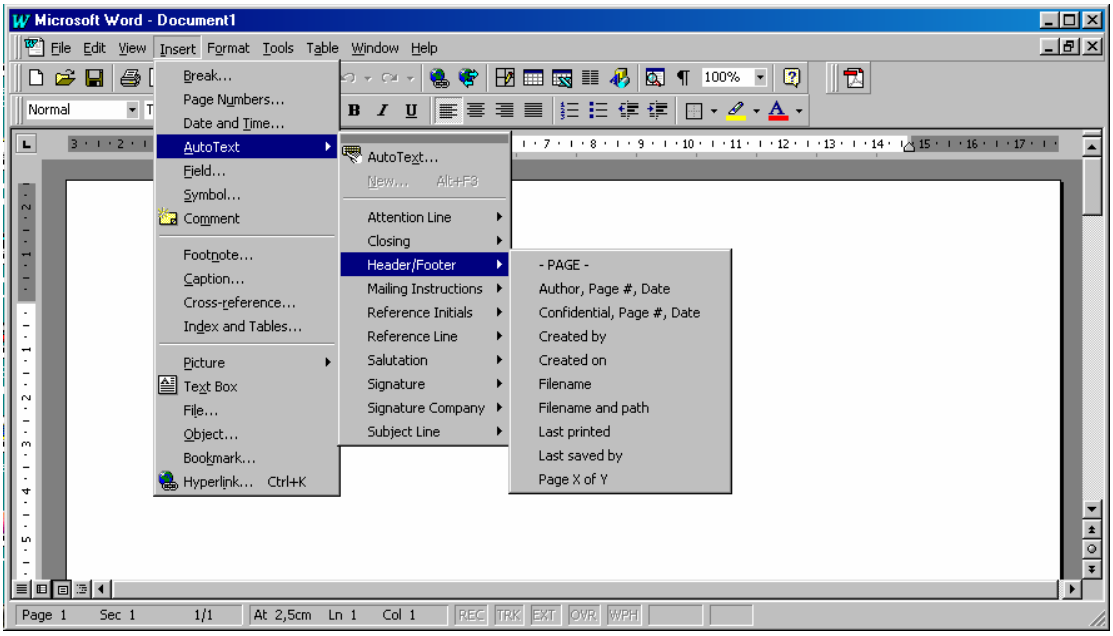
Työnäkymä on tässä yleistys sovelluskohtaiselle komponentille, joka esittää sovelluksen käsittelemää tietoa sopivassa muodossa. Seuraavat ovat esimerkkejä työnäkymistä:

- *dokumentti* (*document*) tekstinkäsittelysovelluksessa
- *taulukko* (*sheet*) taulukkolaskentasovelluksessa
- *kangas* (*canvas*) kuvankäsittelysovelluksessa

Työnäkymän ulkoasuun ja toiminnallisuuteen liittyviä ominaisuuksia on lähes mahdotonta määritellä sovelluskohtaisuuden vuoksi. Työnäkymillä on kuitenkin yksi sovelluksen käyttöliittymän rakenteeseen vaikuttava ominaisuus, nimittäin *tilavalikko* (*context menu*). Tilavalikon avulla on mahdollista suorittaa useasti käytettäviä toimenpiteitä nopeammin kuin päävalikon kautta. Jos työnäkymä havainnollistaa heterogeenistä tietoa, niin tilavalikko voi muuttua riippuen kohdasta, jossa tilavalikko avattiin. Esimerkiksi työnäkymässä, joka havainnollistaa verkon rakennetta, solmujen ja kaarien tilavalikot ovat erilaisia. Esimerkki tällaisesta verkosta on maantieverkosto.

2.2 Microsoft Word

Microsoft Word on tekstinkäsittelyohjelma. Se on yksi Office -toimistosovelluspaketin sovelluksista. Kuvassa 2.5 on esitetty sovelluksen käyttöliittymä. Myös Microsoft Word -sovelluksen käyttöliittymässä esiintyvät valikkorivi, työkalurivialue, työnäkymä ja tilarivi.



Kuva 2.5 Microsoft Word -sovelluksen käyttöliittymä

2.2.1 Valikot

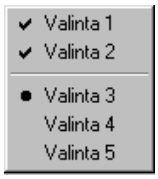
Enemmistö sovelluksen toiminnoista suoritetaan valikoiden kautta. Ennen kuin on mahdollista suorittaa toiminto valikoiden kautta, täytyy olla jokin tapa avata valikko. Kyseistä tapaa nimitetään tässä valikon *alkukohdaksi*. Microsoftin sovelluksissa ja käyttöjärjestelmissä esiintyy pääsääntöisesti kolme erilaista alkukohtaa:

- valikkorivi
- nappi
- työnäkymä

Yleensä valikon alkukohtana toimii sovelluksen ylälaidassa sijaitseva valikkorivi. Kuvassa 2.5 on havainnollistettu valikko, jonka alkukohta on valikkorivi. Selkeästi harvemmin käytetty tapa avata valikko on nappi-komponentti. Esimerkki tällaisesta valikon alkukohdasta on Windows 98 -käyttöjärjestelmän käynnistä -valikko. Yleensä työnäkymällä on tilavalikko, joka avataan painamalla hiiren oikeaa nappia työnäkymän alueella. Tilavalikon kautta muokataan työnäkymän tilaa tai työnäkymässä esitettyä tietoa.

Valikkojen toteutustapa ja hierarkisuuden luonne säilyy samana alkukohdasta riippumatta. Valikoiden hierarkisuus toteutetaan erilaisten *valintojen* (*menu item*) avulla. Pääsääntöisesti valinnat ovat *komentoja* (*command*), jotka suorittavat sovelluksen kannalta mielekkään operaation. Hierarkia

muodostetaan *alivalikkojen* (*submenu*) avulla. Alivalikkoihin pätevät samat säännöt kuin valikkoihin. Valikoissa esiintyy väliviivoja valintojen ryhmittelemiseksi.



Kuva 2.6 Valikoissa esiintyviä eksoottisia valintoja

Kuvassa 2.6 on esitetty *tila* (*menu state*) ja *valintaryhmä* (*menu radiogroup*). Tilalla on totuusarvo, joka ilmaisee onko toiminto käytössä. Valinnan edessä esiintyvä √-merkki ilmaisee, että toiminto on käytössä. Valikoissa voi esiintyä myös valintaryhmiä, joiden valinnoista ainoastaan yksi voi olla aktiivinen. Aktiivinen valinta ilmaistaan pallomerkillä.

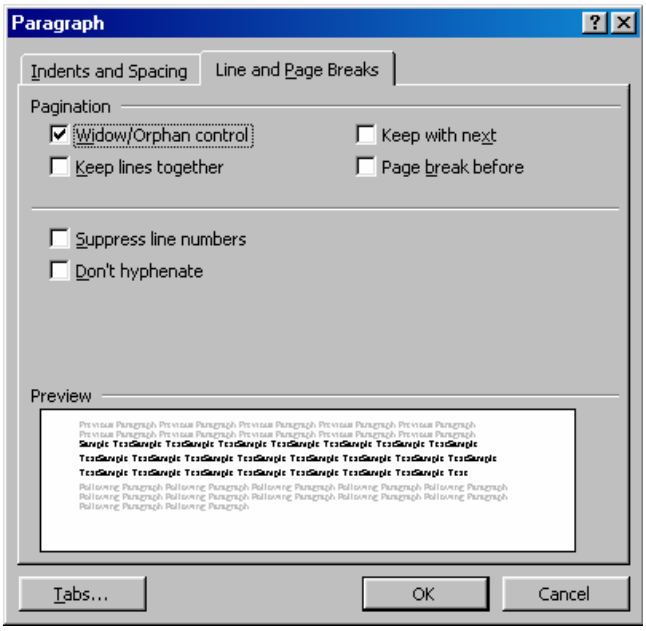
2.2.2 Työkaluikkunat

Työkaluikkuna (*tool window*) on ikkuna, joka sisältää yhden työkalurivin. Työkaluikkuna on riittävän suuri näyttämään koko työkalurivin. Esimerkiksi Microsoft Word -sovelluksessa sivujen ylä- ja alatunnisteiden käsittely tapahtuu työkaluikkunan avulla. Kuvassa 2.7 on kuva kyseisestä työkaluikkunasta.

Kuva 2.7 Microsoft Word -sovelluksen eräs työkaluikkuna

2.2.3 Ponnahdusikkunat

Kuvassa 2.8 on esimerkki Microsoft Word -sovelluksessa esiintyvästä *ponnahdusikkunasta* (*dialog window*). Eräs ponnahdusikkunoissa esiintyvä käyttöliittymän rakenteeseen vaikuttava ominaisuus ovat *lehdet* (*tabcontrol*). Lehtien avulla käyttöliittymän komponentteja voidaan ryhmitellä toiminnallisiin kokonaisuuksiin. Jos ponnahdusikkunaan kuuluvat komponentit vievät paljon tilaa kuvaruudulla, lehtien hyödyntäminen voi pienentää ponnahdusikkunoiden kokoa. Ainoastaan yksi lehti voi olla näkyvissä kerralla.



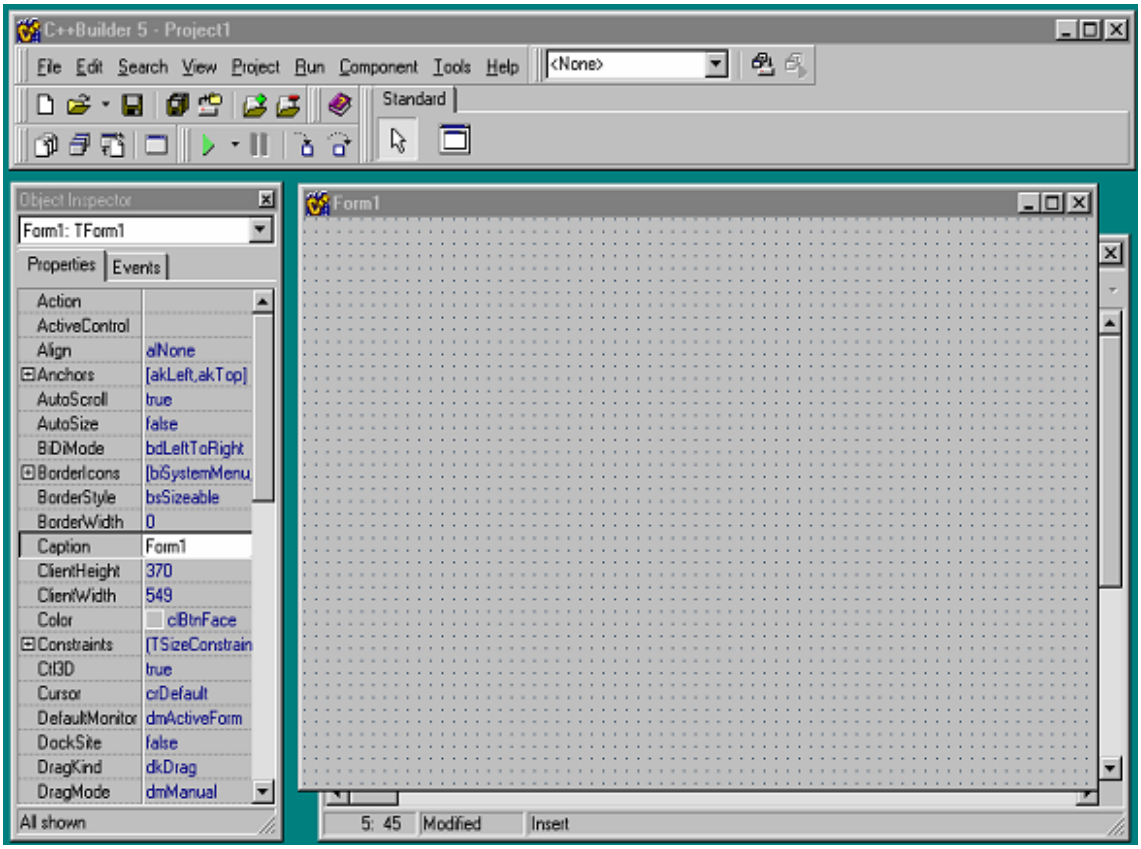
Kuva 2.8 Microsoft Word –sovelluksen ponnahdusikkuna

Ponnahdusikkunat koostuvat komponenteista. Komponenttien avulla esitetään tietoa, joiden perusteella suoritetaan operaatioita esimerkiksi työnäkymissä esittävälle tiedolle. Esimerkkejä ponnahdusikkunoissa käytettävistä komponenteista ovat:

- nappi
- alasvetovalikko
- *syötekenttä* (*inputbox*)
- *valintaruutu* (*checkbox*)
- *valintaryhmä* (*radiogroup*)
- *luetteloruutu* (*listbox*)

2.3 Borland C++ Builder 5.0

Borland -ohjelmistotalon sovelluskehittimet noudattavat samoja periaatteita kuin Microsoftin käyttöjärjestelmät ja sovellukset. Ainoa ero on kehittimen jako useampaan rinnakkaiseen ikkunaan sekä kehitettävän sovelluksen koostuminen myös useasta rinnakkaisesta ikkunasta. Tällä seikalla ei kuitenkaan ole mitään vaikutusta käyttöliittymän rakenteeseen. Kuvassa 2.9 on esitetty Borland C++ Builder -sovelluksen käyttöliittymä.



Kuva 2.9 Borland C++ Builder -sovelluksen käyttöliittymä

2.3.1 Lapsi-ikkunat

Kuvan 2.9 oikeassa laidassa päällekkäin sijaitsevat *lomake* (*form*) ja *lähdekoodi-ikkuna* (*edit window*) ovat esimerkkejä *lapsi-ikkunoista* (*child window*). Myös kuvan vasemmassa laidassa sijaitseva ikkuna on lapsi-ikkuna. Kuvan ylälaidassa sijaitsee pääikkuna. Lapsi-ikkunoita voidaan hyödyntää sovelluksen käyttöliittymässä kahdella tapaa. Lähestymistavat eivät kuitenkaan ota kantaa siihen, miten lapsi-ikkunan käyttöliittymä on toteutettu. Lähestymistavat ovat:

- Lapsi-ikkunat sijaitsevat isäntäikkunan sisäpuolella
- Lapsi-ikkunat sijaitsevat isäntäikkunan ulkopuolella

Jos lapsi-ikkunat sijaitsevat pääikkunan sisäpuolella, toimivat ne pääikkunan ehdoilla. Esimerkiksi tekstinkäsittelyohjelmissa lapsi-ikkunoita ei saa raahattua pääikkunan ulkopuolelle. Jos lapsi-ikkunat sijaitsevat pääikkunan ulkopuolella, toimivat ne ainakin osittain pääikkunan ehdoilla. Ikkunoita voidaan siirtää toisistaan riippumatta, mutta eräissä sovelluskehittimissä kaikki avoinna olevat ikkunat pienentyvät (minimize) sovelluksen pääikkunan pienentyessä.

Lähestymistavat vaikuttavat lähinnä tiedon esittämiseen, eivät sovelluksen toiminnallisuuteen tai edes olennaisesti käyttöliittymän rakenteeseen. Molemmissa tapauksissa lapsi-ikkunat toimivat sovelluksen pääikkunan alaisuudessa. Lapsi-ikkunoilla voi olla omat valikot, työkalurivit ja tilarivi, mutta näin ei välttämättä ole. Lapsi-ikkunoissa esiintyy yleensä työnäkymiä, jotka esittävät sovelluksen käsittelemän tiedon sopivassa muodossa. Lapsi-ikkunoissa voi esiintyä komponentteja, joita esiintyy myös ponnahdusikkunoissa.

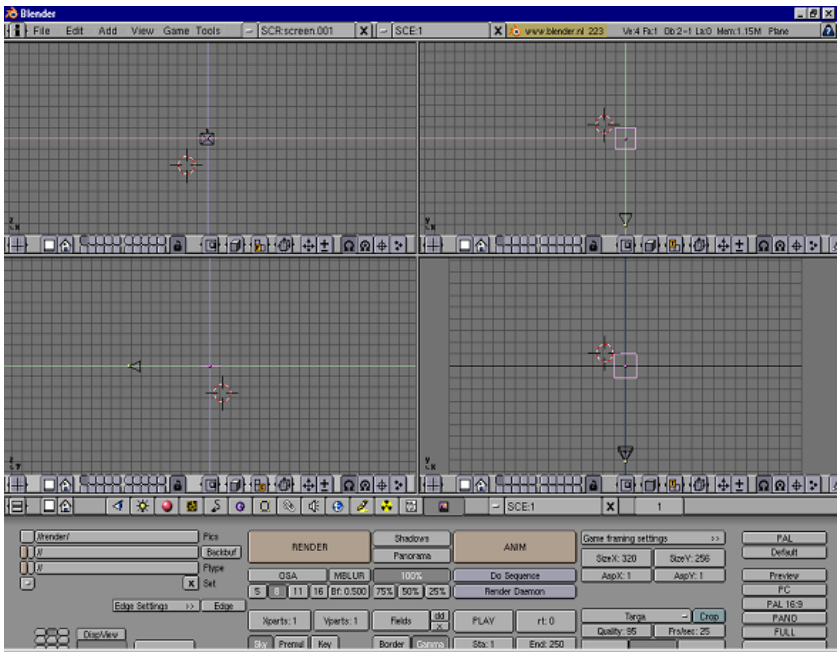
Lapsi-ikkunoilla ja ponnahdusikkunoilla on myös käytännön rajoituksia. Ponnahdusikkunat toimivat toisinaan siten, että ne on suljettava tavalla tai toisella ennen kuin sovelluksen käyttöliittymän muihin osiin pääsee käsiksi. Ponnahdusikkunoiden koko on yleensä kiinnitetty. Lapsi-ikkunoilla ei ole tällaista rajoitusta. Näillä seikoilla ei kuitenkaan ole mitään tekemistä käyttöliittymän rakenteen kanssa.

2.4 Creative Mixer ja Play Center

Creative Mixer ja PlayCenter ovat äänen toistoon liittyviä sovelluksia. Sovellukset eivät noudata käyttöjärjestelmän määrittelemää tai Microsoftin noudattamaa käytäntöä käyttöliittymän ulkoasun suhteen. Molemmissa sovelluksissa on päädytty tavallisesta poikkeavaan ulkoasuun: Mixer -sovelluksen käyttöliittymä muistuttaa miksauspöydän käyttöliittymää, ja PlayCenter-sovelluksen käyttöliittymä muistuttaa CD-soittimen käyttöliittymää. Ulkoasu ei kuitenkaan vaikuta sovellusten käyttöliittymän rakenteeseen. Molemmat sovellukset noudattavat tavallisten ponnahdusikkunoiden rakennetta. Myös muissa multimediasovelluksissa, kuten DVD-soittimissa, esiintyy tavallisesta poikkeavia käyttöliittymä ratkaisuja. Nämäkin sovellukset noudattavat ponnahdusikkunoiden rakennetta.

2.5 Blender Creator

Kuten Creativen Mixer ja PlayCenter -sovelluksissa, myös Blender Creator -sovelluksen käyttöliittymällä on tavallisesta poikkeava ulkoasu. Kuvassa 2.10 on esitetty sovelluksen käyttöliittymä. Komponenttien sijoittelu kuvaruudulla poikkeaa suuresti Microsoftin tuotteista, mutta muuten käyttöliittymän rakenne on suurelta osin samanlainen, tosin poikkeavuuksiakin löytyy.



