Roope Raisamo (toim.)

Käyttöliittymien ohjelmistoarkkitehtuurit

TIETOJENKÄSITTELYTIETEIDEN LAITOS
TAMPEREEN YLIOPISTO

B-2006-1

TAMPERE 2006

Sisällys

Seminaari ”Käyttöliittymien ohjelmistoarkkitehtuurit”

Roope Raisamo

Johdanto

Perinteisesti käyttöliittymän toteutusta on pidetty ohjelmistokehitysprosessissa vain pienenä yksittäisenä osatehtävänä. Uudet monipuoliset käyttöliittymäratkaisut, erityisesti niiden vuorovaikutteisuus- ja tehokkuusvaatimukset, vaativat kuitenkin merkittävää panostusta myös käyttöliittymien toteutusratkaisujen suunnitteluun. Joissakin tapauksissa käyttöliittymän toteutusarkkitehtuuri määrää jopa koko sovelluksen arkkitehtuurin, eikä päinvastoin, kuten perinteisissä ohjelmistoissa on ollut tapana.

Yksinkertaisimmillaan graafisen käyttöliittymän toteutusarkkitehtuuri on peräisin suoraan valitusta käyttöjärjestelmästä (esim. linux, Microsoft Windows) tai suoritusympäristöstä (esim. Java). Tämä on kuitenkin vain perustaso graafisenkin käyttöliittymän toteutuksessa, ja sovelluksesta riippuen erilaiset korkeamman tason arkkitehtuurit ovat usein tarpeen. Uusissa moniaistisissa käyttöliittymissä arkkitehtuurien vaatimukset ovat selvästi suuremmat, koska eri syötteiden ja palautteiden reaaliaikaisuus ja synkronointi ovat niissä keskeisiä.

Tässä Tampereen yliopiston Tietojenkäsittelytieteiden laitoksella syksyllä 2005 järjestetyssä seminaarissa keskityttiin eritasoisiin käyttöliittymien toteutusarkkitehtuureihin. Tavoitteena oli saada osallistujille kokonaisnäkemys käyttöliittymäarkkitehtuurien nykytilasta ja niiden tulevaisuuden kehitystarpeista.

Toivon, että julkaisu nyt valmistuttuaan toimii lähtökohtana käyttöliittymien ohjelmistoarkkitehtuurien opetukselle ja tutkimukselle Suomessa. Monien julkaisuun sisältyvien seminaaritöiden pohjalta on parhaillaan tekeillä pro gradu -tutkielmia, joista yksi on jo valmistunutkin. Lisäksi seminaarissa mukana olleet kolme jatko-opiskelijaa jatkavat aihepiiriin liittyvä työtään omassa tutkimuksessaan.

Mobiilisovellusten käyttöliittymäarkkitehtuurit

Tomi Heimonen

Tiivistelmä.

Tässä työssä käsitellään mobiilisovellusten toteutukseen käytettävissä olevia ohjelmistoalustoja, niiden tarjoamia sovelluskehyksiä, ja erityisesti ohjelmistokehysten käyttöliittymäarkkitehtuureita. Ohjelmistoalustoista ja sovelluskehyksistä esitellään niiden sovelluskehityksen kannalta keskeiset piirteet sekä käyttöliittymäarkkitehtuurin toimintaperiaate. Ohjelmistoalustojen ja sovelluskehysten vertailussa keskitytään ohjelmistokehityksen kannalta olennaisiin kysymyksiin, kuten käyttöliittymäkomponenttien asemoinnin hallintaan ja käyttöliittymän adaptiivisuuteen eri laiteympäristöissä.

Avainsanat ja -sanonnat: mobiililaitteet, ohjelmistoalusta, sovelluskehys, käyttöliittymäarkkitehtuuri
CR-luokat: H.5.2, D.2.2

1. Johdanto

Tässä artikkelissa tarkastellaan sovelluskehitystä mobiileilla päätelaitteilla, erityisesti tarjolla olevia ohjelmistoalustoja ja niiden käyttämiä sovelluskehyksiä. Tällä hetkellä markkinoilla on useita erityyppisiä mobiileja päätelaitteita, joista suurimpaan osaan on mahdollista kehittää ns. kolmannen osapuolen sovelluksia. Tämä on pääosin mahdollista, koska laite- ja ohjelmistovalmistajat yleisesti tarjoavat kehittäjille ohjelmistokehityksessä tarvittavat ohjelmistoalustat, sovelluskehykset sekä ohjelmointirajapinnat.

Tämä työ pyrkii kokoamaan yhteen keskeisimmät ohjelmistoalustat ja sovelluskehykset. Pääpaino tarkastelussa on alustojen ja sovelluskehysten käyttöliittymäarkkitehtuurien esittelyssä. Lopuksi vertaillaan eri sovelluskehyksissä ja arkkitehtuureissa esitettyjä ratkaisumalleja ja niiden vahvuuksia ja heikkouksia mobiililaitteiden ohjelmistokehityksessä.

2. Mobiililaitteiden ohjelmistokehitys

Mobiililaitteiden ohjelmistokehitys on haastava ohjelmistotekniikan osa-alue. Viime vuosiin asti mobiililaitteet ovat olleet suljettuja ympäristöjä, joten sovelluskehitys on pääosin ollut laitevalmistajien vastuulla. Nykyisin sovelluskehitys on kuitenkin mahdollista lähestulkoon kaikille, kiitos tarjolla olevien avoimien rajapintojen.

Mobiilisovellusten kehitys on kuitenkin itsessään erittäin haasteellista, johtuen mobiiliympäristön asettamista rajoitteista [Abrahamsson *et al.*, 2004], kuten:

- laitteiden asettamat tekniset rajoitteet ja päätelaitteiden uusiutumistahti
- erilaiset standardit, tiedonsiirtoprotokollat ja verkkoteknologiat
- eri ohjelmistoalustojen huomioiminen
- käyttäjien erikoistuneet tarpeet
- tuotekehitykselle asetetut tiukat tavoiteaikataulut (ns. *time to market requirements*)

Erilaisten mobiililaitteiden erot, esimerkiksi teknisten ja fyysisten ominaisuuksien suhteen, asettavat vaatimuksia käytettäville sovelluskehitysmenetelmille. Tämän lisäksi vaikuttavana tekijänä voidaan nähdä toteutettavan sovelluksen tyyppi. Leghari [2003] jakaa mobiililaitteille kehitettävät sovellukset voidaan jakaa pääsääntöisesti kahteen eri kategoriaan: (1) web-asiakasohjelmassa ajettava web-sovellukset ja (2) mobiililaitteessa ajettavat natiivisovellukset. Näiden lisäksi on erilaisia SMS (*Short Message Service*) ja MMS (*Multimedia Messaging System*) -teknologioihin perustuvia sovelluksia, joita käytetään yleisesti esimerkiksi soittoäänien, taustakuvien ja mobiilipelien lataamiseen päätelaitteeseen. Niitä ei käsitellä tässä arvioinnissa, johtuen niiden rajoittuneista vuorovaikutustarpeista ja -mahdollisuuksista käyttöliittymän suhteen.