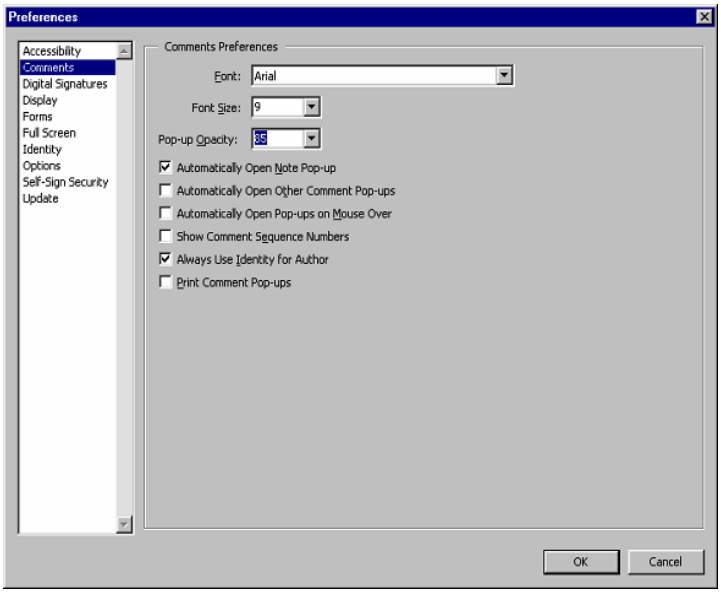
Kuva 2.10 Blender Creator -sovelluksen käyttöliittymä

Käyttöliittymässä esiintyy työkalurivejä, joiden sijoittelu poikkeaa Microsoftin käytännöstä. Työkalurivit esiintyvät neljään osaan jaetun työnäkymän osien alapuolella. Huomattavaa on myös se, että käyttöliittymästä löytyy paljon nappeja, jotka avaavat valikon.

Myös työnäkymien hyödyntäminen on Microsoftin sovelluksista poikkeavaa. Pääosa sovelluksen ikkunasta on jaettu neljään työnäkymään, joiden avulla laaditaan kolmiulotteisia malleja. Kolme työnäkymää esittää tiedon kaksiulotteisena xy-, yz- ja xz -tasoissa, yksi kolmiulotteisena. Tähän on järkevä syy. Tiedon määritteleminen kolmiulotteisessa työnäkymässä on mahdotonta, kuten on myös kolmiulotteisen mallin hahmottaminen kolmen kaksiulotteisen näkymän avulla.

2.6 Acrobat Reader

Acrobat Reader -sovellus noudattaa samoja käyttöliittymän suunnitteluperiaatteita kuin Microsoftin sovellukset. Ponnahdusikkunoissa esiintyy vaihtoehtoinen tapa jakaa ponnahdusikkuna lehtiin. Kuva 2.11 havainnollistaa tätä tapaa. Acrobat Reader -sovellus hyödyntää *listaa* (*list*) lehden valinnan suorittavana komponenttina. Lista esiintyy ponnahdusikkunan vasemmassa laidassa, aktiivinen lehti ponnahdusikkunan oikeassa laidassa. Microsoftin sovelluksissa esiintyy lehdenvalintakomponentti, jossa valittavat lehdet esitetään aktiivisen lehden yläpuolella. Kuva 2.8 havainnollistaa tätä lähestymistapaa.



2.11 Acrobat Reader -sovelluksen eräs ponnahdusikkuna

2.7 Esimerkkejä käyttöliittymän muokkauksesta

Sovelluksissa on yleensä ominaisuuksia, joiden avulla käyttäjät muokkaavat sovelluksesta miellyttävämmän. Tällaisia ominaisuuksia ovat muun muassa pikanäppäimien ja makrojen määrittely. Kuvassa 2.3 on Microsoft Word -sovelluksen asetusparametri-ikkuna, jonka kautta muutetaan sovelluksen asetuksia vastaamaan käyttäjän tarpeita.

Yksinkertainen esimerkki sovelluksen käyttöliittymän muokattavuudesta esiintyy muun muassa Microsoft Word -sovelluksessa. Käyttäjä voi raahata työkalurivin haluamalleen paikalle. Tämä ajatus on tarkoitus laajentaa koskemaan sovelluksen käyttöliittymän jokaista osa-aluetta määrittelemällä käyttöliittymän komponenttien sijainnit kuvaruudulla ulkoisessa tiedostossa.

Eräissä sovelluksissa on mahdollista äänittää makroja. Käyttäjä ensin valitsee toiminnon, joka aloittaa makron äänittämisen. Tämän jälkeen hän suorittaa joukon toimintoja, jotka muodostavat makron. Lopuksi valitaan toiminto, joka pysäyttää makron äänittämiseen. Käyttäjä voi myöhemmin toistaa makron toiminnot. Microsoft Word -tekstinkäsittelysovellus generoi makroista Visual Basic -lähdekoodia. Microsoft Word -sovelluksessa on myös mahdollista kirjoittaa makro suoraan Visual Basic -ohjelmointikielellä.

3 Käyttöliittymän sisäinen esitystapa

Tämän luvun tarkoitus on luoda pohja muokattavien käyttöliittymien toteutukselle. Tavoitteeseen päästään esittelemällä edellisen luvun asiat uudesta näkökulmasta. Luvun 2 kuvissa on annettu esimerkkejä eräiden sovellusten käyttöliittymistä. Tässä luvussa esittelen käyttöliittymistä ominaisuuksia, joita käyttäjä voi muokata. Käyttöliittymät koostuvat komponenteista. Tämän luvun näkökulmasta komponentit jakautuvat kahteen ryhmään: *yksittäiset komponentit* ja *rakenteelliset komponentit*. Termit selitetään luvuissa 3.1 ja 3.2.

Esittelen ensin yksittäiset komponentit, jotta rakenteellisten komponenttien yhteydessä lukijalla olisi jo ajatuksia, miten käyttöliittymien muokattavuus voisi toimia. Luvun tavoitteen näkökulmasta pääpaino on rakenteellisten komponenttien käsittelyssä. Lopuksi esittelen tärkeitä johtopäätöksiä. Yksi tärkeä johtopäätös on tämän luvun asioiden pohjalta laadittu tietorakenne, jota tarvitaan muokattavan käyttöliittymän sisäisessä toteutuksessa. Lukijan tulee ymmärtää tietorakenne ennen kuin seuraavassa luvussa esiteltävä käyttöliittymien muokkaukseen soveltuva merkkauskieli on ymmärrettävissä.

3.1 Yksittäiset komponentit

Yksittäisellä komponentilla on jokin tiedon käsittelyyn liittyvä toiminto käyttöliittymässä. Yksittäisen komponentin avulla suoritetaan tiedon välitys käyttäjältä sovellukselle. Käsittelen seuraavat yksittäiset komponentit:

- nappi
- syötekenttä
- alasvetovalikko
- valintaruutu
- valintaryhmä
- luetteloruutu

Yksittäisestä komponentista esittelen ensimmäiseksi käyttötarkoitukset, joissa komponentti esiintyy käyttöliittymässä. Tämän jälkeen luettelen yksittäisten komponenttien ominaisuuksia, joita käyttäjä

todennäköisesti haluaa muokata. Komponenteilla on ominaisuuksia, jotka vaikuttavat ainoastaan sovelluksen ulkoasuun. Tällaisia ominaisuuksia ovat esimerkiksi komponentin sijainti ja koko kuvaruudulla. Komponenteissa esiintyvät merkkijonot, kirjasinlajit ja värit ovat myös ulkoasuun vaikuttavia ominaisuuksia. Luvussa on tarkoitus esitellä komponenteista ominaisuuksia, joilla on vaikutusta käyttöliittymän rakenteeseen tai toiminnallisuuteen.

Esittelen viimeiseksi tietorakenteen, jonka avulla komponentin muokattavuus on toteutettavissa. Tietorakenteet ovat lähes poikkeuksetta tietueita. Tietueille luetellaan kentät, jotka vaikuttavat käyttöliittymän rakenteeseen tai toiminnallisuuteen.

3.1.1 Nappi

Nappi esiintyy käyttöliittymässä laukaisemassa sovelluksen kannalta mielekkään operaation. Kuvassa 2.8 esiintyvän Microsoft Word -sovelluksen ponnahdusikkunan *Ok*-napin painallus laukaisee operaation, joka muotoilee tekstikappaleen. Nappi voi suorittaa myös käyttöliittymän kannalta mielekkään operaation. Kuvassa 2.8 esiintyvän *Cancel*-napin painallus todennäköisesti ainoastaan sulkee ponnahdusikkunan suorittamatta ainoatakaan tekstinkäsittelyyn liittyvää operaatiota.

Käyttäjä määrittelee operaatioon, jonka napin painallus laukaisee. Aikaisemmin mainittiin, että napin painallus voi laukaista käyttöliittymään tai sovellusalueeseen liittyvän operaation. Ajatus käyttöliittymän muokattavuudesta äärimmäisyyksiin vietynä antaa ymmärtää, että käyttäjä määrittelee suoritettavan toimenpiteen. Käytännössä tämä tarkoittaa komentojonon määrittelemistä ohjelmointikielellä. Edellä mainituista asioista voi päätellä, että käyttää määrittelee napin *kytkennän* (*coupling*) suoritettavaan operaatioon. Kytkentä tarkoittaa pro gradu -tutkielmassani tapaa, jolla käyttäjän syöttämä tieto välitetään sovelluksen käyttöliittymästä sovelluksen toiminnalliseen ytimeen käsiteltäväksi. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus.

3.1.2 Tilanvalitsin

Tilanvalitsin esiintyy käyttöliittymässä ilmaisemassa onko toiminto käytössä. Kuvassa 2.3 on Microsoft Excel –sovelluksen työkalurivialueen toisella rivillä työkalurivi, jonka avulla voi vaikuttaa

tekstin muotoiluun. Työkalurivissä on tilanvalitsimet lihavoinnin, kursivoinnin ja alleviivauksen vaihtamiseksi.

Käyttöliittymän toiminnallisuuden näkökulmasta katsottuna käyttäjä määrittelee operaation, joka vastaanottaa tilanvalitsimella asetetun tilan. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus.

3.1.3 Valintaruutu

Valintaruutu esiintyy käyttöliittymässä ilmaisemassa onko toiminto käytössä. Kuvassa 2.8 on Microsoft Word -sovelluksen eräs ponnahdusikkuna, jossa esiintyy valintaruutuja. Kuvassa 3.3 on esimerkki valintaruudusta. Jos valintaruutu on rastitettu, niin dokumentin ensimmäiselle sivulle tulostetaan sivunumero.

Kuva 3.3 Esimerkki valintaruudusta

Käyttäjä määrittelee operaation, joka vastaanottaa valintaruudussa suoritetun valinnan. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus.

3.1.4 Syötekenttä

Syötekenttä esiintyy käyttöliittymässä vastaanottamassa näppäimistöltä syötettyä tietoa. Kuvassa 2.9 esiintyvän Borland C++ Builder -sovelluksen käyttöliittymässä on korostettuna syötekenttä, johon käyttäjä voi syöttää lomakkeen otsikon. Syötekenttä esiintyy kuvassa vasemmalla keskellä. Kuvassa 2.11 on Acrobat Reader -sovelluksen eräs ponnahdusikkunassa, jossa esiintyy kaksi syötekenttää, jotka vastaanottavat kokonaislukuja. Kuvassa 3.1 on syötekenttä, johon voi kirjoittaa tallennettavan tiedoston nimen.

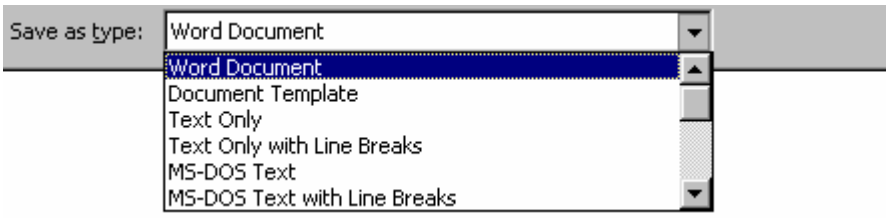
Kuva 3.1 Esimerkki syötekentästä

Halutessaan käyttäjä voi muokata syötekentän vastaanottaman tiedon laatua. Tämä onnistuu helpoimmin määrittelemällä käyttöliittymäkirjastoon useampia erilaisia syötekenttiä, jotka ovat omistautuneet yhden tyyppisen tiedon vastaanottamiseen. Esimerkkejä tietotyypeistä ovat merkkijonot ja kokonaisluvut.

Käyttäjä määrittelee operaation, joka vastaanottaa syötekenttään syötetyn tiedon. Tiedon vastaanottavan operaation näkökulmasta tieto voi olla aina merkkijonomuotoista, koska käyttöliittymäkirjaston syötekentät ovat omistautuneet ainoastaan yhden tyyppisen tiedon vastaanottamiseen. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus.

3.1.5 Alasvetovalikko

Alasvetovalikko esiintyy käyttöliittymässä esittämässä käyttäjälle joukon valintoja, joista käyttäjä voi valita yhden. Kuvassa 2.3 on Microsoft Excel -sovelluksen työkalurivialue, jonka toisen rivin vasemmassa laidassa esiintyvästä alasvetovalikosta käyttäjä voi valita kirjasinlajin. Kuvassa 3.2 on alasvetovalikko, josta valitaan tiedoston formaatti.

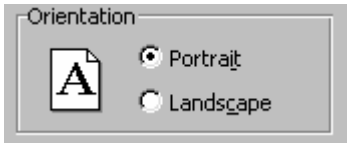


Kuva 3.2 Esimerkki alasvetovalikosta

Käyttäjä määrittelee alasvetovalikossa esiintyvät valinnat ja operaation, joka vastaanottaa alasvetovalikossa suoritetun valinnan. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus. Alasvetovalikossa näytettävät valinnat ovat toteutettavissa lista -tietorakenteena, joka sisältää valinnat (*merkkijono*, *arvo*) -pareina. Pari ilmaisee sekä käyttäjälle näytettävän merkkijonon että operaation vastaanottaman kokonaislukuarvon.

3.1.6 Valintaryhmä

Valintaryhmä esiintyy käyttöliittymässä esittämässä käyttäjälle joukon valintoja, joista käyttäjä voi valita yhden. Kuvassa 3.4 on esimerkki valintaryhmästä. Kuvan valintaryhmän avulla määrätään suunta, jossa teksti tulostetaan paperille.



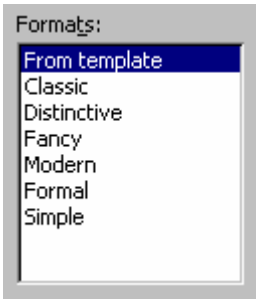
Kuva 3.4 Esimerkki valintaryhmästä

Käyttäjä määrittelee valintaryhmässä esiintyvät valinnat ja operaation, joka vastaanottaa valintaryhmässä suoritetun valinnan. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus. Valintaryhmässä näytettävät valinnat ovat toteutettavissa lista -tietorakenteena, joka sisältää valinnat (*merkkijono*, *arvo*) -pareina. Pari ilmaisee sekä käyttäjälle näytettävän merkkijonon että operaation vastaanottaman kokonaislukuarvon.

3.1.7 Luetteloruutu

Luetteloruutu esiintyy käyttöliittymässä esittämässä käyttäjälle joukon valintoja, joista käyttäjä voi valita yhden. Kuvassa 3.5 on esimerkki luetteloruudusta, jonka avulla valitaan sisällysluettelon ulkoasu tekstinkäsittelysovelluksessa.

Käyttäjä määrittelee luetteloruudussa esiintyvät valinnat ja operaation, joka vastaanottaa luetteloruudussa suoritetun valinnan. Sisäisen toteutustavan näkökulmasta kytkentä voidaan toteuttaa merkkijonona, jonka arvo on suoritettavan operaation tai komentojonon yksiselitteinen tunnus. Luetteloruudussa näytettävät valinnat ovat toteutettavissa lista -tietorakenteena, joka sisältää valinnat (*merkkijono*, *arvo*) -pareina. Pari ilmaisee sekä käyttäjälle näytettävän merkkijonon että operaation vastaanottaman kokonaislukuarvon.



Kuva 3.5 Esimerkki luetteloruudusta

3.2 Rakenteelliset komponentit

Luvussa 3.1 käsittelin yksittäisiä komponentteja. Yksittäisten komponenttien avulla suoritetaan tiedon välitys käyttäjältä sovellukselle. Rakenteellisella komponentilla ei ole vastaava tiedonvälitystehtävää käyttöliittymässä.

Useat luvussa 2 esitellyt käyttöliittymän osat ovat esimerkkejä rakenteellisista komponenteista. Rakenteellisten komponenttien avulla muodostetaan toiminnallisia kokonaisuuksia luvussa 3.1 käsitellyistä komponenteista. Rakenteelliset komponentit koostuvat yksittäisistä komponenteista. Käsittelen seuraavat rakenteelliset komponentit:

- valikot
- työkalurivialue ja työkalurivit
- ponnahdusikkunat
- lapsi-ikkunat

3.2.1 Valikot

Yleensä valikot tarjoavat käyttäjälle hierarkian valintoja, joiden avulla laukaistaan sovelluksen toiminnan kannalta mielekkäitä operaatiota. Microsoft Word -tekstinkäsittelysovelluksen valikoiden kautta on mahdollista suorittaa dokumentin ja tekstin muotoiluun liittyviä toimintoja. Microsoft Excel -taulukkolaskentasovelluksen valikoiden kautta on mahdollista suorittaa taulukon ja solujen muotoiluun liittyviä toimintoja. Borland C++ Builder -sovelluskehittimen valikoiden kautta on mahdollista asettaa lähdekoodin kääntämiseen liittyviä parametrejä. Valikoiden sisältö on sovelluskohtaista. Yleensä valikot koostuvat seuraavista valinnoista:

- komento
- tila
- valintaryhmä
- alivalikko

Komennon tapauksessa käyttäjä määrittelee komentojonon, jonka komennon valitseminen laukaisee. Tilan ja valintaryhmän tapauksessa käyttäjä määrittelee operaation, jolle käyttäjän suorittama valinta välitetään. Käyttäjä määrittelee valinnan kytkennän sovelluksen toiminnalliseen ytimeen. Valikon ja alivalikon tapauksessa käyttäjä määrittelee valikossa esiintyvät valinnat ja niiden järjestyksen.

Valikko ja alivalikko esitetään tietueina, joihin taltioidaan valikkoon kuuluvat valinnat lista -tietorakenteen avulla. Valinta esitetään tietueena, jossa esiintyy käyttäjälle esitettävä merkkijono ja valinnan kytkentä toiminnallisen ytimen operaatioon.

3.2.2 Työkalurivialue ja työkalurivit

Käyttäjä määrittelee työkalurivialueen koostumuksen ja työkalurivien järjestyksen työkalurivialueella. Sisäisen esitystavan näkökulmasta katsottuna työkalurivialueen koostumus ja työkalurivien järjestys ovat toteutettavissa lista -tietorakenteen avulla. Työkalurivialue koostuu työkaluriveistä.

Käyttäjä määrittelee työkalurivin koostumuksen ja yksittäisten komponenttien järjestyksen työkalurivissä. Sisäisen esitystavan näkökulmasta katsottuna työkalurivin koostumus ja yksittäisten komponenttien järjestys ovat toteutettavissa lista -tietorakenteen avulla. Yleisimmin työkaluriveissä esiintyvät komponentit ovat tilanvalitsimet, syötekentät ja alasvetovalikot.

3.2.3 Ponnahdusikkunat ja lapsi-ikkunat

Käyttäjä määrittelee yksittäisten komponenttien ryhmittelyn toiminnallisiin kokonaisuuksiin. Tämä tapahtuu sijoittelemalla yksittäisiä komponentteja ponnahdusikkunoihin ja lapsi-ikkunoihin. Sisäisen esitystavan näkökulmasta katsottuna ponnahdusikkunan koostumus on toteutettavissa lista -tietorakenteen avulla. Yleisimmin ponnahdusikkunoissa esiintyviä yksittäisiä komponentteja ovat:

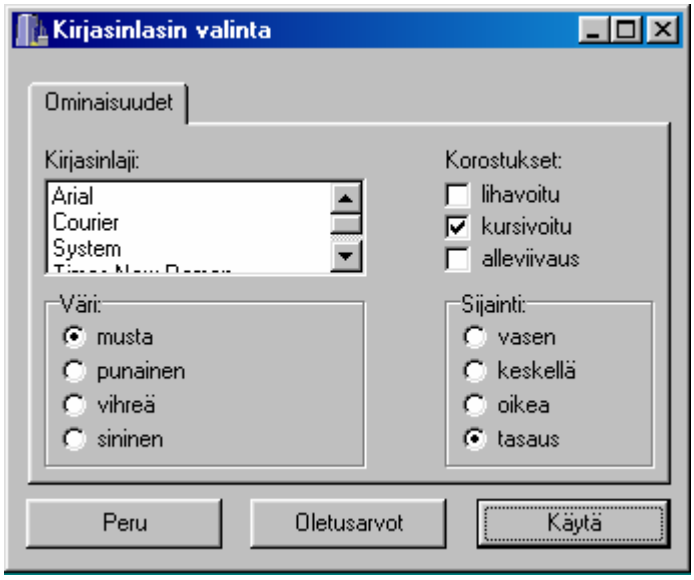
- nappi
- syötekenttä
- alasvetovalikko
- luetteloruutu
- valintaruutu
- valintaryhmä

Myös lapsi-ikkunan sisäinen esitystapa on toteutettavissa lista -tietorakenteen avulla. Lapsi-ikkunat voivat koostua yksittäisten komponenttien lisäksi mm. sovelluskohtaisista työnäkymistä, valikoista, työkalurivialueista ja tilarivistä.

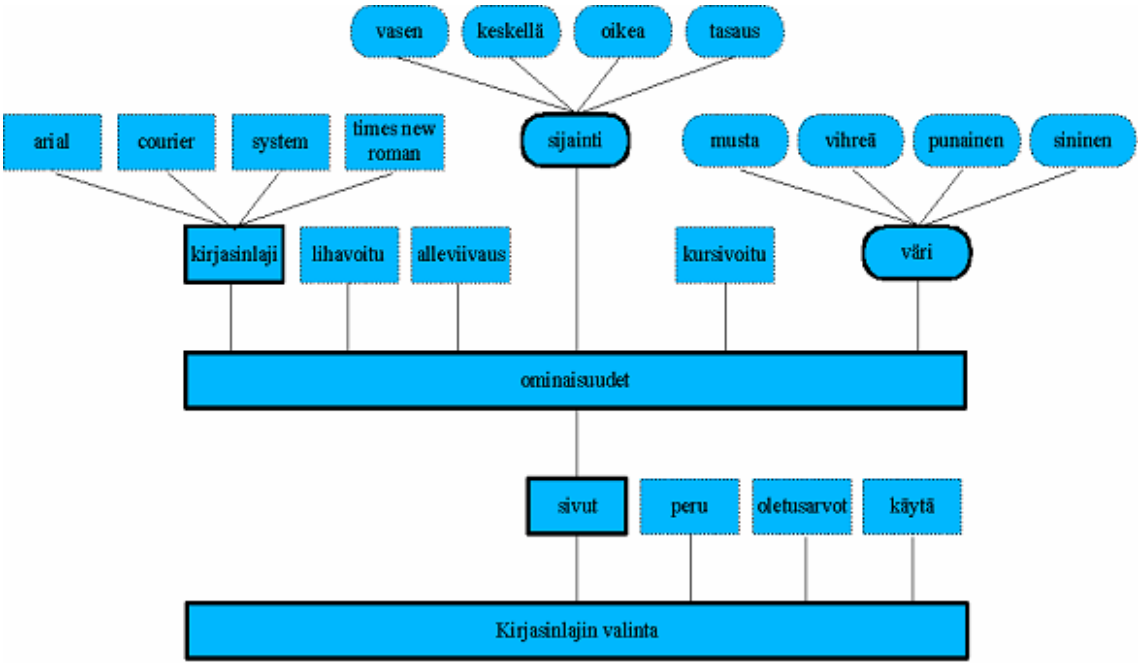
3.3 Tietorakenne

Tässä luvussa on esitelty komponenteista ominaisuuksia, joita käyttäjä halunnee muokata. Yksittäiset komponentit sisältävät viittauksen toiminnallisen ytimen operaatioon, jolle komponentin ylläpitämä tieto välitetään. Rakenteelliset komponentit sisältävät lista -tietorakenteen, jossa on lueteltu rakenteellisen komponentin koostumus. Rakenteellinen komponentti voi koostua sekä rakenteellisista että yksittäisistä komponenteista. Tämän vuoksi käyttöliittymä on esitettävissä puurakenteen avulla [8, ss. 128-138, ss. 218-223].

Kuvassa 3.7 on esimerkki ponnahdusikkunasta, jossa on käytetty useita tässä luvussa esiteltyjä yksittäisiä komponentteja. Kuvassa 3.8 on havainnollistettu puurakenteena kuvan 3.7 ponnahdusikkuna. Puurakenteen juurena on ponnahdusikkuna, jonka nimi on *kirjasinlajin valinta*. Ponnahdusikkunaan on liitetty kolme nappia ja sivu. Nappien nimet ovat *peru*, *oletusarvot* ja *käytä*. Sivun nimi on *ominaisuudet*. Sivuun on liitetty luetteloruutu, kaksi valintaryhmää ja kolme valintaruutua. Luetteloruudun nimi on *kirjasinlaji.* Luetteloruudusta löytyy neljä valintaa. Valintaryhmien nimet ovat *väri* ja *sijainti.* Valintaruutujen nimet ovat *lihavoitu*, *kursivoitu*, *alleviivaus*.



Kuva 3.7 Esimerkki ponnahdusikkunasta



Kuva 3.8 Esimerkki puurakenteesta

3.4 Esimerkkejä muokattavuudesta

Tässä luvussa on käsitelty yksittäisiä ja rakenteellisia komponentteja, joista on esitelty mm. käyttötarkoitus käyttöliittymässä. Eräillä käsitellyistä komponenteista on yhteisiä piirteitä. Yhteisistä piirteistä voi tehdä sen johtopäätöksen, että komponentit ovat keskenään vaihtoehtoisia saman toiminnon suorittamiseksi. Vaihtoehtoja esiintyy ainakin seuraavissa tilanteissa:

- komentojen suorittaminen
- valinnan suorittaminen
- tilan valitseminen
- toiminnallisuuden ryhmitteleminen

3.4.1 Komennon suorittaminen

Tässä komennolla tarkoitetaan toimenpidettä, jonka suorittaminen ei edellytä parametrejä. Tällaisia toimenpiteitä voivat olla toiminnallisen ytimen operaatiot, käyttöliittymään liittyvät operaatiot tai komentojonot. Seuraavat lähestymistavat komennon suorittamiseksi ovat keskenään vaihtoehtoiset:

- valikon komento
- nappi

3.4.2 Valinnan suorittaminen

Tässä valinnalla tarkoitetaan toimenpidettä, josta usean valinnan joukosta valitaan yksi. Tällainen komponentti voidaan kytkeä toiminnallisen ytimen operaatioon, joka vastaanottaa tiedon käyttäjän suorittamasta valinnasta. Seuraavat lähestymistavat valinnan suorittamiseksi ovat keskenään vaihtoehtoiset:

- valikon valintaryhmä
- ponnahdusikkunan valintaryhmä
- luetteloruutu
- alasvetovalikko

3.4.3 Tilan valitseminen

Tilalla tarkoitetaan tässä toimintoa, joka voi olla käytössä. Tällainen komponentti voidaan kytkeä toiminnallisen ytimen operaatioon, joka vastaanottaa tiedon käyttäjän suorittamasta valinnasta. Seuraavat lähestymistavat komennon suorittamiseksi ovat keskenään vaihtoehtoiset:

- valintaruutu
- valikon tila
- tilan valitsin

3.4.4 Toiminnallisuuden ryhmitteleminen

Luvuissa 3.4.1– 3.4.3 esiteltyjen vaihtoehtoisten lähestymistapojen pohjalta on laadittu taulukko 3.1. Toiminnallisuuden uudelleenjärjestely sovelluksen käyttöliittymässä edellyttää useasti komponentin vaihtamista, mutta kytkennän säilyttämistä aikaisemman ja uuden komponentin välillä. Taulukossa vasemmanpuoleisessa sarakkeessa on nimetty komponentti, joka esiintyy alkuperäisessä käyttöliittymässä ponnahdusikkunoissa. Keskimmäisessä sarakkeessa mainitaan rakenteellinen komponentti, jonka alaisuuteen alkuperäisen komponentin toiminnallisuus on siirrettävissä. Oikeanpuoleisessa sarakkeessa on nimetty yksittäinen komponentti, jota käyttämällä rakenteellisessa komponentissa saadaan aikaiseksi alkuperäistä käyttöliittymää vastaava toiminnallisuus. Taulukon ensimmäinen rivi tulee lukea seuraavasti: ponnahdusikkunoissa esiintyvä nappi on siirrettävissä valikkoon komennoksi.

Taulukko voidaan lukea myös kääntäen: valikoissa esiintyvä komento on siirrettävissä ponnahdusikkunaan napiksi. Taulukossa 3.1 tähdillä merkityt muutokset eivät ole aina toteutettavissa. Alasvetovalikoissa ja luetteloruuduissa esiintyvien valintojen lukumäärät voivat vaihdella sovelluksen suorituksen aikana. Alasvetovalikoissa tai luetteloruuduissa voi yksinkertaisesti olla liian paljon valintoja näytettäväksi ponnahdusikkunassa Tämän vuoksi komponentin muuttaminen valintaryhmäksi ei välttämättä ole mielekästä.

Taulukko 3.1 Vaihtoehtoisia komponentteja

Alkuperäinen komponentti:	**Rakenteellinen komponentti:**	**Uusi komponentti:**
Nappi	Valikko	Komento
Syötekenttä	Työkalurivi	Syötekenttä
Alasvetovalikko	Valikko	Valintaryhmä*
Alasvetovalikko	Työkalurivi	Alasvetovalikko
Alasvetovalikko	Ponnahdusikkuna	Valintaryhmä*
Luetteloruutu	Valikko	Valintaryhmä*
Luetteloruutu	Työkalurivi	Alasvetovalikko
Luetteloruutu	Ponnahdusikkuna	Alasvetovalikko
Valintaruutu	Valikko	Tila
Valintaruutu	Työkalurivi	Tilan valitsin
Valintaryhmä	Valikko	Valintaryhmä
Valintaryhmä	Työkalurivi	Alasvetovalikko
Valintaryhmä	Ponnahdusikkuna	Luetteloruutu

4 Käyttöliittymän merkkauskieli

Tässä luvussa esittelen käyttöliittymien määrittelyyn soveltuvan merkkauskielen. Merkkauskielen avulla on mahdollista määritellä käyttöliittymän rakenteeseen ja ulkoasuun liittyviä seikkoja. Käyttöliittymän merkkauskielen avulla käyttäjä määrittelee komponenttien kytkennät sovelluksen toiminnalliseen ytimeen.

Luvun tarkoitus on esitellä tapoja määritellä käyttöliittymän ominaisuuksia. Ensimmäiseksi tarkastelen asetuskieltä, jolla käyttäjä määrittelee mieltymyksensä käyttöliittymän toimivuuden suhteen. Tämän jälkeen tarkastelen HTML- ja UIML -merkkauskieliä. Seuraavaksi esittelen yksinkertaisen käyttöliittymän merkkauskielen, jolla käyttäjä voi määritellä käyttöliittymän. Viimeiseksi annan esimerkin merkkauskielen käytöstä.

4.1 Asetuskieli

Tämän luvun tarkoitus on pohjustaa ajatusta kielestä, jolla käyttäjä voi määritellä käyttöliittymän ominaisuuksia. Luku esittelee asetuskielen, jolla käyttäjä voi määritellä mieltymyksiään sovelluksen käytön suhteen. Asetuskieli mahdollistaa yksinkertaisen tavan muuttaa sovelluksen toimintaa ennalta määrätyissä tilanteissa. Tilanteet ja tavat, joissa sovelluksen toimintaa voi muokata asetuskielen avulla, ovat sovelluskohtaisia.

Tarkastellaan ensimmäiseksi asetuskielen kielioppia. Mikä tahansa tekstitiedosto voi noudattaa asetuskielen kielioppia. Yleisesti tiedostoa, joka noudattaa asetuskielen kielioppia, kutsutaan tiedoston laajennusosasta riippumatta asetustiedostoksi. Yleisimmin käytettyjä laajennusosia ovat .INI ja .CFG.

```
; ensimmäisen ryhmän määrittely
[ryhmän_nimi_1]
tunnus11 = arvo11
tunnus12 = arvo12

; toisen ryhmän määrittely
[ryhmän_nimi_2]
tunnus21 = arvo21
```

Listaus 4.1 Esimerkki asetustiedostosta

Listauksessa 4.1 on esimerkki asetuskieltä noudattavasta tekstitiedostosta. Esimerkissä on havainnollistettu asetuskielen pääpiirteet. Asetuskielen pääajatus on seuraavanlainen: tunnuksen tulee kuulua ryhmään ja tunnukselle tulee sijoittaa arvo. Käytännössä pääajatus toteutuu noudattamalla joukkoa yksinkertaisia sääntöjä.

Asetuskieli koostuu kahdenlaisista määrityksistä: ryhmän ja tunnuksen määrittelystä. Ensimmäiseksi tulee esiintyä ryhmän määrittely, muutoin seuraavaksi määriteltävä tunnus ei kuuluisi mihinkään ryhmään. Tämän jälkeen ryhmien ja tunnuksien määrittelyjä voi esiintyä mielivaltaisesti. Asetuskielessä voi esiintyä myös kommentteja. Kommentit ja määrittelyt sijoitetaan kukin omalle rivillensä.

Ryhmän määritteleminen tapahtuu kirjoittamalla ryhmän nimi hakasulkujen sisälle. Ryhmässä ei tarvitse olla yhtään tunnuksen määrittelyä tai määrittelyjä voi olla useita. Ryhmässä ei kuitenkaan voi olla kahta samaa tunnusta. Tunnuksen määritteleminen tapahtuu kirjoittamalla uudelle riville tunnuksen nimi, yhtäläisyysmerkki ja tunnuksen arvo. Toisin kuin ohjelmointikielissä, asetuskielessä ei tarvitse määritellä tunnuksen tyyppiä. Tyypillinen asetuskieli hyväksyy arvoiksi kokonaislukuja, liukulukuja, totuusarvoja ja merkkijonoarvoja. Monipuolisempi asetuskieli voi hyväksyä arvoiksi listoja, taulukoita ja tietueita.

Asetuskielen avulla on mahdollista määritellä sovelluksen toiminnan oletusarvot. Tässä sovelluksen toiminnalla tarkoitetaan tietojen käsittelyä. Asetuskielen avulla on mahdollista määritellä sovelluksen toimintaan liittyviä parametrejä, esimerkiksi matemaattisissa kaavoissa esiintyviä vakioita.

Asetuskielen avulla on mahdollista määritellä käyttöliittymän ominaisuuksia. Eräs määriteltävissä oleva kohde voi olla käyttöliittymässä hyödynnettävä kieli, jonka voi vaihtaa esimerkiksi englanninkielestä suomenkieleen.

Asetuskielen avulla on mahdollista määritellä tärkeiden komponenttien sijainnit kuvaruudulla. Yleensä asetuskielen avulla ei kuitenkaan määritellä jokaisen yksittäisen komponentin sijaintia kuvaruudulla koordinaattien avulla. Asetuskieli on mielekäs tilanteessa, jossa sovellus esittää käyttäjälle työnäkymän, joka voidaan jakaa pienempiin osiin esitettävän tiedon perusteella, esimerkiksi puurakenteisiin ja listoihin. Esimerkki tällaisesta sovelluksesta on Windows Explorer. Asetustiedoston avulla voidaan määritellä edellä mainittu jako. Yksinkertainen tapa on määritellä ensin suhde, jossa työnäkymä on jaettu osiin, ja tämän jälkeen kunkin työnäkymän osion sisältö

määritellään erikseen. Tässä tapauksessa asetuskieli ei mahdollista täysin mielivaltaista työnäkymän jakoa.

Asetuskieli ei sovellu käyttöliittymän komponenttirakenteen uudelleenmäärittelyyn, vaikka tietyin järjestelyin tämä on mahdollista. Edellytyksenä on komponentin ominaisuuksien sijoittaminen omaan ryhmäänsä. Lisäksi täytyy määritellä komponenttien vanhempi-lapsi -suhteet esimerkiksi asettamalla *vanhempi* -tunnuksen arvoksi ryhmän nimi. Jos tavoitteena on mahdollistaa käyttöliittymän rakenteen määritteleminen, ei ole järkevää käyttää asetuskieltä vaan tarkoitusta varten suunniteltua merkkauskieltä.

4.2 HTML –lomakkeet

Lomakkeen määritteleminen HTML -kielellä ja lomakkeeseen syötettyjen tietojen käsittely muistuttaa läheisesti ongelmaa, johon tutkielma pyrkii tarjoamaan vastauksen [11, ss. 213-233]. Tämän HTML -lomakkeita käsittelevän luvun tarkoitus on tuoda julki tämä yhteys. Ensimmäiseksi tarkastellaan miten HTML -kielessä määritellään lomake. Seuraavaksi esitellään miten lomakkeeseen syötetyt tiedot käsitellään. Tämän jälkeen kerrotaan miten tulokset näytetään käyttäjälle. Toiminta kokonaisuutena käydään läpi esimerkin avulla. Lopuksi esitellään yhteneväisyydet HTML -lomakkeiden toiminnan ja tässä tutkielmassa käsiteltävän ongelman välillä.

Luvussa 3 käsittelin muokattavia käyttöliittymiä rajatun komponenttijoukon avulla. Komponentit valitsin toiminnallisuuden perusteella. Myös HTML -dokumenttien lomakkeissa voi esiintyä rajallinen joukko komponentteja. Todennäköisesti HTML -lomakkeissa sallittujen komponenttien joukko on rajattu myös toiminnallisuuden suhteen. Lomakkeissa voi esiintyä seuraavanlaisia komponentteja:

- Tekstikentät ja tekstialueet
- lähetä ja palauta -painikkeet
- valintaruudut ja valintanapit
- luettelo

HTML -kieli on luonteeltaan rakenteinen. Tämä tarkoittaa sitä, että jokaisella kielen elementillä on alku- ja loppumerkitsimet. Lomake alkaa <form> -merkitsimestä ja loppuu </form> -merkitsimeen.

Osa lomakkeiden sallituista komponenteista noudattaa tätä sääntöä, osa ei. Lomakkeiden koostumus ilmaistaan merkitsimien ja niiden lisäpiirteiden avulla.