2.1. Mobiilit web-sovellukset

Mobiilit web-sovellukset muistuttavat arkkitehtuuriltaan normaaleja, työpöytäympäristöön kehitettyjä web-sovelluksia. Mobiililaitteessa ajettava selainasiakasohjelma ottaa yhteyden palvelinkoneessa ajettavaan palvelinohjelmaan ja pyytää siltä resurssin, esimerkiksi HTML-dokumentin, jonka se sitten näyttää käyttäjälle oman käyttöliittymänsä välityksellä. Tyypillisesti selainasiakasohjelma hoitaa itse resurssin muokkaamisen

mobiililaitteella esitettävään muotoon, mutta viime aikoina ovat yleistyneet ratkaisut, joissa esitysmuodon prosessointi tapahtuu palvelimen toimesta [Opera Mini, 2005].

Kaksi tärkeintä mobiilien web-sovellusten toteuttamiseen käytettävää sovellusympäristöä ovat WML-kuvauskieleen (*Wireless Markup Language*) [WML, 2001] perustuva WAP-arkkitehtuuri (*Wireless Application Protocol*) [WAP, 2001] ja cHTML-kuvauskieleen (*compact HTML*) [Kamada, 1998] perustuva, lähinnä Aasiassa ja joissain Keski-Euroopan maissa käytössä oleva i-mode -järjestelmä [i-mode, 2005]. Nämä toistensa kanssa kilpailevat sovellusarkkitehtuurit mahdollistivat ensimmäisinä rikkaampien, vuorovaikutteisia lisäarvopalvelujen toteuttamisen etenkin matkapuhelinympäristöön.

Näiden erikoistuneiden ympäristöjen lisäksi mobiileja web-sovelluksia voidaan kehittää myös käyttämällä World Wide Web - konsortion kehittämää XHTML-kuvauskieltä (*eXtensible Hypertext Markup Language*) [XHTML, 2002]. Nykyisin markkinoille tulevissa mobiililaitteissa on pääsääntöisesti XHTML-sivujen käyttöön soveltuva selainohjelmisto, joten oletettavasti XHTML-pohjaisten web-sovellukset tulevat mitä todennäköisimmin syrjäyttämään edellämainitut, erikoistuneisiin kuvauskieliin perustuvat sovellusympäristöt.

Mobiiliympäristöön onkin kehitetty oma varianttinsa nimeltä XHTML Mobile Profile [XHTML MP, 2001], joka sisältää XHTML:n keskeisimmät osat. XHTML MP:n tarkoituksena onkin pyrkiä vähentämään web-sovellusten toteuttamiskustannuksia mahdollistamalla saman kuvauskielen käytön sekä mobiileilla web-sovelluksissa että työpöytäkäyttöön tarkoitetuissa web-sovelluksissa.

2.2. Mobiililaitteessa ajettavat sovellukset

Mobiilisovellukset eroavat mobiileista web-sovelluksista siinä, että ne asennetaan ja ajetaan omina prosesseinaan mobiililaitteessa ja niillä on omat, erikoistuneet käyttöliittymänsä. Mobiilisovellukset toteutetaan tyypillisesti tietylle ohjelmistoalustalle. Eri sovellusalustojen tukeminen vaatii tällöin eri versioiden kehittämistä kullekin tuettavalle alustalle.

Mobiilisovelluksilla on tyypillisesti käytössään mobililaitteen muisti-, tiedostojärjestelmä- ja verkkoyhteysresurssit, toisin kuin mobiileilla web-sovelluksilla [Leghari, 2003]. Se, miten laajasti näitä resursseja voidaan käyttää osana sovelluksen toiminnallisuutta riippuu ohjelmistoalustan luonteesta, so.

kuinka laaja pääsy kyseisiin resursseihin sallitaan. Näihin yksityiskohtiin palataan tarkemmin eri ohjelmistoalustoja tarkasteltaessa.

Mobiilisovellusten toteuttamiseen on olemassa useita eri ohjelmistoalustoja ja ohjelmistokehyksiä. Tässä työssä keskitytään tarkastelemaan mobiilisovelluksia, koska niiden toteuttamiseen käytetyt ympäristöt tarjoavat monipuoliset vaihtoehdot käyttöliittymän toteutukselle. Ensin kuvaillaan keskeisimmät käytössä olevat mobiilit ohjelmistoalustat. Tarkastelussa keskitytään pääasiallisesti matkapuhelinkeskeisiin ohjelmistoalustoihin ja niiden arkkitehtuurien käyttöliittymäratkaisuihin. Työn keskeisimmän osan muodostaa kyseisten ohjelmistoalustojen sovelluskehysten esittely. Sovelluskehysten tarkastelussa pääpaino kiinnitetään erityisesti sovelluskehyksen käyttöliittymäarkkitehtuuriin. Lopuksi vertaillaan työssä esitettyjä ohjelmistoalustoja ja sovelluskehyksiä sekä esitetään yhteenveto tarkastelun tuloksista.

3. Mobiilisovellusten ohjelmistoalustat

Mobiilisovellusten keskeisimmistä ohjelmistoalustoista tähän tarkasteluun on valittu Symbian [Symbian, 2005a], Java 2 Micro Edition [J2ME, 2005] sekä BREW [BREW, 2005]. Kyseiset ohjelmistoalustat ovat käytössä pääsääntöisesti matkapuhelinlaitteissa, joskin matkapuhelimen ja PDA-laitteiden raja on hämärtynyt ns. älypuhelinten kehityksen myötä.

3.1. Symbian

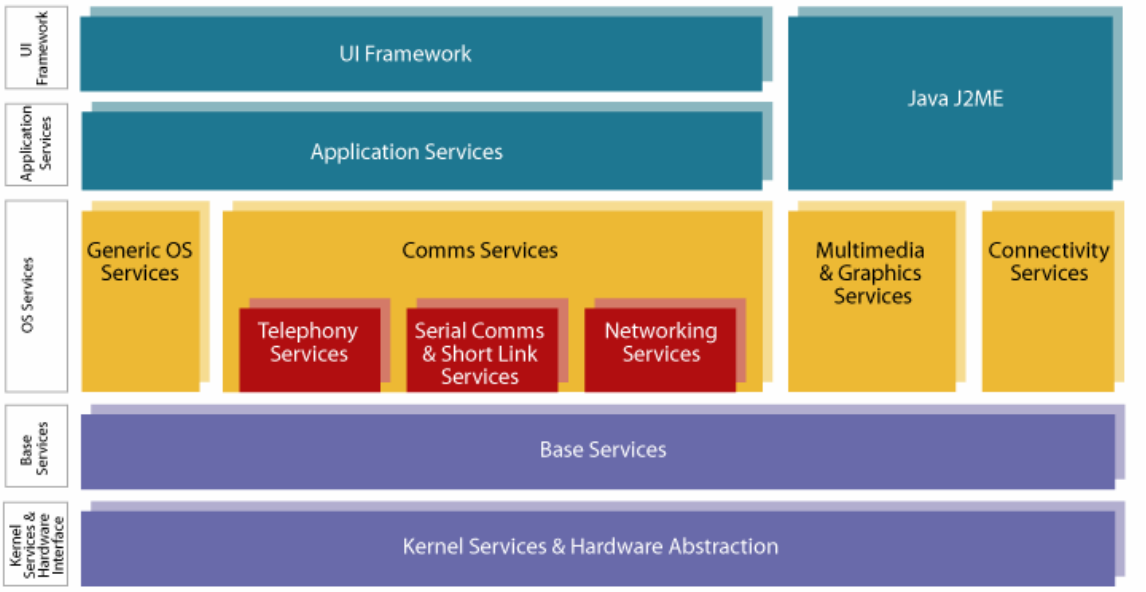
Symbian on tällä hetkellä johtava ns. älypuhelimien ohjelmistoalusta ja käyttöjärjestelmä, jota käyttävät tuotteissaan muun muassa Nokia, NTT DoCoMo ja SonyEricsson. Symbian-käyttöjärjestelmä tarjoaa sovelluskehittäjille monipuolisen ARM-laitealustalla toimivan kehitysympäristön, joka tukee säikeistystä ja samanaikaista suoritusta.

Käyttöjärjestelmän perusrakenne noudattelee kerrosarkkitehtuuria [Symbian, 2005b], jossa on viisi kerrosta:

- Ydin- ja laitteistointegraatiokerros (*kernel and hardware integration*) huolehtii käyttöjärjestelmän ydintoiminnoista ja tarjoaa laitteistoajurit.
- Perustoimintokerros (*base services*) sisältää matalan tason sovelluskehityskehyksen sekä tiedostojärjestelmärajapinnat (esim. muistikortteihin).

- Käyttöjärjestelmäpalvelukerros (*operating system services*) sisältää käyttöjärjestelmän keskeisimmät toiminnot, kuten multimedia- ja grafiikkajärjestelmät, verkkoyhteydet yms.
- Sovelluspalvelukerros (*application services*) tarjoaa sovelluksille keskeiset geneeriset sovellusmoottorit (*application engines*) yhteystietojen, puhelinnumeroiden, viestien ja vastaavien hallintaan.
- Käyttöliittymäkehys sisältää tuen erilaisten käyttöliittymäratkaisujen toteuttamiseen.

Symbian-käyttöjärjestelmän toiminnallinen kuvaus on esitetty kuvassa 1.



Kuva 1. Symbian-käyttöjärjestelmän version 9 toiminnallinen kuvaus [Symbian, 2005b].

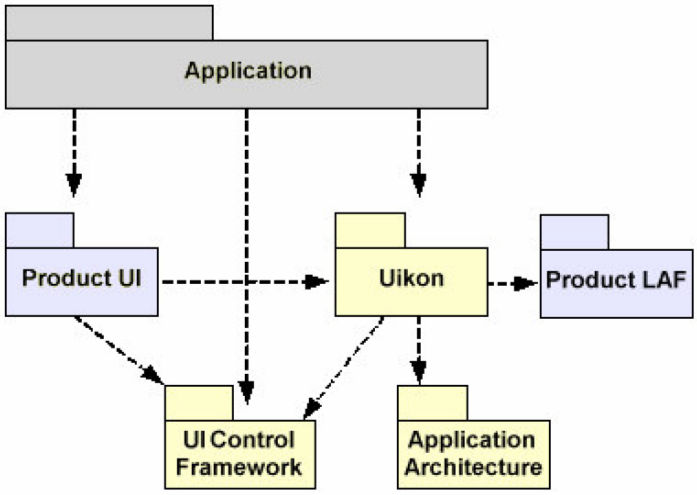
Näiden kerrosten lisäksi käyttöjärjestelmä sisältää Java-sovellusten (J2ME) ajamisen mahdollistavan virtuaalikoneajoympäristön. Käyttöliittymien sovelluskehityksen kannalta Symbian-käyttöjärjestelmän keskeisimmät kerrokset ovat käyttöliittymäkehys sekä Java-tuen tarjoava ajoympäristö. Java-pohjaisia J2ME-mobiilisovelluksia käsitellään tarkemmin sovelluskehityskehyksiä käsittelevässä luvussa.

Käyttöliittymäkehys (*Uikon*) koostuu kahdesta osasta: itse sovelluskehyksestä ja työkaluista (*toolkit*). Käyttöliittymäsovelluskehyksen keskeisimmät ominaisuudet ovat:

- tapahtumavetoinen käyttöliittymäarkkitehtuuri

- ikkunointijärjestelmä
- sovellusten välinen tehtävävetoinen linkitys
- käyttöliittymän ulkoasun muokkaus (*look and feel*)
- rajapinnat puheentunnistus- ja käsialantunnistuskomponenteille
- tuki useille samanaikaisille puoliläpinäkyville ikkunoille.

Symbian-käyttöliittymäarkkitehtuurin perusrakenne [de Jode ja Turfus, 2005] on esitetty kuvassa 2.



Kuva 2. Symbian-käyttöliittymäarkkitehtuurin rakenne

Käyttöliittymäsovelluskehyksen suunnittelussa on ollut tavoitteena tarjota mahdollisimman kevyt kehysrakenne, jonka päälle laitevalmistajat voivat toteuttaa omat käyttöliittymäratkaisunsa (*Product UI*).

Käyttöliittymätyökalut tarjoavat sovelluskehittäjille joukon sovelluksia, jotka nopeuttavat esim. ulkoasun ja grafiikan toteuttamista. Symbian OS -alustalle on kehitetty useita referenssikäyttöliittymätoteutuksia, kuten Nokian kehittämä S60 [S60, 2005], SonyEricssonin tuotteissaan käyttämä UIQ [Mýren, 2005].

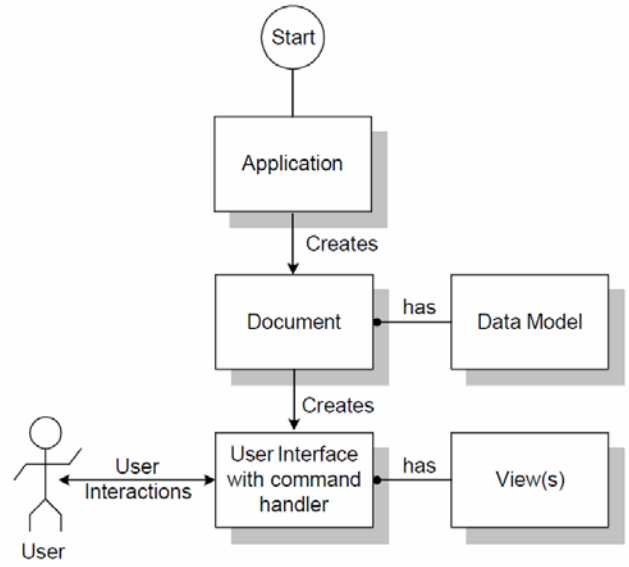
3.1.1. Symbian-sovellukset

Symbian-sovellukset ovat olioperustaisia ja niiden rakenne perustuu MVC-suunnittelumalliin [Burbeck, 1992; Näsi, 2006]. Jokaisessa sovellus sisältää omat luokat sovellukselle (*application*), dokumentille (*document*) ja sen sisältämälle tietomallille sekä käyttöliittymälle ja sen sisältämille näkymille (*views*). Käyttöliittymäkomponentit määritellään erillisissä resurssitiedostoissa. Symbian-sovellukset voivat käyttää myös sovelluksen ulkoisia tietolähteitä kuten laitteen tiedostojärjestelmää (so. tietokantaa) tiedon tallennukseen.