Lomakkeet ovat lisätty kieleen luomaan interaktiivisuutta palvelun ja käyttäjän välille, mutta kielen avulla määritellään ensisijaisesti dokumentteja. Tämän vuoksi kielessä ei ole mahdollista määritellä esimerkiksi lomakkeen komponenttien sijainteja kuvaruudulla, sillä selain sijoittelee dokumentin sisällön kuvaruudulle selaimen ikkunan koon perusteella.

Lomakkeeseen syötetyt tiedot lähetetään ohjelmalle, joka vastaa tietojen käsittelystä. Ohjelman vastuulla on myös tuloksen palauttaminen HTML -dokumenttina. Tällaista ohjelmaa kutsutaan CGI -ohjelmaksi. Lomakkeen tiedot välitetään CGI -ohjelman käsiteltäväksi joko GET- tai POST -metodilla. Sekä ohjelman nimi, jolle tiedot välitetään että tiedon välitystapa määritellään <form> -merkitsimen lisäpiirteiden avulla. Menetelmä vaikuttaa tapaan, jolla CGI -ohjelma vastaanottaa lomakkeen tiedon. GET -metodissa CGI -ohjelma voi lukea lomakkeen tiedot ympäristömuuttujista. POST -metodissa CGI -ohjelma voi lukea lomakkeen tiedot standardista tulostusvirrasta.

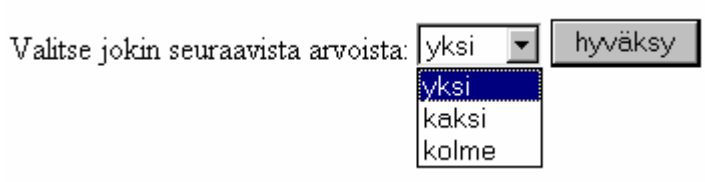
Jos on olemassa tapa välittää lomakkeeseen syötetyt tiedot käsiteltäväksi, täytyy olla myös jokin tapa vastaanottaa käsittelyn tulokset. CGI -ohjelma on vastuussa tietojen käsittelystä. Tämän vuoksi CGI -ohjelma on myös vastuussa tuloksien välittämisestä käyttäjälle. CGI -ohjelma palauttaa tulokset tulostamalla HTML -kieltä noudattavan dokumentin, joka välitetään näytettäväksi käyttäjän selaimessa.

```
<form method="GET" ACTION="example.cgi">
<p>
Valitse jokin seuraavista arvoista:
<select name="arvo">
<option value=1>yksi
<option value=2>kaksi
<option value=3>kolme
</select>
<input type=submit name=toiminta value=hyväksy>
</p>
</form>
```

Listaus 4.2 Esimerkki HTML –lomakkeen määrittelystä

Listauksessa 4.2 on esimerkki HTML –kielellä laaditusta lomakkeen määrittelystä. Ensimmäisenä listauksessa esiintyy merkitsin, joka aloittaa lomakkeen määrittelyn. Merkitsimen lisäpiirteiden avulla on määritelty tapa, jolla lomakkeen tiedot käsitellään sekä ohjelma, joka käsittelee tiedot. Ensimmäisenä lomakkeen sisällä esiintyy viesti, jolla opastetaan käyttäjää. Tämän jälkeen

määritellään alasvetovalikko, ja siinä esiintyvät arvot. Seuraavaksi määritellään nappi, jota painamalla lomakkeen tiedot käsitellään. Viimeisenä esiintyy merkitsin, joka päättää lomakkeen määrittelyn.



Kuva 4.1 Esimerkki HTML –lomakkeen ulkoasusta

Oletetaan, että käyttäjä lataa selaimeensa dokumentin, jossa esiintyy listauksessa 4.2 määritelty lomake. Todennäköisesti hän ei tiedä mitä dokumentin määrittely pitää sisällään. Kuvassa 4.1 on havainnollistettu miltä lomake näyttää selaimessa. Jos käyttäjä painaa hiirellä nappia, jossa on nuolen kärki alaspäin, käyttäjälle näytetään vaihtoehdot. Käyttäjä voi valita vaihtoehdoista sopivan. Käyttäjän tulee painaa *hyväksy*-nappia lomakkeeseen syötettyjen tietojen käsittelemiseksi.

Kun käyttäjä painaa *hyväksy*-nappia, selain ensin muodostaa kaksi ympäristömuuttujaa CGI -ohjelmaa varten. Muuttujan *toiminta* arvoksi asetetaan *hyväksy*. Muuttujan *arvo* riippuu käyttäjän tekemästä valinnasta, mutta kyseessä on kokonaisluku. Tämän jälkeen suoritetaan CGI -ohjelma, joka suorittaa toimenpiteitä ympäristömuuttujien arvojen perusteella. Tämän esimerkin tapauksessa CGI -ohjelma voisi palauttaa sivun, jossa kerrotaan käyttäjälle hänen suorittama valintansa.

Esimerkin yhteys muokattaviin käyttöliittymiin on seuraava: Sovellus, joka toteuttaa tässä tutkielmassa esitetyt asiat, vastaa Internet -selainta. Käyttöliittymän merkkauskieli vastaa HTML -kielen lomakkeiden määrittelyä. Komponentin kytkentä toiminnallisen ytimen palveluun vastaa lomakkeeseen syötettyjen tietojen välittämistä ympäristömuuttujissa CGI -ohjelmalle. Sovelluksen toiminnallinen ydin vastaa CGI -ohjelmaa.

4.3 UIML

User Interface Markup Language (*UIML*) on metakieli, jolla voidaan määritellä sovelluksen käyttöliittymän ominaisuudet [1]. Käyttöliittymän määrittely koostuu neljästä osasta: *rakenteesta* (*structure*), *tyylistä* (*style*), *sisällöstä* (*content*) ja *toiminnallisuudesta* (*behaviour*).

Rakenne määrittelee käyttöliittymän muodostavien komponenttien suhteet. Suhde viittaa yleensä komponenttien muodostamaan puurakenteeseen. Käyttöliittymän määrittely voi koostua yhdestä tai useammasta rakenteesta.

Tyyli määrittelee käyttöliittymän ulkoasuun liittyviä ominaisuuksia. Tällaisia ominaisuuksia ovat muun muassa käytettävät kirjasinlajit ja värit. Jokaiselle hyödynnettävälle käyttöliittymäkirjastolle täytyy määritellä oma tyyli.

Sisältö määrittelee käyttöliittymässä esiintyvät merkkijonot, äänet ja kuvat. Yksi sisällön uudelleenmäärittelyn sovelluskohde on eri kielille kotoistettujen käyttöliittymien määritteleminen puuttumatta muihin käyttöliittymän osiin. Kielen määrittely ehdottaa sisällön uudelleenmäärittelyä eri tasoisille käyttäjäryhmille ja fyysisiä vajavaisuuksia poteville käyttäjäryhmille.

Toiminnallisuuden määritteleminen perustuu sääntöihin. Sääntö koostuu ehdosta ja joukosta toimintoja. Aina kun ehto on tosi, suoritetaan joukko ennalta määrättyjä toimintoja. Toiminnot voivat muuttaa käyttöliittymän ominaisuuksia, kutsua ohjelmointikielellä määriteltyä aliohjelmaa, kutsua taustaobjektin aliohjelmaa tai laukaista tapahtuman.

Kielessä on kuitenkin yksi selvä ongelma, nimittäin toiminnallisuuden määritteleminen. Kielioppi toiminnallisuuden määrittelylle on vaikeatajuinen. Ohjelmistosuunnittelija tuskin koskaan tulee kirjoittamaan käyttöliittymän toiminnallisuuden kuvausta UIML -kielellä. Toiminnallisuus on myös mahdollista määritellä piirtämällä vuokaavio. Perinteinen ohjelmointikieli on ainoa järkevä tapa määritellä käyttöliittymän toiminnallisuus. Merkkauskieli soveltuu tiedon määrittelemiseen ja ohjelmointikieli toiminnallisuuden määrittelemiseen.

On mahdollista toteuttaa visuaalisia suunnittelutyökaluja, joilla määritellään sovelluksen käyttöliittymä. Visuaaliset sovelluskehittimet ovat esimerkkejä tästä. Suunnittelutyökalu voi generoida käyttöliittymälle UIML -kielisen määrittelyn vastaavalla tavalla kuin visuaaliset sovelluskehittimet generoivat resurssitiedoston. Suunnittelutyökalu voi myös generoida ohjelmointikielestä UIML -kielisen toiminnallisuuden määrittelyn.

UIML -kielen määrittelydokumentti mainitsee, että yksi kielen sovelluskohde on *muunnoksen* (*transformation*) tekeminen. Käytännössä tämä tarkoittaa, että ensin on olemassa käyttöliittymän määrittely. Sopivalla työkalulla käyttöliittymän määrittelystä on mahdollista generoida:

- HTML -dokumentti näytettäväksi selaimessa
- WML -dokumentti näytettäväksi WAP -laitteessa
- C++ -rajapintoja C++ -projekteihin

UIML -kieli on suunniteltu palvelemaan ohjelmistosuunnittelijaa, ei sovelluksen käyttäjää. Kielen määrittely mainitsee korostaen, että kielen suunnittelun tavoite on esittää käyttöliittymä muodossa, joka on siirrettävissä olemassa oleviin kieliin.

UIML -kieli on ensisijaisesti ajattelutyökalu, jonka hyödyntäminen edellyttää päätöksentekoja. Päätöksen tekojen seurauksena kiinnitetään yksityiskohtia, joihin UIML -kieli ei ota kantaa. Yksi tällainen kokonaisuus on esityskirjasto. UIML -kieli on tarkoitettu ohjelmistosuunnittelijoiden käytettäväksi yhdessä muiden teknologioiden kanssa ohjelmiston suunnittelun aikana.

4.4 Käyttöliittymän merkkauskieli

Seuraavaksi esittelen kehittelemäni yksinkertaisen merkkauskielen. Sen tarkoitus on osoittaa mitä asioita käyttöliittymässä on mahdollista muuttaa merkkauskielen avulla. Pääpaino on käyttöliittymän rakenteen määrittelemisessä ja yksittäisten komponenttien kytkennässä toiminnalliseen ytimeen. Eräiden komponenttien kohdalla on tärkeämpää määritellä komponenttien hierarkia, esimerkkinä valikko. Toisten kohdalla on tärkeämpää määritellä kytkentä toiminnalliseen ytimeen, esimerkkinä syötelaatikko. Tavoitteena ei ole ollut viimeistellä ulkoasuun liittyviä seikkoja. Merkkauskieleen sisällytettyjä komponentteja ovat:

- sovellus
- käyttöliittymä
- valikko
- työkalurivi
- lapsi-ikkuna
- ponnahdusikkuna
- nappi
- syötekentät
- luetteloruutu

Merkkauskieleen sisällytettyjen komponenttien joukkoa on rajattu käsiteltävän tiedon näkökulmasta. Mukaan on otettu komponentteja, jotka esiintyvät yleisimmin käyttöliittymässä tiedonkäsittelytarkoituksessa. Merkkauskieli ei ole täydellinen rajauksen vuoksi.

Merkkauskieltä ei ole viimeistelty yhtenäiseksi. Esimerkiksi yhden komponentin määrittelyssä ei ole menetelty samoin kuin toisen komponentin tapauksessa, vaikka molemmille komponenteille olisi löydettävissä yhtenäisempi määrittelytapa.

4.4.1 Sovellus –merkitsin

Luvussa on tarkoitus esitellä merkkauskieli, jonka avulla on mahdollista määritellä käyttöliittymän rakenne. Tästä huolimatta merkkauskielen ensimmäisenä merkitsimenä esiintyy *application*. Syy tähän on se, että *application* -merkitsimien sisällä voi esitellä sovelluksen toimintaan liittyviä ominaisuuksia. Näiden ominaisuuksien arvot todennäköisesti välitetään käyttöjärjestelmälle tai ikkunointiympäristölle sovelluksen alustusvaiheessa. Esimerkki tällaisesta ominaisuudesta on sovelluksen pääikkunan koko.

```
<application interface="name1">
</application>
```

Listaus 4.3 *Application* -merkitsimien käyttö

Listauksessa 4.3 on havainnollistettu *application* -merkitsimien käyttöä. Merkitsimien välissä voi esiintyä yksi tai useampia vaihtoehtoja sovelluksen käyttöliittymän rakenteeksi. Alkumerkitsimellä on *interface* -lisäpiirre, joka ilmaisee käytettävän rakenteen nimen.

4.4.2 Käyttöliittymä -merkitsin

Ylimmällä tasolla sovelluksen käyttöliittymän määrittelyssä on pääikkunan koostumuksen määrittely. Käyttöliittymän rakenne määritellään *interface* -merkitsimien välissä. Merkitsimet voivat esiintyä ainoastaan *application* -merkitsimien välissä

```
<interface name="name1">
</interface>
```

Listaus 4.4 *Interface* -merkitsimien käyttö

Listauksessa 4.4 on havainnollistettu *interface* -merkitsimien käyttöä. Alkumerkitsimellä on *name* -lisäpiirre, jonka avulla nimetään esiteltävä rakenne. Käyttöliittymän nimeäminen mahdollistaa mieluisan käyttöliittymän valitsemisen *application* -merkitsimen yhteydessä.

4.4.3 Valikko -merkitsin

Listauksessa 4.5 on havainnollistettu valikon määrittelyä. Valikko määritellään *menu* -merkitsimien avulla. Alkumerkitsimellä on *name* -lisäpiirre, jonka avulla valikolle annetaan nimi. Aikaisemmissa luvuissa on esitelty valikoiden alkukohtia, joista valikko näytetään käyttäjälle. Alkukohtia ovat mm. napit, valikkorivit ja yksittäinen komponentti. Alkukohdasta on mahdollista viitata valikkoon nimen perusteella.

```
<menu name="name1">
</menu>
```

Listaus 4.5 Valikkojen määrittely

Valikon valinnat määritellään *menuitem* -merkitsimien avulla. Valintojen määrittelyjen tulee sijaita *menu* -merkitsimien välissä. Valintoja on seuraavanlaisia:

- komento ja komentojono
- tila
- ryhmä
- alivalikko

```
<menuitem type=command operation="name1">Komento</menuitem>
<menuitem type=script operation="name1">Komentojono</menuitem>
```

Listaus 4.6 Komennon ja komentojonon määrittely

Listauksessa 4.6 on havainnollistettu sekä komennon että komentojonon määrittelyä. Lisäpiirteen *type* arvo *command* ilmaisee, että kyseessä on komento. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation, joka suoritetaan komennon valinnan yhteydessä. Tässä tapauksessa toiminnallisen ytimen operaatio ei vastaanota parametrejä. Esimerkki toiminnallisen ytimen operaatiosta on esimerkiksi tekstinkäsittelyohjelmien tavutus.

Lisäpiirteen *type* arvo *script* ilmaisee, että kyseessä on komentojono. Komentojono on joukko toimenpiteitä, jotka käyttäjä voi määritellä tarkoitukseen kehitetyllä ohjelmointikielellä. Esimerkki tällaisesta ohjelmointikielestä esitellään seuraavassa luvussa. Lisäpiirteen *operation* arvo ilmaisee suoritettavan komentojono tunnuksen tai sijainnin.

Alku- ja loppumerkitsimien välissä esiintyvä teksti näytetään käyttäjälle valinnan otsikkona. Kaikkien valintojen tila voidaan määritellä *state* -lisäpiirteen avulla. Sallittuja arvoja ovat *enabled*, *disabled* ja *hidden*. Jos tila on *enabled*, valinta näkyy valikossa ja käyttäjä suorittaa valinnan. Jos tila on *disabled*, valinta näkyy valikossa, mutta valintaa ei voida suorittaa. Jos tila on *hidden*, valinta ei näy valikossa eikä valintaa voida suorittaa.

```
<menuitem type=checked operation="name1"</menuitem>
```

Listaus 4.7 Tilavalinnan määrittely

Listauksessa 4.7 on havainnollistettu tilavalinnan määrittelyä. Tilavalinnan avulla käyttäjä voi esimerkiksi valita onko toiminto päällä vai pois päältä. Lisäpiirteen *type* arvo *checked* ilmaisee, että kyseessä on tilavalinta. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation, joka vastaanottaa tiedon siitä onko toiminto päällä vai pois päältä.

```
<menuitem type=radiogroup operation="name1">
  <menuitem type=radioitem value=1>X</radioitem>
  <menuitem type=radioitem value=2>Y</radioitem>
  <menuitem type=radioitem value=3>Z</radioitem>
</menuitem>
```

Listaus 4.8 Valintaryhmän määrittely

Listauksessa 4.8 on havainnollistettu valintaryhmän määrittelyä. Valintaryhmään kuuluu useita valintamahdollisuuksia, joista ainoastaan yksi voi olla valittuna. Lisäpiirteen *type* arvo *radiogroup* ilmaisee, että kyseessä on valintaryhmä. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation, joka vastaanottaa tiedon käyttäjän suorittamasta valinnasta.

Valinnat määritellään valintaryhmän *menuitem* -merkitsimien välissä. Lisäpiirteen *type* arvo *radioitem* ilmaisee, että kyseessä on valintaryhmän valinta. Lisäpiirteen *value* arvo ilmaisee valinnan kokonaislukuarvon, joka välitetään toiminnallisen ytimen operaatiolle, kun valinta suoritetaan. Merkitsimien väliin jäävä teksti näytetään käyttäjälle valinnan merkkijonona.

```
<menuitem type=submenu>
  <menuitem>X</menuitem>
  <menuitem>Y</menuitem>
  <menuitem>Z</menuitem>
</menuitem>
```

Listaus 4.9 Alivalikon määrittely

Listauksessa 4.9 on havainnollistettu alivalikon määrittelyä. Lisäpiirteen *type* arvo *submenu* ilmaisee, että kyseessä on alivalikon määrittely. Alivalikon valinnat määritellään alivalikon *menuitem* -merkitsimien välissä. Esimerkissä on tarkoitus havainnollistaa alivalikon määrittelyn periaatetta, joten valinnoille ei ole määritelty lisäpiirteen *type* arvoa. Merkitsimien väliin jäävä teksti näytetään käyttäjälle valinnan merkkijonona.

4.4.4 Työkalurivi –merkitsin

Työkalurivi toimii oikotienä sovelluksen useasti käytettyihin toimintoihin. Työkalurivi on rakenteeltaan joukko yksittäisiä komponentteja, jotka on kytketty toiminnallisen ytimen operaatioihin.

```
<toolbar name="name1">
</toolbar>
```

Listaus 4.10 Työkalurivin määrittely

Listauksessa 4.10 on havainnollistettu työkalurivin määrittelyä. Työkalurivialueella voi esiintyä vaihteleva kokoonpano työkalurivejä. Tämän vuoksi on mielekästä nimetä työkalurivi. Alkumerkitsimellä on *name* lisäpiirre, jonka avulla työkaluriville annetaan nimi. Työkaluriviin kuuluvat komponentit määritellään *toolbar* -merkitsimien välissä.

4.4.5 Lapsi-ikkuna-merkitsin

Lapsi-ikkunaa käytetään jakamaan esitettävä tieto helpommin käsiteltäviin kokonaisuuksiin. Sovelluksen suorituksen aikana käyttäjä voi avata useamman kuin yhden samanlaisen ikkunan. Tämän vuoksi ikkunan määrittely tulee mieltää mallin määrittelyksi, jolle tulee antaa nimi.

```
<window name="name1">
</window>
```

Listaus 4.11 Lapsi-ikkunan määrittely

Listauksessa 4.11 on havainnollistettu lapsi-ikkunan määrittelyä. Alkumerkitsimellä on *name* lisäpiirre, jonka avulla ikkunalle annetaan nimi. Ikkunaan kuuluvat komponentit määritellään *window* -merkitsimien välissä. Ponnahdusikkunat määritellään vastaavalla tavalla hyödyntäen *dialog* -merkitsimiä.