Sovelluksen ajonaikainen luominen tapahtuu vaiheittaisesti:

- sovellusluokka luo dokumentin
- dokumenttiluokka luo käyttöliittymän
- käyttöliittymäluokka luo näkymät

Käyttöliittymän ja sovelluksen välinen kommunikaatio tapahtuu tapahtumankäsittelijämekanismin avulla, jotka määritellään käyttöliittymäluokassa. Symbian-sovelluksen rakenne [Ortiz, 2002] on esitetty kuvassa 3.



Kuva 3. Symbian-sovelluksen rakenne

Symbian-sovellusten käyttöliittymäarkkitehtuurin rakennetta UIQ-sovelluskehyksessä tarkastellaan lähemmin luvussa 4.1.

3.2. Java 2, Micro Edition

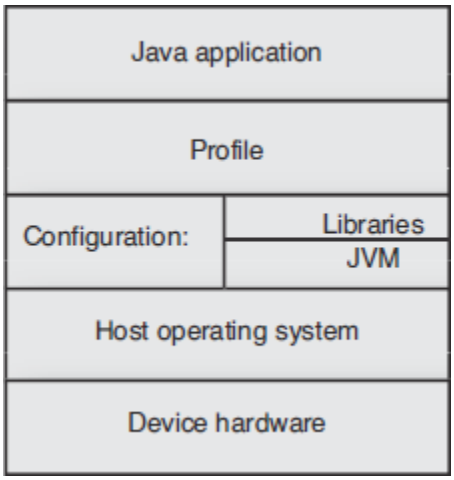
J2ME on Java-ohjelmointirajapinnoista koostuva ohjelmistoalusta, joka on tarkoitettu Java-sovellusten toteutukseen sulautetuissa laitteissa. Lähtökohtana J2ME:n suunnittelussa on ollut toimiminen rajoitetutussa laiteympäristössä, joka heijastuu alustan rajoitteisiin: vain osaa Java-virtuaalikoneen ja Java-ohjelmointirajapintojen ominaisuuksista tuetaan.

Java-luokkakirjastojen ja virtuaalikoneen lisäksi J2ME-arkkitehtuuri koostuu profiileista (*profile*) ja konfiguraatioista (*configuration*). Konfiguraatiot määrittelevät toteutusympäristöön kuuluvat luokkakirjastot ja järjestelmätason

ohjelmointirajapinnat sekä virtuaalikoneen ominaisuudet. Profiili puolestaan sisältää sovellustason ohjelmointirajapinnat, mm.

- sovelluksen elinkaaren hallintaan
- käyttöliittymän toteutukseen
- ajastinten käyttöön
- verkkoyhteyksien käyttöön
- tiedostojärjestelmään

J2ME-arkkitehtuurin rakenne [Piroumian, 2002] on esitetty kuvassa 4.



Kuva 4. J2ME-arkkitehtuurin rakenne.

Mobiililaitteiden kannalta tärkein profiili on Mobile Information Device Profile [MIDP, 2002], joka toimii yhdessä Connected Device Limited Configuration [CLDC, 2003] -konfiguraation kanssa. Yhdessä ne tarjoavat sovelluskehittäjälle Java-sovellusten ajoympäristön erityistesti mobiileille päätelaitteille, kuten matkapuhelimille ja PDA-laiteille. MIDP-profiilin lisäksi CLDC-konfiguraatio sisältää erillisiä ohjelmointirajapintojen määrityksiä esimerkiksi 3D-grafiikan, langattoman viestinnän ja multimedian käyttämiseksi J2ME-sovelluksissa. MIDP-profiilia ja sen käyttöliittymätoteutusta tarkastellaan lähemmin luvussa 4.2.

3.3. BREW

BREW (*Binary Runtime Environment for Wireless*) on mobiililaitteiden sovelluskehitykseen tarkoitettu avoimen lähdekoodin ohjelmistoalusta, joka kattaa myös sovellusten levityksen mobiililaitteisiin. Natiiviohjelmointikieli BREW-alustalla on C++, mutta se tukee myös muilla ohjelmointikielillä

toteutettuja sovelluksia, esimerkiksi J2ME-sovelluksia. BREW-alusta on J2ME:n tavoin saatavissa useille eri laitealustoille.

BREW-arkkitehtuuri koostuu kolmesta osasta:

- sovelluskehitysrajapinnoista ja kirjastoista koottu työkalukokoelma (*software development toolkit*)
- mobiililaitteen muistiin asennettu suoritusympäristö BREW-sovellusten ajamista varten
- mekanismi sovellusten levitykseen ilmarajapinnan (*over-the-air*; OTA) kautta

BREW-suoritusympäristön rakenne on esitetty kuvassa 5 [BREW, 2005].

Kuva 5. BREW-suoritusympäristön rakenne.

BREW tarjoaa sovelluskehittäjille sovelluskehitysrajapintojen kautta erittäin laajan pääsyn mobiililaitteen muisti-, verkko- ja tallennustoimintoihin. Tämän takia sovellukset tulee sertifioida Qualcomm, Inc. -yhtiön toimesta ja jaka laitteisiin operaattorien levityskanavien kautta. Tämä tekee sovellusten jakelusta muita ohjelmistoalustoja hankalampaa, mutta toisaalta lisää osaltaan tietoturvaa.

Käyttöliittymien kehittäminen BREW-alustalla toimiviin sovelluksiin tapahtuu käyttämällä BREW-käyttöliittymäkehystä (*BREW UI Framework*) ja sen käytön mahdollistavaa työkalukokoelmaa (*BREW UI Toolkit*) [Rischpater, 2004]. BREW UI Toolkit -työkokoelmaa tarkastellaan tarkemmin luvussa 4.3.

3.4. Muita ohjelmistoalustoja

Edellä mainittujen lisäksi on olemassa useita muita mobiililaitteiden ohjelmistoalustoja, kuten Microsoftin Windows Mobile -käyttöjärjestelmäperhe ja PalmSourcen Palm OS -käyttöjärjestelmä. Ne ovat pääosin PDA-laitteiden ohjelmistoalustoja, tosin molempia on kehitetty viime vuosina palvelemaan paremmin matkapuhelimen ja taskutietokoneen ominaisuudet yhdistävien ns. hybridilaitteiden tarpeita.

4. Käyttöliittymien sovelluskehykset

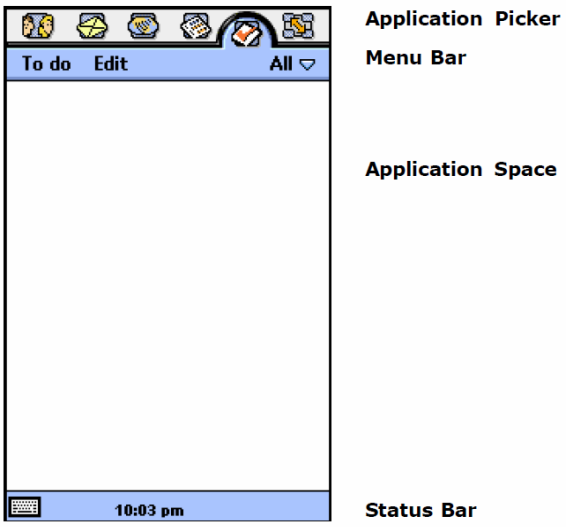
4.1. UIQ

UIQ (*User Interface for Quartz*) on UIQ-yrityksen Symbian-alustalle kehittämä käyttöjärjestelmään integroitu käyttöliittymäsovelluskehys. Kuten Symbian-alustassa, myös UIQ-kehyksessä kantavana suunnitteluperiaatteena on ollut helppo muunneltavuus eri operaattoreiden tarpeisiin. Tämän vuoksi UIQ-alustalle on mahdollista tehdä käyttöliittymämäärityksiä, joissa kuvataan mm. näytön resoluutio, vuorovaikutustyyli ja näytön orientaatio. Määrittelyjen avulla UIQ-sovellukset pystytään pääsääntöisesti mukauttamaan eri laitteille automaattisesti. [Ortiz, 2002]

UIQ rakentuu mm. seuraavista komponenteista:

- käyttöliittymäkomponenttikirjasto
- järjestelmäpalvelut
- sovelluksen rakenteen ja asettelun toteuttava sovelluskehys
- J2ME-suoritusympäristö
- Symbian-käyttöjärjestelmän palvelut

UIQ-sovellukset rakentuvat kolmesta osasta: (1) valikkorivistä, (2) sovellusalueesta ja (3) tilarivistä (kuva 6). Sovellusalue sisältää sovelluksen näkymäkomponentin sisällön (kts. luku 3.1.1.). Kuten aikaisemmin todettiin, näkymät luodaan sovelluksen käynnistyessä ja niitä koordinoi sovelluksessa erityinen käyttöliittymäluokka.



Kuva 6. UIQ-käyttöliittymän rakenne [Ortiz, 2002]

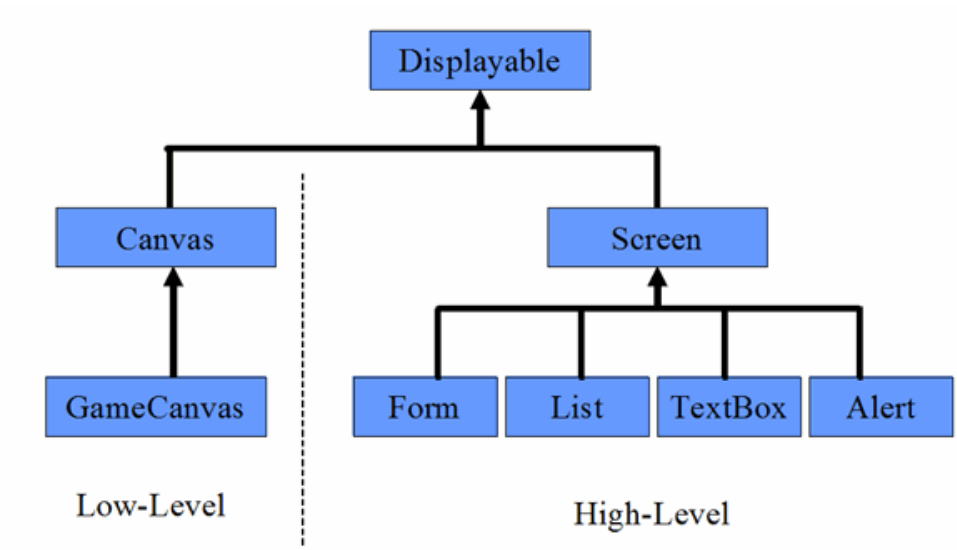
Jokaisessa UIQ-sovelluksessa on vähintään kaksi näkymää: (1) lista (*list*) ja (2) yksityiskohtainen näkymä (*detail*). Tämän vaatimuksen taustalla on pyrkimys johdonmukaisuuteen eri sovellusten välillä sekä ulkoasun samankaltaisuuden säilyttämiseen eri määrittelyn omaavilla laitteilla.

Näkymän sisältämiä käyttöliittymäkomponentteja hallitaan ns. kaavaimilla (*building block*), jotka määrittelevät sisältämänä komponenttien koon ja paikan ruudulla. Kaavaimet vastaavat perusperiaatteeltaan täten esimerkiksi Java Swing -arkkitehtuurin *layout manager* -mekanismia. UIQ-sovelluskehyksen mukana toimitetaan joukko peruskaavaimia, joilla on mahdollista toteuttaa yksinkertaisia asetteluja. Kaavaimia on myös mahdollista yhdistää haluttaessa toteuttaa monimutkaisempia asetteluja (kuva 7).

Kuva 7. Kuudesta kaavaimesta rakennettu yhdistelmänäkymä [Mýren, 2005].

4.2. MIDP

MIDP on yhdessä CLDC-määrittelyn kanssa J2ME-alustan osa, joka määrittelee sovellustason ohjelmointirajapinnat mobiililaitteen tallennusmekanismeihin, verkkoyhteysresursseihin ja muihin resursseihin, kuten multimediaominaisuuksiin. Sen keskeinen tehtävä on myös määritellä sovelluksen käyttöliittymäarkkitehtuuri. MIDP-käyttöliittymäarkkitehtuuri (*LCDUI*) koostuu kahdesta rinnakkaisesta päätasosta: korkean tason käyttöliittymäkomponenteista ja matalan tason käyttöliittymäkomponenteista (kuva 8).



Kuva 8. MIDP-käyttöliittymäarkkitehtuurin rakenne [Hemphill, 2004].

Korkean tason komponentit tarjoavat kokoelman erikoistuneita käyttöliittymäkomponentteja, kuten syötekentät, listat ja painikkeet, sovellusten käyttöliittymän toteutukseen sekä tarvittavat tapahtumankäsittelijät, joilla komponentit saadaan liitettyä osaksi sovellusta. Korkean tason komponenteilla toteutettu sovellus on periaatteessa mahdollista siirtää sellaisenaan toiseen MIDP-määrittelyä tukevaan laitteeseen. Korkean tason komponentteja käytettäessä tapahtumankäsittely toteutetaan komponentteihin kytkettyjen kuuntelijoiden (ns. *Observer*-suunnittelumalli) käyttäen.