4.4.6 Nappi –merkitsin

Nappi voi olla kytketty toiminnallisen ytimen operaatioon, komentojonoon tai valikkoon. Nappi määritellään *button* -merkitsimien avulla. Alku- ja loppumerkitsimien väliin jäävä teksti näytetään käyttäjälle napin otsikkona. Listauksessa 4.12 on havainnollistettu napin määrittelyä.

```
<button type=command operation="name1">W</button>
<button type=script operation="name2">X</button>
<button type=menu operation="name3">Y</button>
<button type=state operation="name4">Z</button>
```

Listaus 4.12 Napin määrittelyjä

Lisäpiirteen *type* arvo *command* ilmaisee, että napin painallus suorittaa toiminnallisen ytimen operaation. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation nimen. Tässä tapauksessa toiminnallisen ytimen operaatio ei vastaanota parametrejä.

Lisäpiirteen *type* arvo *script* ilmaisee, että napin painallus suorittaa komentojonon. Lisäpiirteen *operation* arvo ilmaisee komentojonon tunnuksen tai sijainnin. Seuraavassa luvussa tarkastellaan erästä komentojonojen määrittelyyn soveltuvaa ohjelmointikieltä.

Lisäpiirteen *type* arvo *menu* ilmaisee, että nappi toimii valikon alkukohtana ts. napin painallus avaa valikon käyttäjälle. Valikko on määritelty aikaisemmin *menu* -merkitsimien avulla. Lisäpiirteen *operation* arvo ilmaisee nimetyn valikon nimen.

Lisäpiirteen *type* arvo *state* ilmaisee, että kyseessä on tilanappi. Tilanappi voi olla painettuna alas tai nostettuna ylös. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation nimen. Tässä tapauksessa toiminnallisen ytimen operaatio vastaanottaa parametrinä totuusarvon.

4.4.7 Syötekenttä –merkitsin

Syötekenttä on komponentti, johon käyttäjä voi syöttää merkkijonotietoa. Listauksessa 4.13 on havainnollistettu syötekentän määrittelyä. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation nimen. Tässä tapauksessa toiminnallisen ytimen operaatio vastaanottaa parametrinä merkkijonon.

```
<inputbox operation="name1"></inputbox>
```

Listaus 4.13 Syötekentän määrittely

4.4.8 Valintaryhmä –merkitsin

Valintaryhmä on komponentti, joka koostuu useasta valinnasta. Valinnoista yksi voi olla valittu. Listauksessa 4.14 on havainnollisettu valintaryhmän määrittelyä. Valintaryhmä määritellään *radiogroup* -merkitsimien avulla. Lisäpiirteen *operation* arvo ilmaisee toiminnallisen ytimen operaation nimen. Tässä tapauksessa toiminnallisen ytimen operaatio vastaanottaa parametrinä kokonaisluvun.

```
<radiogroup operation="name1">
  <radioitem value=1>X</radioitem>
  <radioitem value=2>Y</radioitem>
  <radioitem value=3>Z</radioitem>
</radiogroup>
```

Listaus 4.14 Valintaryhmän määrittely

Valintaryhmän valinnat määritellään *radiogroup* -merkitsimien välissä. Valinta määritellään *radioitem* -merkitsimien avulla. Lisäpiirteen *value* arvo ilmaisee valinnan kokonaislukuarvon. Alku- ja loppumerkitsimien väliin jäävä teksti näytetään käyttäjälle valinnan merkkijonona. Alasvetovalikko määritellään vastaavalla tapaa hyödyntäen *dropmenu* -merkitsimiä. Vastaavasti luetteloruutu määritellään *listbox* -merkitsimillä.

4.4.9 Työnäkymä

Työnäkymät määritellään alku- ja loppumerkitsimien välissä. Merkitsimien tulee kuvata työnäkymän toiminnallisuutta tai työnäkymässä esitettävää tietoa. Merkitsimien lisäpiirteet ovat työnäkymäkohtaisia. Tässä luvussa esitetyt merkitsimet esiintyvät *application* -merkitsimien sisäpuolella.

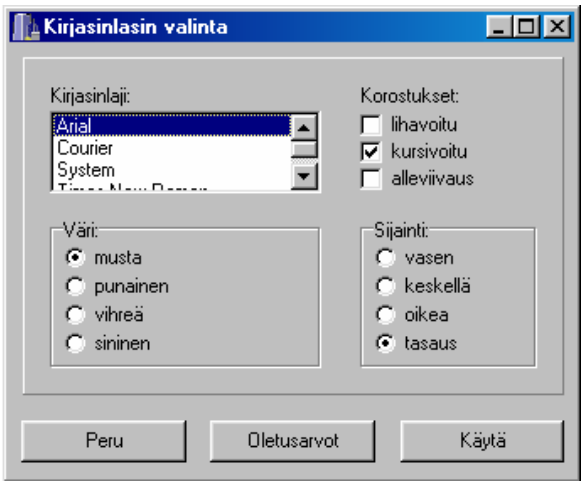
Vastaavasti työnäkymä voi koostua merkitsimistä, jotka voivat esiintyä ainoastaan kyseisen työnäkymän merkitsimien sisäpuolella. Tällaiset merkitsimet ovat sovelluskohtaisia.

4.5 Esimerkki merkkauskielen käytöstä

Tarkastelen seuraavaksi esittelemääni käyttöliittymän merkkauskieltä esimerkin avulla. Ensiksi esittelen yksittäisen ponnahdusikkunan kuvan avulla ja tämän jälkeen esittelen määrittelyn kyseiselle ponnahdusikkunalle. Tämän jälkeen ponnahdusikkunan toiminnallisuus siirretään valikkoihin ja työkaluriveihin. Havainnollistan sekä valikkoa että työkaluriviä kuvan ja määrittelyn avulla. Merkkauskielisissä määrittelyissä ei anneta ensimmäistäkään komponenttien ulkoasuun tai sijaintiin liittyvää ohjetta. Esimerkin tarkoitus on havainnollistaa seuraavia asioita:

- miten käyttöliittymän osia määritellään merkkauskielellä
- miltä määrittely mahdollisesti näyttää kuvaruudulla
- miten määritellään komponentin kytkentä toiminnallisen ytimen operaatioon
- miten vaihtoehtoinen komponentti kytketään samaan toiminnallisen ytimen operaatioon

Kuvassa 4.2 on esitetty tarkasteltava ponnahdusikkuna. Listauksessa 4.23 on havainnollistettu, miltä ponnahdusikkunan määrittely näyttää. Tulee huomata, että merkkauskielen määrittely havainnollistaa ensisijaisesti käyttöliittymän rakennetta ja komponenttien kytkentää toiminnalliseen ytimeen *operation* -lisäpiirteen avulla. Merkitsimissä ei määritellä komponenttien sijainteja eikä muitakaan ulkoasuun liittyviä seikkoja. Tämän vuoksi merkkauskielen määrittelyä vastaavan komponentin löytää helpoimmin ponnahdusikkunaa esittävästä kuvasta vertailemalla merkitsimien välissä ja käyttöliittymässä esiintyviä merkkijonoja. Yksittäisten komponenttien ja toiminnallisen ytimen operaatioiden nimet on annettu englanninkielisinä ja käyttäjälle esitettävät merkkijonot on annettu suomenkielisinä.



Kuva 4.2 Esimerkki ponnahdusikkunan ulkoasusta

Esimerkin tarkoitus on havainnollistaa, miten merkkauskielellä on mahdollista muuttaa käyttöliittymän rakenne toisenlaiseksi. Käytännössä tämä tarkoittaa uusien komponenttien määrittelyä, jotka kytketään samoihin toiminnallisen ytimen operaatioihin kuin aikaisemman käyttöliittymän komponentit. Tässä esimerkissä ponnahdusikkunan toiminnallisuus siirretään valikkoon ja työkaluriviin.

```
<dialog name="selectFont">
  <listbox operation="setTextFont">
    <listitem value=1>Arial</listitem>
    <listitem value=2>Courier</listitem>
    <listitem value=3>System</listitem>
    <listitem value=4>Times New Roman</listitem>
  </listbox>
  <checkbox operation="setBoldState">lihavoitu</checkbox>
  <checkbox operation="setItalicsState">kursivoitu</checkbox>
  <checkbox operation="setUnderlineState">alleviivaus</checkbox>
  <radiogroup operation="setTextColor">
    <radioitem value=1>musta</radioitem>
    <radioitem value=2>punainen</radioitem>
    <radioitem value=3>vihreä</radioitem>
    <radioitem value=4>sininen</radioitem>
  </radiogroup>
  <radiogroup operation="setTextAlignment">
    <radioitem value=1>vasen</radioitem>
    <radioitem value=2>keskellä</radioitem>
    <radioitem value=3>oikealla</radioitem>
    <radioitem value=4>tasaus</radioitem>
  </radiogroup>
  <button type=script operation="restoreProperties"></button>
  <button type=script operation="">setDefaultProperties</button>
  <button type=command operation="">closeDialog</button>
</dialog>
```

Listaus 4.23 Esimerkki ponnahdusikkunan määrittely

Listauksessa 4.24 on esimerkki valikon määrittelystä. Siinä esiintyy kaksi valintaryhmän määrittelyä, jotka vastaavat ponnahdusikkunan määrittelyssä esiintyneitä valintaryhmiä. Valikon valintaryhmien määrittelyssä on viittauksia toiminnallisen ytimen operaatioihin, joita on myös listauksen 4.23 määrittelyssä. Kuva 4.3 havainnollistaa, miltä valikko näyttää.

Kuva 4.3 Esimerkki valikon ulkoasu

```
<menu name="textPropertiesMenu">
  <menuitem type=radiogroup operation="setTextAlignment">
    <menuitem type=radioitem value=1>vasen</menuitem>
    <menuitem type=radioitem value=2>keskellä</menuitem>
    <menuitem type=radioitem value=3>oikea</menuitem>
    <menuitem type=radioitem value=4>tasaus</menuitem>
  </menuitem>
  <menuitem type=line></menuitem>
  <menuitem type=radiogroup operation="setTextColor">
    <menuitem type=radioitem value=1>musta</menuitem>
    <menuitem type=radioitem value=2>punainen</menuitem>
    <menuitem type=radioitem value=3>vihreä</menuitem>
    <menuitem type=radioitem value=4>sininen</menuitem>
  </menuitem>
</menu>
```

Listaus 4.24 Esimerkki ponnahdusikkunan määrittely

Listauksessa 4.25 on esimerkki työkalurivin määrittelystä. Työkalurivissä esiintyy alasvetovalikon määrittely sekä kolmen napin määrittelyt. Työkalurivin alasvetovalikko on kytketty samaan toiminnallisen ytimen operaatioon kuin ponnahdusikkunan lista. Tilanapit ovat kytketty samoihin toiminnallisen ytimen operaatioihin kuin ponnahdusikkunan valintaruudut. Kuvassa 4.4 on havainnollistettu, miltä työkalurivi näyttää.

Kuva 4.4 Esimerkki työkalurivin ulkoasu

```
<toolbar name="textPropertiesToolbar">
  <dropmenu operation="setTextFont">
    <dropitem value=1>Arial</dropitem>
    <dropitem value=2>Courier</dropitem>
    <dropitem value=3>System</dropitem>
    <dropitem value=3>Times New Roman</dropitem>
  </dropmenu>
  <button type=state operation="setBoldState">B</button>
  <button type=state operation="setItalicsState">I</button>
  <button type=state operation="setUnderlineState">U</button>
</toolbar>
```

Listaus 4.25 Esimerkki työkalurivin määrittely

5 Käyttöliittymän ohjelmointikieli

Tässä luvussa esittelen ohjelmointikielen, jolla ohjelmoidaan aikaisemmissa luvuissa mainittuja komentojonoja. Yksinkertaisimmillaan komentojono on joukko toimenpiteitä. Tässä luvussa esitelty ohjelmointikieli mahdollistaa esimerkiksi tietotyyppien, muuttujien, ehtolauseiden ja silmukkalauseiden hyödyntämisen. Komentojonossa voi siis esiintyä myös tietojen käsittelyä.

Ohjelmointikieli on tarkoitettu kieliopiltaan yksinkertaiseen tietojenkäsittelyyn ja ohjelman suorituksen ohjaamiseen. Ohjelmointikieli ei sisällä kieliopissaan ominaisuuksia, jotka suoraan mahdollistavat vuorovaikutuksen käyttäjän kanssa.

Ensimmäiseksi tarkastelen ohjelmointikielen kielioppia. Seuraavaksi esittelen laajennoksia käyttöliittymän merkkauskieleen, jotka mahdollistavat viittaukset komentojonoihin. Tämän jälkeen tarkastelen sekä toiminnallisen ytimen että sovelluskehyksen operaatiota, joita voidaan kutsua komentojonoista vuorovaikutuksen saavuttamiseksi käyttäjän kanssa. Viimeisenä vuorossa on esimerkki käyttöliittymän määrittelystä, jossa on hyödynnetty tässä luvussa esiteltyä ohjelmointikieltä.

5.1 Ohjelmointikielen kielioppi

En esittele tässä luvussa ohjelmointikielen täydellistä kielioppia, ainoastaan kielen tärkeimmät ominaisuudet. Tavoitteena on esitellä joukko ohjelmointikielen ominaisuuksia, joilla komentojonojen määrittely onnistuu.

Yksi ohjelmointikielten tärkeimmistä ominaisuuksista on ihmisten ymmärtämien tunnusten määrittely. Myös tässä luvussa esiteltävässä kielessä on mahdollista määritellä tunnuksia. Ne jakautuvat viiteen ryhmään:

- vakiot
- tietotyypit
- muuttujat
- proseduurit
- funktiot

Edellä mainittujen tunnusten lisäksi ohjelmointikielellä määritellään komentojonojen toiminnallisuus. Edellisessä luvussa esitelty käsite komentojono viittaa sekä proseduureihin että

funktioihin. Myös aliohjelma tarkoittaa sekä proseduuria että funktiota. Aliohjelmien toiminnallisuus määritellään kirjoittamalla aliohjelmiin lauseita. Kielen kielioppiin on otettu mukaan mahdollisimman pieni joukko lauseita, joilla voidaan määritellä sekä toiminnallisen ytimen operaatioiden suoritusjärjestys että yksinkertaista tietojen käsittelyä. Kielessä esiintyvät lauseet ovat:

- kutsulause
- yhdistelause
- sijoituslause
- ehtolause
- silmukkalause

Luvussa esiteltävä ohjelmointikieli muistuttaa Pascal -kieltä. Yksi tärkeimpiä ohjelmointikielen ominaisuuksia tämän pro gradu -tutkielman näkökulmasta on se, että aliohjelman kutsu voidaan ohjata toiminnallisen ytimen operaatioon. Tällaista mahdollisuutta ei ole standardissa Pascal -kielessä.

5.1.1 Valintaryhmä –merkitsin

Komentojonoissa on mahdollista määritellä symbolisia vakioita. Symbolisten vakioiden käyttö johtaa helpommin ylläpidettävään lähdekoodiin. Lähdekoodi on myös luettavampaa Listauksessa 5.1 on havainnollistettu kokonaisluku-, totuusarvo- ja merkkijonovakioiden määrittelyjä. Arvo voi koostua mistä tahansa matemaattisesta kaavasta, jonka operandit ovat vakioita.

```
Const
  Kokonaisluku   = 10;
  Totuusarvo     = false;
  Merkkijonoarvo = "string";
```

Listaus 5.1 Esimerkkejä vakioiden määrittelystä

5.1.2 Tietotyypit

Komentojonoissa on mahdollista hyödyntää tietotyyppejä. Niitä esiintyy kahdenlaisia: perustietotyyppejä ja käyttäjän määrittelemiä tietotyyppejä. Perustietotyyppejä tarvitaan ensisijaisesti tiedon

välittämisessä toiminnalliselle ytimelle. Sekä perustietotyyppejä että käyttäjän määrittelemiä tyyppejä voi hyödyntää komentojonoissa esiintyvässä tietojen käsittelyssä.

Ohjelmointikielen perustietotyypit ovat kokonaisluku (*integer*), totuusarvo (*boolean*) ja merkkijono (*string*). Perustietotyypit määräytyvät toiminnallisen ytimen operaatioiden vastaanottamien parametrien tietotyyppien perusteella. Myös liukuluku (*float*) ja merkki (*char*) ovat useissa kielissä perustietotyyppejä, mutta toiminnallisen ytimen operaatiot eivät merkkauskielen näkökulmasta katsottuna vastaanota parametreinään näitä tietotyyppejä.

```
Type
  KokonaislukuTaulukko = array[ 1 .. 100 ] of integer;
  KokonaislukuTietue = record
                         Lukumäärä: integer;
                         Kokonaisluvut: KokonaislukuTaulukko;
                       end;
```

Listaus 5.2 Esimerkkejä tietotyyppien määrittelyistä

Käyttäjän määrittelemät tietotyypit ovat taulukoita tai tietueita. Tietotyypit ovat tarkoitettu käytettäviksi komentojonoissa, jotka suorittavat tietojen käsittelyä ennen kuin kutsuvat toiminnallisen ytimen operaatioita. Listauksessa 5.2 on esimerkkejä käyttäjän määrittelemistä tietotyypeistä. Taulukot soveltuvat määrittelemään joukon muuttujia, joilla on sama tietotyyppi. Tietotyyppi voi olla perustietotyyppi tai käyttäjän määrittelemä. Tietueet soveltuvat määrittelemään joukon muuttujia, joilla ei välttämättä ole sama tietotyyppi. Tietueen muuttujia nimitetään kentiksi. Kenttien tietotyypit voivat olla perustietotyyppejä tai käyttäjän määrittelemiä tietotyyppejä.

Eräät sovelluksen toiminnallisen ytimen operaatiot voivat vastaanottaa parametreinä käyttäjän määrittelemiä tietotyyppejä. Operaatiot ovat tässä tapauksessa tarkoitettu kutsuttavaksi käyttäjän määrittelemistä komentojonoista käsin.

5.1.3 Muuttujat

Komentojonoissa on mahdollista hyödyntää muuttujia. Muuttujia tarvitaan tapauksissa, joissa käyttäjä haluaa komentojonojen suorittavan tietojen käsittelyä ennen tiedon välittämistä sovelluksen toiminnalliselle ytimelle tai käyttöliittymäkirjastolle.

```
Var
  Kokonaisluku: integer;
  Kokonaislukuja: KokonaislukuTietue;
```

Listaus 5.3 Esimerkkejä muuttujien määrittelyistä

Listauksessa 5.3 on havainnollistettu muuttujien määrittelyä. Ensimmäiseksi on esitelty kokonaislukumuuttuja, jota käytetään esimerkiksi silmukkalauseessa. Toiseksi on esitelty käyttäjän määrittelemä tietotyyppi, joka on taulukkomuuttuja.