Matalan tason komponentit tarjoavat sovelluskehittäjälle mahdollisuuden toteuttaa sovelluksen käyttöliittymän itsenäisesti, käyttämällä esimerkiksi omia käyttöliittymäkomponentteja. Tällaisia sovelluksia ovat esimerkiksi mobiilipelit, joiden ulkoasua ei voida mielekkäästi toteuttaa valmiita

käyttöliittymäkomponentteja käyttämällä. Tällöin ohjelmistokehittäjän tulee itse huolehtia käyttöliittymän ulkoasun päivityksestä (*repaint*) ja käyttäjän syötteiden käsittelystä.

Riippumatta siitä, käytetäänko matalan vai korkean tason komponentteja, jokaisen MIDP-sovelluksen tulee käyttää näyttö-luokan (*Display*) metodeja, joilla saadaan yhteys MIDP-käyttöliittymäarkkitehtuurin piirtomekanismeihin.

4.3. BREW UI Toolkit

BREW UI Toolkit on Qualcomm Inc. -yhtiön kehittämä käyttöliittymien ohjelmistokehitykseen tarkoitettu sovelluskehys, joka perustuu vahvasti MVC-suunnittelumalliin [Rischpater, 2004; Näsi, 2006].

BREW UI Toolkit -sovelluskehyksen käyttöliittymäkomponentit koostuvat kahdesta perusryhmästä: käyttöliittymäkomponenteista (*widget*) ja lomakkeista (*forms*). Käyttöliittymäkomponentit muodostavat laajenettavan käyttöliittymäohjelmistokehyksen, joka sisältää muun muassa erilaiset painikkeet ja vierityspalkit sekä muita komponentteja sisältävät kokoelmat (*containers*). Lomakkeet puolestaan tarjoavat sovelluksille ikkunointikehyksen, joka muodostuu erityyppisistä peruslomakekompontenteista, kuten listoista, ja erilaisista dialogeista. [Rischpater, 2004]

Käyttöliittymäkehitykseen on tarjolla myös muita sovelluskehyksiä, esimerkiksi Sophia Cradle, Inc. -yrityksen toteuttama SophiaFramework-luokkakirjasto [SophiaFramework, 2005], joka rakentuu BREW-alustan tarjoamien sovelluskehitysrajapintojen päälle.

4.3.1. Komponenttiarkkitehtuuri

BREW UI Toolkit -sovelluskehyksen käyttöliittymäarkkitehtuuri koostuu kolmesta osasta: mallista, käyttöliittymäkomponentista ja kokoelmasta, johon komponentti kuuluu.

Malli sisältää komponentin sisältämän tiedon ja hoitaa kytkeytymisen kuuntelijarajapintoihin. Mallilla ei kuitenkaan ole tietoa komponentin visuaalisesta ulkoasusta. Erityyppiselle tiedolle on omat mallinsa (esimerkiksi listat, taulukot, arvot), joihin kohdistuvista muutoksista malli tiedottaa komponentille.

Käyttöliittymäkomponentit piirtävät itsensä niiden mallin sisältämän mukaisesti. Komponentti itsessään ei ota kantaa siihen, mihin kohtaan näyttöä se piirretään, vaan sisältää tiedon ainoastaan sisällöstään ja koostaan. Komponentin

perustehtävä on kuunnella mallia ja piirtää itsensä pyynnöstä. Komponentti käsittelee lisäksi fokuksen vaihtoon liittyvät ja näppäintapahtumat.

Kokoelma hoitaa yksittäisten käyttöliittymäkomponenttien asemoinnin näytölle. Kokoelma on tyypillisesti itsessään erikoistunut käyttöliittymäkomponentti. Sen perustehtäviin kuuluu lisäksi piirtojärjestyksen kontrollointi. Kokoelman sisältämät komponentit piirretään näytölle alhaalta-ylös -järjestyksessä. Kokoelman erikoistapaus on ns. koristelijakomponentti (*decorator*), jonka tehtävänä on hallita tiettyjä erikoistuneita käyttöliittymäkomponentteja, kuten vierityspalkkeja.

4.3.2. Lomakepohjainen sovellusarkkitehtuuri

Sovelluskehyksen lomakearkkitehtuuri hallinnoi sovelluksen suoritusjärjestystä. Sovelluksen suoritusjärjestys on pinoperustainen - kulloinkin aktiivisena oleva lomake sijaitsee pinon päällimmäisenä, mistä se suljettaessa poistetaan.

Lomakkeet jakautuvat kahteen osaan: juurilomakkeeseen (*root form*) ja muihin lomakkeisiin. Juurilomakkeen tehtävä on hallita sovelluksen yleistä ulkoasua (sovelluksen otsikkoalue, ohjelmoitavat painikkeet (*soft keys*), taustakuva). Juurilomake olettaa, että sovelluksella on tietty ulkoasu, joka koostuu otsikkoalueesta, ohjelmoitavat painikkeet sisältävästä alueesta sekä niiden välisestä sovellusikkuna-alueesta. Tämän lisäksi juurilomake käsittelee kaikki tapahtumaviestit, joita muut lomakkeet tai komponentit eivät käsittele.

Muut lomakkeet sisältävät sovellusten varsinaisen sovelluslogiikan ja määrittävät otsikkoalueen, ohjelmoitavien painikkeiden ja sovellusikkuna-alueen varsinaisen sisällön ja välittää juurilomakkeelle tiedon muutoksista kyseisiin kenttiin. Aktiivisena oleva lomake myös käsittelee kaikki tapahtumat, joita aktivoitu käyttöliittymäkomponentti ei käsittele. Koska tapahtumat välittyvät yksittäiseltä komponentilta aina tarvittaessa juurilomakkeelle asti, ne on mahdollista ottaa kiinni viimeistään sovelluksen korkeimmalla tasolla ja käsitellä asianmukaisesti.

Tyypillisesti yksi lomake käsittää yksittäisen sovellusnäkymän, johon sisältyy käyttöliittymäkomponenttien luonti ja asemointi, mallin sisältämän tiedon asettaminen sekä sovelluksen suoritusjärjestyksen hallinnointi avaamalla tarvittaessa uusia lomakkeita tai dialogeja palautteena käyttöliittymätapahtumiin.

4.4. Muita käyttöliittymien sovelluskehyksiä

Mobiilisovellusten toteutukseen on edellä esiteltyjen sovelluskehysten lisäksi olemassa lukuisia vaihtoehtoja, esimerkiksi Windows Mobile -alustalle on tarjolla .NET Compact Framework [.NET Compact Framework, 2005], ja Palm OS -käyttöjärjestelmälle sen omien käyttöliittymäkirjastojen lisäksi avoimen lähdekoodin Waba-sovelluskehys [Waba, 2005].

5. Ohjelmistoalustojen ja sovelluskehysten vertailua

Tässä työssä esiteltiin mobiilisovellusten ohjelmistokehitystä ja siihen käytettävissä olevia ohjelmistoalustoja. Varsinaisesti mobiilisovellusten ohjelmistokehityksellä tarkoitetaan mobiililaitteessa ajettavien sovellusten suunnittelua toteuttamista. Tähän on tarjolla eri valmistajien tuottamia sovelluskehyksiä ja ohjelmsitorajapintoja, jotka tarjoavat pääsyn mobiililaitteen resursseihin.

5.1. Tietoturva

Ohjelmistoalustasta riippuen pääsy resursseihin vaihtelee. BREW-arkkitehtuuri antaa melko vapaan pääsyn laitteen resursseihin, minkä johdosta sovellusten levitys vaatii aina validoinnin ja tapahtuu operaattorien välityksellä. Toisaalta J2ME-sovellukset on suljettu toimimaan ns. hiekkalaatikko-mallin mukaisesti varsin suljetussa ympäristössä, josta pääsy resursseihin on rajattu tiettyihin ohjelmointirajapintoihin. Symbian-alustan tietoturvamalli perustuu myös lupamenettelyyn, mutta on BREW-alustaa joustavammin toteutettu.

5.2. Suunnittelumallit

Sovelluskehykset, ja etenkin niiden käyttöliittymäarkkitehtuuri, käyttävät esiteltyjen kehysten tapauksessa pääsääntöisesti MVC-suunnittelumallia, jossa tiedon rakenne, esitystapa ja sovelluksen ohjaus on eriytetty toisistaan. Tämä luo yhtäältä kehykselle helposti omaksuttavan rakenteen ja toimintaperiaatteen, mutta toisaalta pakottaa myös toteuttamaan sovelluksen kyseistä suunnittelumallia käyttäen.

Symbian-käyttöjärjestelmän ja UIQ-sovelluskehyksen tapauksessa mallia on käytetty varsin puhtaasti, kun taas BREW UI Toolkit -kehys kokoaa esitystavan ja ohjauksen samaan komponenttiin. MIDP-arkkitehtuurissa näin selkeää eritystä ei ole nähtävissä ja ohjelmistokehittäjä voi melko vapaasti valita haluamansa suunnittelumallin.

5.3. Käyttöliittymäkomponenttien asemoinnin hallinta

Käyttöliittymän ulkoasun asettelu on keskeinen osa sovelluksen toteutusta. Sekä UIQ että BREW UI Toolkit tarjoavat joustavat ja helppokäyttöiset työkalut käyttöliittymäkomponenttien dynaamiseen asemointiin näytöllä.

MIDP-arkkitehtuurin suurimpana heikkoutena voidaan pitää asemoinnin tuen alkeellisuutta käytettäessä korkean tason komponentteja. Toisaalta MIDP tarjoaa matalan tason komponenttien välityksellä mahdollisuuden monimutkaisempienkin käyttöliittymien toteutukselle helppokäyttöisyyden kustannuksella.

5.4. Käyttöliittymän adaptiivisuus

Käyttöliittymän adaptiivisuus on ohjelmistokehityksen kannalta oleellinen kysymys. Kehitykseen käytettäviä resursseja on mahdollista vähentää, jos sovellusta on mahdollista käyttää useissa eri laitteissa ilman erillistä sopeuttamista. Siirrettävyys on selkeästi ollut tavoitteena kaikkien käyttöliittymäsovelluskehysten suunnittelussa.

UIQ:ssa adaptiivisuus on toteutettu laitemäärittelyjen avulla, joten periaatteessa toteutusohjeita noudattavan sovelluksen tulisi toimia eri syötemenetelmiä ja näytön orientaatiota käyttävissä laitteissa muutoksitta. Natiiveja Symbian-sovelluksia ei ole mahdollista kuitenkaan käyttää muissa kuin Symbian-ympäristöjä tukevissa laitteissa, tosin sovellusten mukauttaminen eri käyttöliittymäarkkitehtuurien (S60, UIQ) välillä on mahdollista.

Myös BREW UI Toolkit -kehystä käyttämällä on valmistajan mukaan mahdollista toteuttaa adaptiivisia käyttöliittymiä käyttöliittymäsovelluskehyksen tasolla ilman erityistä sopeuttamista eri laiteympäristöihin. UIQ-sovellusten tavoin BREW-sovelluksia on mahdollista käyttää vain BREW-arkkitehtuuria tukevissa laitteissa.

MIDP-arkkitehtuurissa adaptiivisuus tukeutuu korkean tason komponenttien käyttöön. Käytettäessä matalan tason komponentteja adaptiivisuuden tukeminen vaatii sovelluksen arkkitehtuurin suunnittelua siten, että se itse pystyy mukautumaan kulloinkin käytettävissä oleviin laiteresursseihin. Eräs MIDP-arkkitehtuurin eittämätön vahvuus on sovellusten siirrettävyys eri laitealustoille, koska J2ME teknologia on saavuttanut varsin laajan tuen valmistajilta. Esimerkiksi sekä UIQ- että BREW-ympäristöihin on integroitu J2ME-suoritusympäristö.

6. Yhteenveto

Tässä työssä tarkasteltiin mobiilisovellusten ohjelmistokehitystä ja siihen tarjolla olevia ohjelmistoalustoja sekä sovelluskehyksiä. Mobiilisovellusten toteutuksessa käyttöliittymä on keskeisessä roolissa ja tästä johtuen työssä keskityttiin myös eri alustoilla käytettävissä oleviin käyttöliittymäsovelluskehyksiin.

Tarkastellut sovelluskehykset voidaan jakaa kahteen leiriin niiden monikäyttöisyyden perusteella: laitealusta- ja käyttöjärjestelmäsidonnaisiin ympäristöihin ja laitealustariippumattomiin ympäristöihin. Ainoastaan J2ME MIDP-arkkitehtuureineen on selkeästi käytettävissä eri laiteympäristöissä, jos kyseiseen ympäristöön on toteutettu vaadittu suoritusympäristö. Vastapainona siirrettävyydelle J2ME-sovellukset kärsivät Java-kielen tulkattavuudesta johtuvista suorituskykyrajoitteista.

Myös käyttöliittymäarkkitehtuurien osalta voidaan tehdä jaottelua eri toteutusmallien välillä. Sekä UIQ että BREW UI Toolkit rakentavat käyttöliittymän toiminnan vahvasti MVC-suunnittelumallin ympärille. MIDP-arkkitehtuuri on toisaalta jaettu itsessään kahteen osaan, joita voidaan sovelluksen tarpeiden mukaan yhdistellä. Tämän tarkastelun perusteella MIDP-arkkitehtuurin lähestymistapa vaikuttaa ongelmineenkin monikäyttöisemmältä kuin kilpailijansa.

Käytettävän alustan ja sovelluskehysten valintaan vaikuttaa useita tekijöitä, joista ohjelmistokehityksen kannalta keskeisinä voidaan pitää esimerkiksi käytettyä tietoturvamallia, käyttöliittymäarkkitehtuurin suunnittelumallia, käyttöliittymäkomponenttien asemoinnin hallintaa ja käyttöliittymän adaptiivisuutta. Jokaisella tarkastelluista alustoista ja sovelluskehyksistä on näillä osa-alueilla omat vahvuutensa ja heikkoutensa. Symbian ja BREW tukeutuvat selkeään käyttöliittymän arkkitehtuurimalliin sekä laiteympäristöön nivoutuvan ohjelmistoalustan suomaan suorituskykyyn, kun J2ME puolestaan antaa ohjelmistokehittäjälle vapaammat kädet sovellusarkkitehtuurin suhteen suorituskyvyn kustannuksella.