5.1.4 Proseduurit

Tässä tapauksessa yksi ohjelmointikielen tärkeimmistä ominaisuuksista on aliohjelmien määrittely, koska merkkauskielestä viitataan ainoastaan aliohjelmiin. Proseduuri on aliohjelma, joka ei palauta arvoa. Proseduureja voi käyttää apuna merkkauskielestä viitatun komentojonon rakenteen selkeyttämisessä.

```
Procedure Suorita( Kokonaisluku: integer; Totuusarvo: boolean );
{ vakioiden, tietotyyppien ja muuttujien määrittelyjä }
Begin
  { lauseita }
End;
```

Listaukset 5.4 Esimerkki proseduurin määrittelystä

Listauksessa 5.4 on havainnollistettu proseduurin määrittelyn rakennetta. Ensimmäisellä rivillä nimetään proseduuri ja määritellään proseduurin parametrit. Tämän jälkeen voi esiintyä mielivaltainen määrä paikallisia vakioiden, tietotyyppien ja muuttujien määrittelyjä. Lopuksi yhdistelauseen sisäpuolelle kirjoitetaan lauseet, joista proseduuri koostuu.

5.1.5 Funktiot

Listauksessa 5.5 on havainnollistettu funktion määrittelyn rakennetta. Aivan kuten proseduurien tapauksessa ensimmäisellä rivillä nimetään funktio, ja määritellään funktion parametrit. Lisäksi ensimmäisellä rivillä määritellään funktion palauttaman arvon tietotyyppi. Tämän jälkeen voi esiintyä mielivaltainen määrä paikallisia vakioiden, tietotyyppien ja muuttujien määrittelyjä. Lopuksi yhdistelauseen sisäpuolelle kirjoitetaan lauseet, joista funktio koostuu. Yhdistelauseen sisäpuolella voi esiintyä sijoitus funktion tunnukseen, mikä ilmoittaa funktion palauttaman arvon.

```
Function Laske( Arvo1, Arvo2: integer ): integer;
{ vakioiden, tietotyyppien ja muuttujien määrittelyjä }
Begin
  { lauseita }
  Laske := arvo;
End;
```

Listaukset 5.5 Esimerkki funktio määrittelystä

5.1.6 Sijoituslause

Ohjelmointikielen tavoitteena on mahdollistaa käyttäjän määrittelemä tietojen käsittely komentojonoissa. Tähän tarkoitukseen tarvitaan sijoituslausetta. Se on tärkeä komentojonoille, joissa esiintyy tietojen käsittelyä, koska olennainen osa tietojen käsittelyä on arvon sijoittaminen muuttujaan.

```
Muuttuja := arvo;
```

Listaus 5.6 Esimerkki arvon sijoituksesta muuttujaan

Listauksessa 5.6 on esimerkki sijoituslauseesta. Sijoitusoperaattorin := vasemmalla puolella esiintyy kohdemuuttujan tunnus ja oikealla puolella matemaattinen kaava. Kohdemuuttuja voi olla viittaus tavalliseen muuttujaan, taulukon alkioon tai tietueen kenttään. Arvo voi koostua viittauksesta muuttujiin, vakioista, funktion kutsusta yms. Matemaattisen kaavan kielioppia ei tässä selitetä.

5.1.7 Aliohjelman kutsulause

Yksi ohjelmointikielen käyttötarkoituksen kannalta tärkeimpiä ominaisuuksia on kutsulause. Määrittelemällä aliohjelma ja kutsumalla aliohjelmaa kutsulauseen avulla, on mahdollista selkeyttää komentojonon rakennetta. Kutsulauseella ohjataan ohjelman suoritus toiminnallisen ytimen operaatioon.

```
Suorita;
Muuttuja := Laske( 1, 2 );
```

Listaus 5.7 Esimerkkejä aliohjelmien kutsulauseista

Listauksessa 5.7 on esimerkkejä aliohjelmien kutsulauseista. Ensimmäinen kutsulause ohjaa ohjelman suorituksen aliohjelmaan nimeltä *Suorita*, joka ei vastaanota parametrejä. Toinen esimerkki havainnollistaa, miten funktiota voidaan kutsua sijoituslauseesta. Sijoituslauseesta kutsutaan funktiota nimeltä *Laske*, joka vastaanottaa kaksi parametriä.

5.1.8 Ehtolause

Ohjelmointikielen tavoitteena on mahdollistaa käyttäjän määrittelemä tietojen käsittely komentojonoissa. Myös ehtolause on tärkeä osa komentojonoissa tapahtuvaa tietojen käsittelyä. Ehtolauseiden avulla hallitaan ohjelman suorituksen etenemistä.

```
If Ehto then Lause1;
If Ehto then Lause1 else Lause2;
```

Listaus 5.8 Esimerkkejä ehtolauseista

Listauksessa 5.8 on havainnollistettu ehtolauseiden rakenne. Ensimmäisessä esimerkissä *Lause1* suoritetaan, jos *Ehto* on tosi. Suoritettavaksi lauseeksi kelpaa mikä tahansa ohjelmointikielen lause. Ehdoksi kelpaa mikä tahansa totuusarvon saava matemaattinen kaava. Toisessa esimerkissä *Lause1* suoritetaan, jos *Ehto* on tosi, muutoin suoritetaan *Lause2*.

5.1.9 Silmukkalause

Ohjelmointikielen tavoitteena on mahdollistaa käyttäjän määrittelemä tietojen käsittely komentojonoissa. Myös silmukkalause on tärkeä osa komentojonoissa tapahtuvaa tietojen käsittelyä. Silmukkalauseella voidaan määrätä lause suoritettavaksi mielivaltaisen monta kertaa.

```
While Ehto do Lause;
```

Listaus 5.9 Esimerkki silmukkalauseesta

Listauksessa 5.9 on havainnollistettu silmukkalauseen rakennetta. Toistettavaksi lauseeksi *Lause* kelpaa mikä tahansa ohjelmointikielenlause. Ehdoksi kelpaa mikä tahansa totuusarvon saava matemaattinen kaava. Lausetta toistetaan niin kauan kuin ehto on tosi.

5.1.10 Yhdistelause

Yhdistelauseen tarkoitus on mahdollistaa useamman kuin yhden lauseen esiintyminen tilanteessa, jossa ainoastaan yksi lause voi esiintyä. Esimerkiksi ehto- ja silmukkalauseille on mahdollista määritellä suoritettavaksi ainoastaan yksi lause. Yhdistelauseella itsellään ei ole toimintaa ohjelman suorituksen kannalta.

```
Begin
  { lauseita }
End;
```

Listaus 5.10 Esimerkki yhdistelauseesta

Listauksessa 5.10 on havainnollistettu yhdistelausetta. Yhdistelauseen sisäpuolella voi esiintyä sijoitus-, ehto- ja silmukkalauseita sekä aliohjelman kutsuja. Yhdistelauseessa esiintyvissä ehto- ja silmukkalauseissa voi esiintyä lisää yhdistelauseita.

5.2 Viittaukset käyttöliittymän merkkauskielestä

Merkkauskieltä käsitelleessä luvussa esiteltiin komponentteja, joiden lisäpiirteen *type* yksi mahdollinen arvo on *script*. Arvo *script* tarkoittaa, että tiettyjen ehtojen vallitessa komponentti kutsuu komentojonoa toimenpiteen suorittamiseksi. Komentojonon suorittamat toimenpiteet ovat käyttäjän määrittelemiä. Komponentin lisäpiirteen *operation* arvo on viittaus suoritettavaan komentojonoon. Lisäpiirteen *operation* arvo voi olla ohjelmointikielellä määritellyn aliohjelman nimi. Komentojonoissa esiintyvien aliohjelmien tunnukset täytyy julkaista käytettäväksi merkkauskielestä käsin ennen kuin viittaus on mahdollista.

Komponentin lisäpiirteen *type* yksi mahdollinen arvo on *command*. Arvo *command* tarkoittaa, että tiettyjen ehtojen vallitessa komponentti kutsuu toiminnallisen ytimen operaatiota toimenpiteen suorittamiseksi. Toiminnallisen ytimen operaation suorittamat toimenpiteet ovat sovelluskohtaisia. Lisäpiirteen *operation* arvo on viittaus suoritettavaan operaatioon. Toiminnallisen ytimen operaatioiden nimet täytyy julkaista käytettäväksi ohjelmointi- ja merkkauskielestä käsin ennen kuin viittaus on mahdollista.

5.3 Käyttöliittymäkirjaston operaatiot

Luvussa 4 esitelty merkkauskieli ei ole täydellinen. Esimerkiksi lapsi- ja ponnahdusikkunoita on mahdollista määritellä, mutta toistaiseksi on jäänyt epäselväksi, miten määritellään tilanteet, joissa ikkunat avautuvat. Merkkauskieltä on mahdollista laajentaa sisältämään kytkennät ikkunoihin, jolloin esimerkiksi valikon valinta avaa ikkunan. Toinen lähestymistapa on mahdollistaa ikkunoiden avaaminen komentojonoista käsin. Esimerkiksi ponnahdusikkunoiden avaaminen ja sulkeminen voidaan hoitaa komentojonoista käsin operaatioilla *OpenDialog* ja *CloseDialog,* jotka ohjaavat ohjelman suorituksen käyttöliittymäkirjaston operaatioihin, joiden vastuulla on ikkunoiden avaaminen ja sulkeminen.

Lisäksi jokaiselle käyttöliittymäkirjaston komponentille voi olla operaatiot ominaisuuksien lukemiselle ja asettamiselle. Esimerkiksi ikkunoilla on otsikko. Otsikon merkkijonon voi palauttaa *getTitle* operaatiolla ja asettaa *setTitle* operaatiolla. Vastaavanlaisia operaatioita voi olla myös komponentin sijainnin ja kirjasinlajin ominaisuuksille. Vastaavalla lähestymistavalla on mahdollista asettaa ohjelmointikielellä määritelty komentojono komponentin kytkennäksi.

5.4 Toiminnallisen ytimen operaatiot

Luvussa 5.5 laajennetaan edellisen luvun esimerkkiä 4.4. Tämän vuoksi tarkastelen tekstinkäsittelysovelluksen toiminnallisen ytimen operaatioita. Tekstinkäsittelysovelluksen toiminnallisessa ytimessä on ainakin seuraavat komentojonojen kirjoittamisen kannalta mielekkäät operaatiot: kohdistimen liikuttaminen, valinnan aloittaminen ja lopettaminen, kopioi, leikkaa ja liitä.

Dokumentin voidaan ajatella koostuvan taulukosta merkkejä. Jokaisella merkillä on useita parametrejä, joita ovat mm. merkki, kirjasinlaji, lihavointi, kursivointi, alleviivaus ja väri. Kohdistinta liikuttamalla valitaan muokattava taulukon alkio. Kohdistin ilmoittaa kohdan, jonne seuraava käyttäjän syöttämä merkki sijoitetaan. Seuraavalle merkille, jonka käyttäjä syöttää on omat erilliset parametrinsä. Niitä muokataan operaatioden *setTextFont*, *setBoldState*, *setItalicsState*, *setUnderlineState*, *setTextColor* ja *setTextAlignment* avulla. Liikutettaessa kohdistinta dokumentin sisällä seuraavan syötettävän merkin parametrit muuttuvat, kunnes edellä mainittuja operaatioita kutsutaan asettamaan uudet parametrit. Kutsumalla edellä mainittuja operaatioita komentojonoista käyttäjä voi määritellä omia tekstinmuokkausoperaatioita, joita suoritetaan esimerkiksi työkalurivin kautta tai pikanäppäintä painamalla.

5.5 Esimerkki ohjelmointikielen käytöstä

Tarkastelen seuraavaksi yksinkertaista esimerkkiä ohjelmointikielen käytöstä komentojonojen määrittelyssä. Tämä esimerkki laajentaa aikaisemmassa luvussa esiintynyttä esimerkkiä 4.4. Listauksessa 4.23 on ponnahdusikkunan merkkauskielinen määrittely. Kuvassa 4.2 on havainnollistettu miltä ponnahdusikkuna näyttää kuvaruudulla.

Listauksessa 5.11 on määritelty kaksi komentojonoa, joihin viitataan listauksesta 4.23. Esimerkki on puutteellinen. Oletetaan tämän vuoksi, että ponnahdusikkunan avautuessa *Asetukset* -muuttujaan tallentuu automaattisesti kirjasinlajin nykyiset parametrit. Jos käyttäjä painaa *Peru*-nappia, ohjelman suoritus ohjautuu *restoreProperties* -aliohjelmaan, joka kumoaa ikkunassa tehdyt muutokset palauttamalla *Asetukset* -muuttujaan taltioidut kirjasinlajin asetukset. Jos käyttäjä painaa *Oletusarvot*-nappia, ohjelman suoritus ohjautuu *setDefaultProperties* -aliohjelmaan, joka asettaa kirjasinlajille oletusarvot. Ponnahdusikkunan *Käytä*-nappi on kytketty toiminnallisen ytimen *closeDialog* -operaatioon, joka sulkee ponnahdusikkunan. Tämän napin painallus ei aseta kirjasinlajin parametrejä, koska ponnahdusikkunassa esiintyvien komponenttien muokkaaminen aiheuttaa toiminnallisen ytimen operaation kutsumisen, joka asettaa parametrin.

Komentojonon määrittelyssä esiintyy aliohjelmakutsu *closeDialog* -nimiseen aliohjelmaan, joka sulkee ponnahdusikkunan. Aliohjelma on osa käyttöliittymäkomponenttikirjastoa, jonka operaatioita on mahdollista kutsua kuten toiminnallisen ytimen operaatioita.

```
Type
   AsetusTietue = record
                    KirjasinLaji: integer;
                    Lihavoitu: boolean:
                    Kursiivi: boolean;
                    Alleviivaus: boolean;
                    Vari: integer;
                    Sijainti: integer;
                  End;

Var Asetukset: AsetusTietue;

Procedure onShow;
Begin
   Asetukset.KirjasinLaji := getTextFont;
   Asetukset.Lihavoitu := getBoldState;
   Asetukset.Kursiivi := getItalicsState;
   Asetukset.Alleviivaus := getUnderlineState;
   Asetukset.Vari := getTextColor;
   Asetukset.Sijainti := getTextAlignment;
 End;

 Procedure restoreProperties;
 Begin
   SetTextFont( Asetukset.KirjasinLaji );
   SetBoldState( Asetukset.Lihavoitu );
   SetItalicsState( Asetukset.Kursiivi );
   SetUnderlineState( Asetukset.Alleviivaus );
   SetTextColor( Asetukset.Vari );
   SetTextAlignment( Asetukset.Sijainti );
   CloseDialog;
 End;

 Procedure setDefaultProperties;
 Begin
   SetTextFont( 3 );
   SetBoldState( false );
   SetItalicsState( false );
   SetUnderlineState( false );
   SetTextColor( 1 );
   SetTextAlignment( 1 );
   CloseDialog;
 End;
```

Listaus 5.11 Esimerkki komentojonon määrittelystä

6 Toiminnallinen ydin

Tämän luvun tarkoitus on yhdistää aiemmissa luvuissa esitellyt asiat toimivaksi kokonaisuudeksi. Käyttöliittymien tutkiminen johti sisäisen esitystavan muodostamiseen. Merkkauskieltä käsittelevässä luvussa esiteltiin kielioppi, jonka avulla on mahdollista määritellä käyttöliittymien rakenne. Ohjelmointikieltä käsiteltävässä luvussa vietiin ajatus muokattavasta käyttöliittymästä äärimmäisyyteen. Tässä luvussa esitellään sovelluksen rakenne, jota noudattamalla on mahdollista toteuttaa tässä pro gradu -tutkielmassa käsitellyt asiat.

Ensimmäiseksi tarkastelen toiminnallisen ytimen rakennetta. Ytimen rakenteen tarkastelu koostuu sekä alustustoimenpiteitä että käyttäjän antamien syötteiden käsittelyä suorittavien komponenttien tarkastelusta. Viimeiseksi käsittelen suorituksen etenemistä sovelluksessa, joka toteuttaa tässä pro gradu -tutkielmassa esitellyt asiat. Suorituksen etenemisen tarkastelu koostuu sovelluksen alustustoimenpiteistä ja käyttäjä antamien syötteiden käsittelystä.

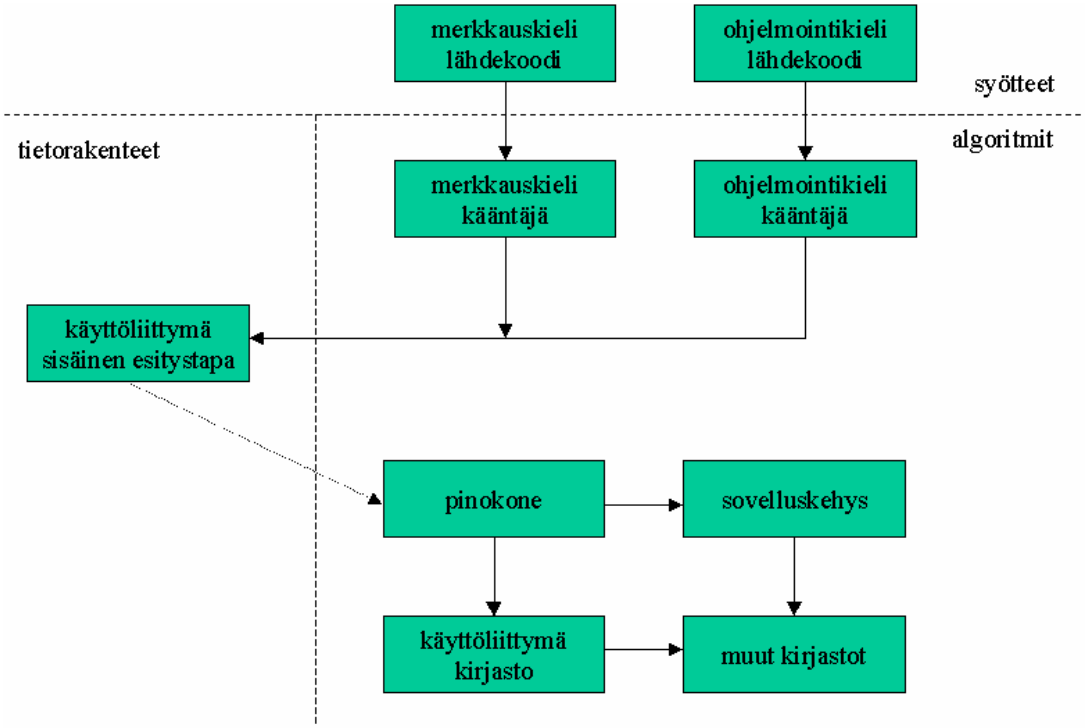
6.1 Ytimen rakenne

Kuvassa 6.1 on havainnollistettu sovelluksen rakennetta, joka toteuttaa pro gradu -tutkielmassani kuvatun käyttöliittymien muokattavuuden. Kuva on jaettu kolmeen osaan: syötteisiin, algoritmeihin ja tietorakenteisiin. Syötteet koostuvat ainoastaan käyttöliittymän määrittelyistä. Alustustoimenpiteistä vastaavat algoritmit käsittelevät syötteet ja rakentavat sovelluksen tarvitsemat tietorakenteet. Käyttöliittymän sisäinen esitystapa on tietorakenne, jonka avulla käyttäjä laukaisee sovelluksen toiminnan kannalta mielekkäitä operaatioita. Sovelluksen kannalta mielekkäät operaatiot kuuluvat kuvassa algoritmien joukkoon.

Merkkauskielen kääntäjän vastuulla on rakentaa sisäinen esitystapa käyttöliittymän määritysten perusteella. Käyttöliittymän määritys on laadittu esimerkiksi luvussa 4 kuvatun käyttöliittymän merkkauskielen avulla.

Käyttöliittymäkirjasto on lähinnä joukko yksittäisiä käyttöliittymäkomponentteja ja niiden toiminnallisuuden määrittelyjä. Käyttöliittymäkirjaston vastuulla on lisäksi sovelluksen käyttöliittymän sisäisen esitystavan määritteleminen. Kirjaston vastuulla on myös esitystavan ylläpitäminen sovelluksen suorituksen aikana.

Ohjelmointikielen kääntäjän vastuulla on lähdekoodin kääntäminen. Lähdekoodissa on käyttäjän määrittelemiä komentojonoja, joita suoritetaan ennalta määrättyjen ehtojen täyttyessä. Käännettävän ohjelmointikielen kieliopilla ei ole merkitystä. Kääntäjän ensisijainen tehtävä on käskyvirran generoiminen pinokoneelle.



Kuva 6.1 Sovelluksen rakenne

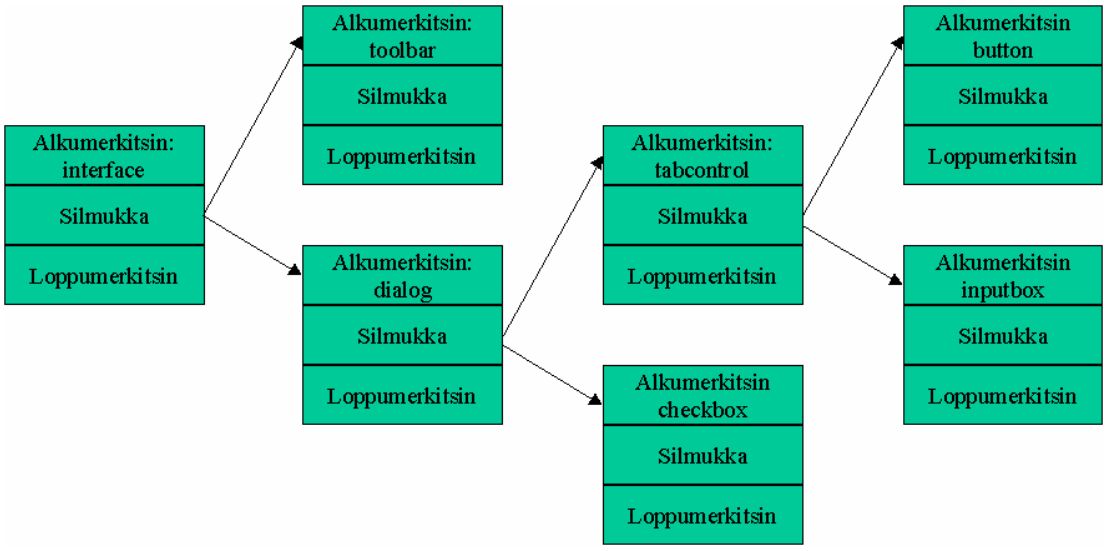
Pinokoneen vastuulla on komentojonojen suorittaminen. Komentojonojen käännös tuottaa pinokoneessa suoritettavan ohjelman, joka voi sisältää sekä laskentaa että ohjelman suorituksen ohjausta. Pinokonetta voi laajentaa käskykannoilla, jotka mahdollistavat ulkopuolisten kirjastojen hyödyntämisen komentojonoista.

Sovelluskehyksen vastuulla on ongelman ratkaisu, jota varten sovellus on laadittu. Jos sovelluskehys suunnitellaan selviytymään ongelman ratkaisusta ainoastaan tiedon käsittelyn tasolla, muodostaa sovelluskehys uudelleenkäytettävän kokonaisuuden.

Kuvassa 6.1 on havainnollistettu muita kirjastoja. Tässä muut kirjastot tarkoittavat mitä tahansa kokonaisuutta, jota kuvassa ei ole täsmällisesti esitetty. Voi olla, että muiden kirjastojen operaatioiden kutsuminen on mielekästä ainoastaan sovelluskehyksestä käsin. Harvemmin yksi sovellus ratkaisee useamman kuin yhden ongelman.

6.1.1 Merkkauskielen kääntäjä

Merkkauskielen kääntäjä koostuu sekä tiedostojen että merkkijonojen käsittelystä. Kääntäjä on jaettu useaan aliohjelmaan, joiden tehtävänä on merkkauskielisen lähdekoodin sisällön selvittäminen. Aliohjelmat noudattavat samaa kaavaa. Aliohjelma lukee alkumerkitsimen sekä lisäpiirteet ja luo dynaamisen olion kuvaamaan juuri luettua tietoa. Tämän jälkeen päädytään silmukkaan, jossa käsitellään alku- ja loppumerkitsimien välissä olevat merkitsimet. Silmukka tunnistaa ainoastaan rajatun joukon merkitsimiä ja ohjaa ohjelman suorituksen aliohjelmaan, joka osaa selvittää kohdatun merkitsimen kuvaaman sisällön. Silmukasta päästään ulos vasta, kun alkumerkitsintä vastaava loppumerkitsin on kohdattu. Aliohjelmissa on myös ohjelmakoodia käyttöliittymän sisäisen esitystavan rakentamiseksi.



Kuva 6.2 Merkkauskielen kääntäjän rakenne

Kuvassa 6.2 on havainnollistettu aliohjelmien rakennetta ja aliohjelmien kutsujärjestystä. Luvussa 4 esiteltiin käyttöliittymän merkkauskieltä. Luvussa ei kuitenkaan tarkkaan määritelty, missä merkitsimet voivat esiintyä. Ei ole mielekästä määritellä valintaruutuja, nappeja tai syötekenttiä välittömästi *interface* -merkitsimien sisällä. Edellä mainitut komponentit voivat esiintyä ponnahdusikkunoissa , joten merkitsimet voivat esiintyä *dialog* -merkitsimien sisäpuolella.

Toimivan merkkauskielen kääntäjän toteuttaminen edellyttää käyttöliittymäkirjaston kiinnittämistä. Olio-ohjelmoinnin suunnittelumalleja apuna käyttäen on kuitenkin mahdollista laatia ratkaisu, jossa käyttöliittymäkirjasto on vaihdettavissa kokoavan uudelleenkäytön avulla. Tässä tapauksessa *kääntäjä* -olio saa parametrinä viittauksen dynaamisesti luotuun *rakentaja* -olioon [6, ss. 97-106],

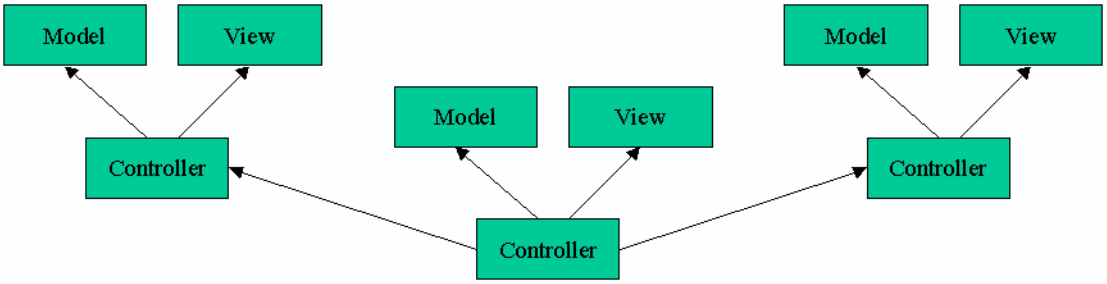
jolle kääntäjä välittää tiedon kohdatuista merkitsimistä ja niiden lisäpiirteistä. Käännöksen päätteeksi käyttöliittymän sisäinen esitystapa on luettavissa rakentaja -oliolta.

6.1.2 Käyttöliittymäkirjasto

Käyttöjärjestelmä lähettää sovelluksille oheislaitteilla generoituja tapahtumia. Esimerkkejä tällaisista tapahtumista ovat näppäimen painallus tai hiiren liikuttaminen. Esimerkiksi Microsoft Windows -käyttöjärjestelmässä hiiren liikuttaminen valikon päällä muuttaa valikon ulkoasua siten, että hiiren kohdalla oleva valikon valinta korostetaan sinisellä taustavärillä. Tässä tapauksessa tapahtuma käsitellään käyttöliittymäkirjaston sisäpuolella. Tapahtumien käsittelyn vastuulla on käyttöliittymän sisäisen esitystavan eheyden ylläpitäminen.

Tapahtumien käsittely saattaa muuttaa komponenttien tilaa. Ennalta määrättyjen ehtojen täyttyessä komponentti suorittaa toimenpiteen, johon käyttöliittymäkirjaston toteutuksessa ei ole otettu kantaa. Tällöin komponentin kytkentä määrää minne ohjelman suoritus ohjautuu.

Sovelluksen käyttöliittymän sisäinen esitystapa on yksinkertainen: tietorakenteeksi riittää puurakenne. Tästä huolimatta käyttöliittymäkirjaston suunnittelijoilla on paljon vaihtoehtoja komponenttien toteutuksessa: yksittäinen komponentti voidaan kapsuloida yhteen luokkaan tai esittää kuvassa 6.3 havainnollistetun *Controller-Model-View* -periaatteen avulla. *Controller* -olio on vastuussa tapahtumien käsittelystä ja saattaa komponentin tila ajan tasalle. *View*-olio on vastuussa komponentin esittämisestä kuvaruudulla *Model*-olio sisältää kuvaruudulla esitettävän tiedon. *Controller-Model-View* -lähestymistavan etuna on, että mikä tahansa kolmesta oliosta on vaihdettavissa ohjelman suorituksen aikana vaikuttamatta kahteen muuhun olioon. Vaihtamalla sekä *Controller*- että *View* -oliot kaikille käyttöliittymän sisäisessä esitystavassa esiintyville komponenteille, on mahdollista toteuttaa käyttöliittymään erilaisia ulkoasuja ja tuntumia. Ulkoasu viittaa komponentin esitystapaan kuvaruudulla. Tuntuma viittaa tapaan, jolla käyttäjä käsittelee komponenttia kuvaruudulla. Yleensä sekä ulkoasu että tuntuma vaihdetaan samanaikaisesti. *Ulkoasu ja tuntuma* (*look-and-feel*) on nimitys ikkunointiympäristön toteutuksessa kiinnitetylle tavalle hoitaa vuorovaikutus käyttäjän kanssa. Vuorovaikutuksessa on huomioitu ainoastaan kuvaruutu, näppäimistö ja hiiri. Esimerkiksi Microsoftin käyttöjärjestelmissä hyödynnetään Windows ulkoasua ja tuntumaa.

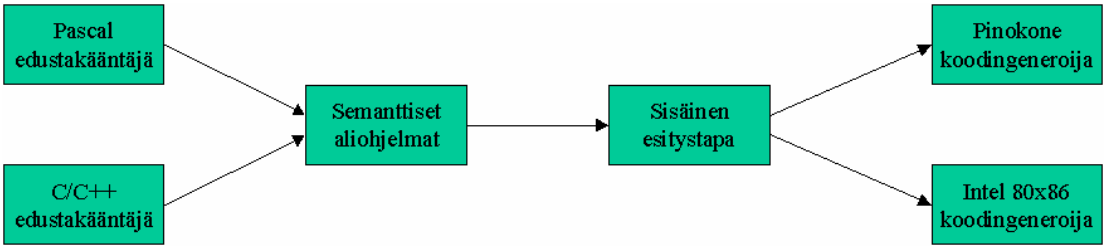


Kuva 6.3 Controller-Model-View-malli

Lähestymistavasta huolimatta käyttöliittymä esitetään puurakenteen avulla. Sisäisen esitystavan läpikäymiseen riittää puu-tietorakenteelle laaditut algoritmit.

6.1.3 Ohjelmointikielen kääntäjä

Ohjelmointikielen kääntäjä sisältää muun muassa tiedostojen ja merkkijonojen käsittelyä. Olio-ohjelmointia, tietorakenteita ja algoritmeja sekä suunnittelumalleja hyödyntämällä ohjelmointikielen toteutus saadaan muistuttamaan sovelluskehystä. Kääntäjän sovelluskehys koostuu *edustakääntäjästä* (*front-end*), *semanttisista aliohjelmista* (*semantic routines*), *sisäisestä esitystavasta* (*internal representation*) ja *koodingeneroijasta* (*back-end*) [5, ss. 8-12]. Luvussa 5 esiteltyyn ohjelmointikieleen on toteutettavissa kääntäjä edellä mainittua rakennetta noudattaen. Barron esittelee kirjassaan [2, ss. 63-82] yksityiskohtaisemmin Pascal-kääntäjän toteuttamiseen liittyviä yksityiskohtia. Kuva 6.4 havainnollistaa kääntäjän rakennetta.



Kuva 6.4 Ohjelmointikielen kääntäjän rakenne

Edustakääntäjä on vaihdettavissa oleva kokonaisuus, joka ymmärtää yhden kielen kieliopin. Edustakääntäjä muodostaa lähdekoodista *alkioita* (*token*). Jos jono alkioita muodostaa kieliopillisesti mielekkään kokonaisuuden, edustakääntäjä välittää tiedon kohtaamastaan kieliopin rakenteesta semanttisille aliohjelmille. Ennen kuin edustakääntäjän toteutuksen suunnittelu on mahdollista on määrättävä käännettävän ohjelmointikielen kieliopin yksityiskohdat.

Semanttiset aliohjelmat suorittavat käännettävälle ohjelmalle *semanttisen tarkistuksen* (*semantic analysis*). Esimerkki semanttisten aliohjelmien suorittamasta toimenpiteestä on sijoituslauseen yhteydessä muuttujan ja sijoitettavan arvon tietotyyppien yhteensopivuuden tarkistaminen. Jos semanttisten aliohjelmien vastaanottama kieliopillisesti sallittu rakenne muodostaa myös semanttisesti sallitun rakenteen, voidaan kyseinen rakenne lisätä osaksi käännetyn ohjelman sisäistä esitystapaa.

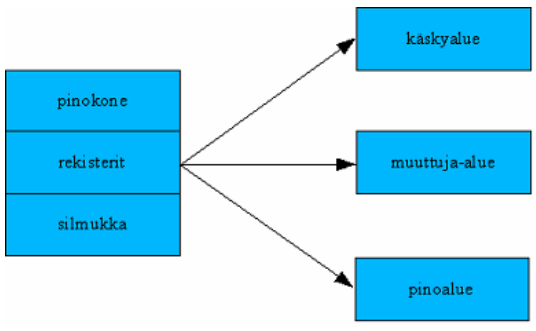
Sisäinen esitystapa on tietorakenne, joka kuvaa käännetyn ohjelman rakenteen. Käännetty ohjelma on sekä kieliopillisesti että semanttisesti virheetön. Sisäiseen esitystapaan on mahdollista soveltaa algoritmeja, jotka optimoivat esitettyä ohjelmaa. Semanttiset aliohjelmat ja käännetyn ohjelman sisäinen esitystapa muodostavat kääntäjän sovelluskehyksen.

Koodingeneroija muodostaa sisäisestä esitystavasta suoritettavaa koodia yhdelle alustalle. Alusta voi olla suoritin tai kuvitteellinen pinokone. Koodingeneroija voi sisältää myös algoritmeja, jotka suorittavat optimointia konekielitasolla.

6.1.4 Pinokone

Pinokoneen tarkoitus on suorittaa komentojonoja. Ohjelmointikielen kääntäjän koodingeneroija on vastuussa pinokoneessa suoritettavan ohjelman generoimisesta. Kuvassa 6.5 on havainnollistettu pinokoneen rakennetta.

Pinokone-olio sisältää kenttiä, jotka toimivat pinokoneen rekistereinä. Pinokoneen toiminnan kannalta tärkeitä ovat viittaukset käsky-, pino- ja muuttuja-alueisiin. Pinokone-olio sisältää myös silmukan, joka suorittaa käskyjä, kunnes kohtaa aliohjelmasta paluun suorittavan käskyn. Tällöin ohjelman suoritus palaa pinokonetta kutsuneen tahon suorittamiseen.



Kuva 6.5 Pinokoneen rakenne

Käskyalue sisältää käskyjä, joita pinokone suorittaa. Käskyalue on toteutettavissa dynaamisesti varatulla taulukkomuuttujalla, joka koostuu kokonaisluvuista. Pinokone-olio sisältää *käskyosoittimen* (*instruction pointer*), jonka ilmaisee seuraavan suoritettavan käskyn sijainnin käskyalueella. Käskyn *toimituskoodi* (*operation code*) on kaksiosainen: se ilmaisee sekä käskykannan että käskyn. Toimituskoodia voi seurata käskyn edellyttämä vakio tai viittaus yleiseen tai paikalliseen muuttujaan.

Pinoalue sisältää pinokoneen suorittamien käskyjen operandeja. Pinoalueelle tallennetaan komentojonojen paikallisia muuttuja ja matemaattisen kaavojen välituloksia. Pinoalue on toteutettavissa dynaamisesti varatulla taulukkomuuttujalla, joka koostuu kokonaisluvuista. Pinokone-olio sisältää *pino-osoittimen* (*stack pointer*), joka on viittaus seuraavaan operandiin. Pino toimii siten, että viimeiseksi pinoon lisätty arvo on luettavissa pinosta ensimmäisenä. Esimerkiksi yhteenlaskukäsky hakee pinosta kaksi operandia, laskee ne yhteen ja työntää yhteenlaskun tuloksen pinoon.

Muuttuja-alue sisältää komentojonoissa määritellyt yleiset muuttujat. Tässä pro gradu -tutkielmassa esitettyjen komentojonojen näkökulmasta muuttuja-alueeseen taltioidaan tietoa, jonka halutaan jaettavan tai säilyvän kahden kytkennän laukaisun välillä. Eräät pinokoneen käskyt lisäävät muuttujan arvon pinoon tai lukeva pinosta arvon muuttujaan.

6.1.5 Sovelluskohtaiset operaatiot

Jos sovellus noudattaa kuvassa 6.1 esitettyä rakennetta, muodostaa sovelluksen ratkaiseman ongelman toteutus sovelluskehyksen. Ongelman ratkaisu voi hyödyntää olio-ohjelmointia, tietorakenteita ja algoritmeja sekä suunnittelumalleja muunneteltavuuden saavuttamiseksi. Ongelman ratkaisu voi hyödyntää muita kirjastoja. Sovelluskehys ratkaisee ainoastaan tiedonkäsittelyyn liittyvän ongelman.

Sovelluskehys voidaan kapsuloida käyttäen *façade*-suunnittelumallia [6, ss.185-194]. Sen avulla määritellään rajapinta, jonka kautta sovelluskehys on helpommin käytettävissä. Jos sovelluskehyksen operaatioita on tarkoitus kutsua komentojonoista käsin, niin rajapinnan operaatiot toimivat kandidaatteina pinokoneen käskykannan käskyiksi.

6.1.6 Muut kirjastot

Kirjastot ovat liitettävissä sovellukseen suunnittelumallien avulla. Tästä lähestymistavasta on esimerkkinä façade -suunnittelumalli. Käyttöliittymäkirjasto voi hyödyntää kirjastoja käyttöliittymäkomponenttien ulkoasun piirtämiseksi. Todennäköisemmin sovelluskehys hyödyntää kirjastoja jonkin pienemmän ongelman ratkaisemiseksi. Pinokonetta voi laajentaa käskykannoilla, jotka ohjaavat ohjelman suorituksen mihin tahansa sovelluksen hyödyntämään kirjastoon.

6.2 Alustustoimenpiteet

Tämän pro gradu -tutkielman näkökulmasta sovelluksen tärkeimmät alustustoimenpiteet ovat komentojonojen ja käyttöliittymän määrittelyn kääntäminen. Käännettävät tiedostot asetetaan asetustiedostossa, jonka rakenne kuvattiin luvussa 4. Jos asetustiedostossa ei ole asetettu komentojonoja tai käyttöliittymän määrittelyjä, tulisi sovelluksen hyödyntää oletusarvoista käyttöliittymää.

Komentojonoista voi olla viittauksia käyttöliittymän merkkauskielellä määriteltyihin nimettyihin komponentteihin, mutta näillä viittauksilla on merkitystä vasta komentojonon suorituksen yhteydessä. Merkkauskielen määrittelyistä voi olla viittauksia komentojonoihin. Nämä viittaukset olisi hyvä saada aikaan välittömästä käyttöliittymän määrittelyjen käännöksen yhteydessä. Tämän vuoksi ensimmäisenä alustustoimena sovellus kääntää komentojonot. Komentojonojen käännöksen yhteydessä on mahdollista rakentaa *hajautettu taulukko* (*hash table*) [8, ss. 425-457], johon taltioidaan komentojonojen tunnuksia ja osoitteita pinokoneen käskyalueella. Komentojonojen käännöksen tuloksena syntyy yksi pinokone ja yksi hajautettu taulukko.

Seuraavana alustustoimenpiteenä sovellus suorittaa käyttöliittymän määritysten kääntämisen. Tässä yhteydessä rakennetaan käyttöliittymän sisäinen esitystapa. Jos sisäisen esitystavan muodostaminen edellyttää käyttöliittymäkirjaston alustamista, tulee alustaminen tehdä ennen käännöksen suorittamista. Merkkauskielen käännöksen yhteydessä haetaan edellisessä kappaleessa mainitusta hajautetusta taulukosta komentojonojen osoitteita tunnuksien perusteella.

Tämän jälkeen sovelluksen tulisi alustaa muut kirjastot, sovelluskehys ja suorittaa mahdolliset muut alustustoimenpiteet. Tämän jälkeen ohjelman suoritus voi edetä käsittelemään käyttäjän antamia syötteitä.

6.3 Ohjelman suoritus

Sovelluksen suoritus on suurimman osan ajasta sovelluksen sisimmässä silmukassa. Silmukassa sovellus odottaa käyttöjärjestelmän lähettämiä tapahtumia. Kun sovellus vastaanottaa tapahtuman käyttöjärjestelmältä, muuttaa sovellus käyttöjärjestelmältä vastaanotetun tapahtuman käyttöliittymäkirjaston ymmärtämään muotoon.

Tämän jälkeen tapahtuma lähetetään käyttöliittymän sisäiselle esitystavalla. Käytännössä tämä tarkoittaa puurakenteen läpikäymistä sopivan algoritmin määräämällä tavalla ja kutsumalla jokaiselle puun solmulle operaatiota, joka vastaanottaa parametrinä tapahtuman. Komponentti käsittelee tapahtuman, jos se on komponentin kannalta mielekäs. Hiiren generoimat tapahtumat ovat komponentin kannalta mielekkäitä, jos hiiren kohdistin sijaitsee komponentin rajojen sisäpuolella.

Yleensä tapahtumien käsittelijät ovat vastuussa käyttöliittymän sisäisen eheyden ylläpitämisestä. Esimerkki eheyden ylläpitämisestä on Microsoft Word -sovelluksen työkalurivit. Komponentti korostetaan tavallisesta poikkeavalla ulkoasulla, jos hiiren kohdistin on sen alueella. Käyttäjä voi ulkoasusta päätellä komponentin, jota seuraava napin painallus käsittelee.

Ennalta määrättyjen ehtojen täyttyessä tapahtuman käsittely laukaisee kytkennän määrittelemän toimenpiteen. Esimerkiksi napin tapauksessa, jos hiiren kohdistin on napin alueella, ja käyttäjä painaa ja vapauttaa hiiren oikean napin, suoritetaan kytkennän määrittelemä toimenpide. Ennalta määrätyt ehdot ovat täysin komponenttikohtaisia. Komponentilla saattaa olla useita ehtoja, jotka laukaisevat eri toimenpiteet.

Kuvassa 6.1 on havainnollistettu tilannetta, jossa kytkentä siirtää ohjelman suorituksen pinokoneelle alkaen aliohjelmasta, jonka kytkentä määrää. Aliohjelma voi suorittaa tietojen käsittelyä, ohjata ohjelman suoritusta tai kutsua käyttöliittymäkirjaston operaatiota. Tärkeimpänä toimenpiteenään aliohjelma voi kutsua sovelluskehyksen operaatiota.

Lopulta pinokoneen aliohjelma suorittaa paluukäskyn aliohjelmasta, ja sovelluksen suoritus palaa kytkennän laukaisseeseen tapahtuman käsittelijään. Tämän jälkeen suoritus palaa tapahtumien välittämisen suorittavaan aliohjelmaan. Viimeiseksi sovelluksen suoritus palaa sovelluksen sisimpään silmukkaan, jossa sovellus odottaa seuraavaa käyttöjärjestelmän lähettämää tapahtumaa.

Kuvassa 6.1 esitetty sovelluksen rakenne saattaa näyttää puutteelliselta sovelluksen suorituksen kannalta, jos tarkastellaan algoritmit -osassa esiteltyjä ohjelmistokomponentteja. Tämä on seurausta siitä, että käyttäjä voi määritellä käyttöliittymän merkkauskielen avulla ja komentojonot ohjelmointikielen avulla. Käyttöliittymän sisäinen esitystapa on keskeisessä asemassa ohjelman suorituksen kannalta.

7 Toteutusvaihtoehtoja

Tämän pro gradu -tutkielman sisältöä havainnollistamaan on laadittu C/C++ -ohjelmointikielellä ohjelma, joka sisältää edellisessä luvussa kuvatut ohjelmistokomponentit. Tässä luvussa esittelen vaihtoehtoisia lähestymistapoja luvussa 6 kuvatun toiminnallisen ytimen toteuttamiseksi. Vaihtoehtoisten lähestymistapojen tarkoitus on vähentää toiminnallisen ytimen toteuttamiseen vaadittua työmäärää. Aluksi tarkastelen ohjelmointikielen kääntäjän toteuttamista helpottavia työkaluja. Lopuksi käsittelen käyttöliittymäkirjaston toteuttamista helpottavia kirjastoja.

7.1 Python

Python-ohjelmointikieli soveltuu *laajennoskieleksi* (*extension language*), jonka avulla muokataan sovelluksen toimintaa. Tämän vuoksi Python -kieli soveltuu esimerkiksi käyttöliittymän toiminnallisuuden määrittelyyn. Käytännössä Python -kielen *tulkki* (*interpreter*) on liitettävissä osaksi sovellusta, joka on kirjoitettu C -kielellä. Tulkki vastaa kuvassa 7.1 esiintynyttä pinokonetta.

Python -ohjelmointikielen kehitystyö on kestänyt yli kymmenen vuotta. Kyseessä on pitkälle kehitetty kokonaisuus, johon sisältyy muun muassa kääntäjä, tulkki ja visuaalinen kehitysympäristö. Kokonaisuus sisältää myös dokumentaation, joka koostuu kieliopin ja kirjastojen kuvauksista. Kirjastot mahdollistavat käyttöjärjestelmäkutsujen, Internet-protokollien ja merkkauskielien hyödyntämisen. Kokonaisuuteen on lisättävissä uusia kirjastoja. Eräs tällainen laajennos on Python OpenGL, joka mahdollistaa kolmeulotteisen tietokonegrafiikan piirtämisen.

7.2 ScanGen, Lex, LLGen ja LALRGen

Ohjelmointikielen kääntäjän toteuttamista ovat helpottamassa työkalut, joiden avulla generoidaan kääntäjän rungon muodostava lähdekoodi. Työkalut saavat syötteenään tiedoston, joka määrittelee käännettävän kielen kieliopin. Työkalut jakautuvat kahteen ryhmään. Ensimmäisen ryhmän työkalut generoivat lähdekoodin, joka tunnistaa käännettävän ohjelmointikielen lähdekoodissa esiintyvät alkiot. Tällaista lähdekoodia kutsutaan *lukijaksi* (*scanner*). Toisen ryhmän työkalut generoivat lähdekoodin, joka tunnistaa alkioiden muodostamia jonoja. Tällaista lähdekoodia kutsutaan *jäsentäjäksi* (*parser*). Työkalujen tavoite on rajoittaa työmäärää.

Ensimmäisessä vaiheessa käännettävä lähdekoodi jaetaan alkioihin. Tätä vaihetta kutsutaan *leksikaaliseksi analyysiksi*. Tunnistettavan kielen alkiot voidaan määritellä *säännöllisillä lauseilla* (*regular expressions*). Lukija on toteutettavissa *deterministinen äärellisen automaatin* (*deterministic finite automaton*) avulla. Toteutus koostuu kahdesta osasta: *siirtymätaulukko* (*transition table*) ja *ajuriohjelmasta* (*scanner driver*). Siirtymätaulukko on johdettavista säännöllisillä lauseilla määritellyistä alkioista. Ajuriohjelma lukee käännettävää lähdekoodia merkki kerrallaan, kunnes kieliopin alkio on tunnistettu. Lukijan lähdekoodin generoivia työkaluja ovat esimerkiksi ScanGen ja LEX.

Toisessa vaiheessa lukijan tunnistamista alkioista muodostetaan *jäsennyspuu* (*parse tree*). Käännettävän ohjelmointikielen kielioppi määrää lailliset alkiojonot. Aina kun jäsentäjä kohtaa laillisen kieliopin lauseen, lauseen jäsennyspuu lisätään osaksi käännettävän ohjelman jäsennyspuuta. Jäsennyspuu kuvaa ainakin kieliopillisesti laillisen ohjelman. Tätä vaihetta kutsutaan *kieliopin analyysiksi* (*grammar analysis*). Tätä vaihetta helpottavia työkaluja ovat esimerkiksi LLGen ja LALRGen. Edellä mainitut työkalut generoivat taulukoita, joita kutsutaan *jäsennystauluiksi* (*parse table*). Jäsennystaulut ovat tarkoitettu käytettäväksi *jäsennysohjelmassa* (*parser driver*).

Kolmanneksi työkalujen onnistunut hyödyntäminen edellyttää merkityksen kiinnittämistä käännettävälle kieliopille. Käytännössä tämä tarkoittaa semanttisten aliohjelmien kirjoittamista, joita kutsutaan jäsennysohjelmasta. Semanttiset aliohjelmat joko taltioivat yksittäisen alkion arvon myöhempää käyttöä varten tai suorittavat toimenpiteen usealle taltioidulle arvolle. Useaan arvoon kohdistuvat toimenpiteet voivat lisätä yhden solmun käännetyn ohjelman *väliaikaiseen esitysmuotoon* (*intermediate representation*). Väliaikainen esitysmuoto on tietorakenne, joka kuvaa sekä kieliopiltaan että merkitykseltään laillisen ohjelman. Tätä vaihetta kutsutaan *semanttiseksi analyysiksi*.

7.3 Win32-rajapinta ja OpenGL-kirjasto

Windows-käyttöjärjestelmässä on toteutettuna ikkunointijärjestelmä, jossa on toteutettuna useat luvussa 3 esitellyt käyttöliittymäkomponentit. Windows-käyttöjärjestelmässä ikkunointijärjestelmän toiminnallisuuteen päästään käsiksi Win32-rajapinnan avulla [12]. Tämän vuoksi toteutettaessa muokattavaa käyttöliittymää ei tarvitse toteuttaa käyttöliittymäkirjastoa itse.

Luvussa 3 esitelty sisäinen esitystapa edellyttää luokkakirjaston toteuttamista Win32-rajapinnan päälle ennen kuin käyttöliittymän muokattavuus on saavutettavissa. Luokkakirjasto on ennen kaik-

kea joukko luokkia, joista on mahdollista muodostaa puurakenne. Tämän vuoksi käyttöliittymä on muokattavissa merkkauskielen avulla. Tässä lähestymistavassa tarvitaan luokka jokaiselle merkkauskielessä esiintyvälle merkitsimelle.

Yksittäinen kirjaston luokka kätkee taakseen joukon kutsuja Win32-rajapintaan. Luokkakirjasto voi sisältää luokan, joka kapsuloi valintaruudun toiminnallisuuden. Luokka sisältää kutsuja Win32-rajapinnan palveluihin, joilla hallitaan käyttöjärjestelmän toteuttamaa valintaruutua. Tällainen luokka muuttaa Win32-rajapinnan ohjelmoinnin olio-ohjelmoinniksi.

Luvussa 6 esittelen miten sovelluksen sisin silmukka käsittelee käyttäjän antamia syötteitä. Tässäkin tapauksessa tapahtumat lähetetään puurakenteelle. Käyttöliittymäkomponenttia esittävä luokka, joka käsittelee tapahtuman, delegoi kutsuja Win32-rajapinnalle. Tällöin käyttöjärjestelmä on vastuussa komponenttien sisäisistä tiloista ja esittämisestä kuvaruudulla. Kaikki luvussa 2 esiintyneet Microsoftin sovellukset hyödyntävät Win32-rajapintaa.

Vaihtoehtoisesti on mahdollista hyödyntää OpenGL-kirjastoa [13]. Sen avulla piirretään grafiikkaa kuvaruudulle. Kirjasto on käyttökelpoinen, koska se on käytettävissä lähes tulkoon jokaisessa käyttöjärjestelmässä. Tässä lähestymistavassa ei toteuteta ainoastaan käyttöliittymän sisäistä esitystapaa, vaan kokonainen käyttöliittymäkirjasto. Jokainen käyttöliittymäkirjaston luokka on vastuussa käyttöliittymäkomponentin sisäisestä tilasta ja esittämisestä kuvaruudulla.

8 Yhteenveto

Tässä luvussa kertaan tämän pro gradu -tutkielman keskeinen asiasisältö. Lisäksi esittelen työhön liittyvää kritiikkiä ja käsittelemättä jääneitä jatkotutkimukseksi kelpaavia aiheita.

Merkkauskielellä käyttäjä voi määritellä käyttöliittymän rakenteen ja ulkoasun. Ohjelmointikielellä käyttäjä voi määritellä käyttöliittymän toiminnallisuuden. Sekä merkkauskielinen että ohjelmointikielinen määrittely sijaitsevat ulkoisissa tiedostoissa. Tämä lähestymistapa mahdollistaa sen, että käyttäjä voi vapaasti muokata sovelluksen käyttöliittymää.

Luvussa 4 esiteltiin merkkauskieli, jolla sovelluksen käyttöliittymä määritellään. Merkkauskielen tulee käsittää suurempi joukko käyttöliittymissä esiintyviä komponentteja kuin olen pro gradu -tutkielmassani esittänyt, jotta merkkauskieli palvelisi käyttötarkoitustaan. Tämä tarkoittaa merkitsimien ja lisäpiirteiden lisäämistä merkkauskieleen kattamaan suuremman joukon rakenteita ja ulkoasuja.

Merkkauskielessä tärkein ominaisuus on kuitenkin komponenttien kytkeminen toiminnalliseen ytimeen. Esittelen kytkennät aliohjelmakutsujen tasolla ja tiedonvälityksen perustietotyyppien tasolla. Merkkauskieli voisi tukea ainoastaan suoran kytkennät komponentista toiminnallisen ytimen operaation ilman, että on mahdollista määritellä komentojonoja. Tiedon välitys voisi tapahtua abstraktien tietotyyppien tasolla.

Luvussa 5 esittelin ohjelmointikielen, jolla sovelluksen käyttöliittymän toiminnallisuus määritellään. Toiminnallisuus sijoitetaan komentojonoihin. Komentojonoista on mahdollista kutsua esimerkiksi käyttöliittymäkirjaston palveluita. Näillä palveluilla muutetaan sovelluksen suorituksen aikana merkkauskielessä esiintyneitä määrityksiä. Käyttöliittymäkirjaston palveluiden tulisi mahdollistaa samojen kohteiden muokkaamisen kuin merkkauskielenkin.

Ohjelmointikieleksi on valittu Pascal, joka on yleisesti käytetty opetuskieli. Tämän vuoksi merkkauskielen lisääminen voidaan mieltää laajennokseksi, jonka avulla Pascal -ohjelmointi on siirretty graafiseen ympäristöön. Ajatusta graafisesta Pascal -ohjelmointikielen oppimisympäristöstä voisi kehittää pidemmällekin.

Luvussa 6 esittelin toiminnallisen ytimen rakenteen, jota sovelluksen tulisi noudattaa. Sovelluskehitin voisi oletusarvoisesti generoida ohjelmakoodia, joka noudattaa pro gradu -tutkielmassani esittämääni toiminnallisen ytimen rakennetta. Ohjelmistosuunnittelijoiden vastuulla on laatia sovelluskehyksen rajapinta ja sovelluskehyksen toteutus. Sovelluskehyksen rajapinnassa esiintyvät operaatioiden määrittelyt ovat viitattavissa merkkauskielestä ja ohjelmointikielestä käsin. Sovelluksen käyttöliittymä kehitetään erillisellä työkalulla, joka voi muistuttaa nykyisiä visuaalisia sovelluskehittimiä. Sovelluskehitin generoi käyttöliittymästä merkkauskieliset ja ohjelmointikieliset määrittelyt, jotka ovat vapaasti kehitettävissä sovelluskehyksestä riippumatta.

Viiteluettelo

[1] Abrams, M. ja Helms, J. *UIML 3.0 Language Specification*, Internet WWW-sivu, URL: http://www.uiml.org/docs/uiml30

[2] Barron, D. *Pascal The Language and its Implementation*, John Wiley & Sons, New York, 1981

[3] Blom, J. ja Monk, F. *Theory of Personalization of Appearance: Why Users Personalize Their PCs and Mobile Phones*, Human-Computer Interaction, 193-228, 2003

[4] Dewan, P. ja Choudhary, R. *Coupling the User Interface of a Multiuser Program*, ACM Transactions on Computer-Human Interaction, 1-39, 1995

[5] Fischer,N. ja LeBlanc, R. *Crafting A Compiler With C*, Benjamin Cummings, Redwood City, California, 1991

[6] Gamma, E., Helm,R., Johnson, R. ja Vlissides, J. *Design Patterns: Elements Of Reusable Object-Oriented Software, Addison-Wesley*, Boston, 1995

[7] Hill, R., Brinck, T., Rohall, S., Patterson, J. ja Wilner, W. *The Rendezvous Architecture and Language for Constructing Multiuser Applications*, ACM Transactions on Computer-Human Interaction, 81-125, 1994

[8] Horowitz, E ja Sahni, S. *Fundamentals of Data Structures In Pascal*, Computer Science Press, New York, 1994

[9] Schiaffino, S. ja Amandi, A. *User-interface agent interaction: personalization issues*, International Journal of Human-Computer Studies, 129-148, 2004

[10] Taylor, R., Nies,K., Bolcer, G., MacFarlane, C., Anderson, K. ja Johnson, G. *Chiron 1: A Software Architecture for User Interface Development, Maintenance and Run-Time Support*, ACM Transactions on Computer-Human Interaction, 105-144, 1995

[11] Peltomäki, J.& al. *CGI- ja ASP-ohjelmointi*, Teknolit, Porvoo, 2000

[12] Petzold, C. Programming Windows Fifth Edition, Microsoft Press, Redmond, Washington, 1999

[13] Woo, M., Neider, J., Davis, T. ja Schreiner D. *OpenGL Programming Guide Third Edition*, Addison-Wesley, sReading, Massachusetts, 1999