Viiteluettelo

[.NET Compact Framework, 2005] Microsoft Corporation, .NET Compact Framework. Saatavilla osoitteessa *http://msdn.microsoft.com/smartclient/understanding/netcf/*

[Abrahamsson *et al.*, 2004] Pekka Abrahamsson, Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Juho Jäälinoja, Mikko Korkala, Juha Koskela, Pekka Kyllönen and Outi Salo, Mobile-D: an agile approach for mobile application development. *OOPSLA Companion 2004*, 174-175.

[BREW, 2005] Qualcomm, Inc., BREW® and J2ME™: A Complete Wireless Solution for Operators Committed to Java™, 2005. Saatavilla osoitteessa *http://brew.qualcomm.com/brew/en/img/about/pdf/brew_j2me.pdf*

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

[CLDC, 2003] Sun Microsystems, Inc., Connected Limited Device Configuration Specification,Version 1.1, 2003. Saatavilla osoitteessa *http://jcp.org/aboutJava/communityprocess/final/jsr139/*

[de Jode and Turfus, 2005] Martin de Jode and Colin Turfus, Symbian OS System Definition, 2005.

[Hemphill, 2004] David Hemphill, Take the High Road when Creating MIDP User Interfaces, 2004. Saatavilla osoitteessa *http://www.devx.com/wireless/Article/21262*

[i-mode, 2005] NTT DoCoMo, What's i-mode, 2005. Saatavilla osoitteessa *http://www.nttdocomo.co.jp/english/p_s/imode/what/what/*

[J2ME, 2005] Sun Microsystems, Inc.,Java 2 Platform, Micro Edition, 2005. Saatavilla osoitteessa *http://java.sun.com/j2me/j2me-ds.pdf*

[Kamada, 1998] Tomihisa Kamada, Compact HTML for Small Information Appliances, 1998. Saatavilla osoitteessa *http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/*

[Leghari, 2003] Nauman Leghari, Tools and Platforms: Choices for a Mobile Application Developer, 2003. Saatavilla osoitteessa *http://weblogs.asp.net/nleghari/articles/44057.aspx*

[MIDP, 2002] JSR 118 Expert Group, Mobile Information Device Profile for Java™ Micro Edition, Version 2.0, 2002. Saatavilla osoitteessa *http://jcp.org/aboutJava/communityprocess/final/jsr118/*

[Mýren, 2005] Sara Mýren, UIQ Product Description, 2005. Saatavilla osoitteessa *http://www.uiq.com/documentation/Web version of UIQ 3.0 Product Description October 2005.pdf*

[Nijdam, 2004] Marc Nijdam, Introducing: BREW UI Toolkit, 2004. Saatavilla osoitteessa *http://www.devx.com/wireless/Link/22084*

[Näsi, 2006] Juuso Näsi, Sovelluslogiikan eriyttäminen käyttöliittymästä. Roope Raisamo (toim.), Käyttöliittymien ohjelmistoarkkitehtuurit, Tampereen yliopisto, Tietojenkäsittelytieteiden laitos, raportti B-2006-x, 2006.

[Opera Mini, 2005] Opera Software ASA, Opera Mini™, 2005. Saatavilla osoitteessa *http://www.opera.com/products/mobile/brochures/OperaMini.pdf*

[Ortiz, 2002] C. Enrique Ortiz, An Introduction to the Symbian OS™ Platform for Palm OS® Developers, 2002. Saatavilla osoitteessa *http://www.metrowerks.com/pdf/IntroSymbianOSforPalmDevelopers.pdf*

[Piroumian, 2002] Vartan Piroumian, *Wireless J2ME Platform Programming*, Prentice Hall PTR, 2002.

[Rischpater, 2004] Ray Rischpater, Build Better UIs with a New Brew Framework, 2004. Saatavilla osoitteessa *http://www.devx.com/wireless/Article/21982*

[S60, 2005] Nokia Corporation, Series 60 Platform 3rd Edition Overview, 2005 . Saatavilla osoitteessa *http://www.series60.com/pics/pdf/Series_60_Platform_3rd_Edition_Overview_Oct 05.pdf*

[SophiaFramework, 2005] Sophia Cradle, Inc., SophiaFramework C++ Class Library for BREW, 2005. Saatavilla osoitteessa *http://www.s-cradle.com/english/products/sophiaframework/*

[Symbian, 2005a] Symbian, Ltd., Symbian OS Technology, 2005. Saatavilla osoitteessa *http://www.symbian.com/technology/technology.html*

[Symbian, 2005b] Symbian, Ltd., Symbian OS Overview, 2005. Saatavilla osoitteessa *http://www.symbian.com/technology/OSoverview/OSoverview_v9.html*

[Waba, 2002] Wabasoft, The Waba Software Development Kit, 2002. Saatavilla osoitteessa *http://www.wabasoft.com/spec_sdk.shtml*

[WAP, 2001] Open Mobile Alliance, Wireless Application Protocol Architecture Specification, 2001. Saatavilla osoitteessa *http://www.wapforum.org/what/technical.htm*

[WML, 2001] Open Mobile Alliance, Wireless Markup Language Specification, 2001. Saatavilla osoitteessa *http://www.wapforum.org/what/technical.htm*

[XHTML, 2002] W3C HTML Working Group, XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), 2002. Saatavilla osoitteessa *http://www.w3.org/TR/xhtml1/*

[XHTML MP, 2001] Wireless Application Protocol Forum, Ltd., XHTML Mobile Profile, 2001. Saatavilla osoitteessa *http://www.openmobilealliance.org/tech/affiliates/wap/wap-277-xhtmlmp-20011029-a.pdf*

Äänimaisemien käyttöliittymäarkkitehtuurit

Anssi Kainulainen

Tiivistelmä.
Tässä tutkimuksessa esittelen kuinka tietoisuutta voidaan tukea jokapaikan tietotekniikan ympäristöissä äänen avulla. Esittelen sovelluksen, joka käyttää äänimaisemia tukemaan ihmisten tietoisuutta toistensa läsnäolosta toimistoympäristössä. Sovellus on osa laajempaa jokapaikan tietotekniikan ympäristöä. Käsittelen erilaisia äänimaisemien rakentamisen ja hallinnan menetelmiä. Lopuksi esittelen arkkitehtuurin äänimaisemien suunnittelua ja hallintaa varten. Arkkitehtuuri perustuu agentteihin ja ohjauskieleen, ja se korostaa uudelleenkäytettävyyttä ja laajennettavuutta, ja se rakentuu yhteisen Jaspis-kehysohjelmiston varaan.

Avainsanat ja -sanonnat: Tietoisuus, jokapaikan tietotekniikka, interaktiivinen ääni.
CR-luokat: H.5.1, H.5.2

1. Johdanto

Tietoisuuden tukeminen jokapaikan tietotekniikan ympäristöissä hyödyttää suuria kohdeyleisöjä laajassa valikoimassa tehtäviä, sosiaalisia tilanteita ja fyysisiä ympäristöjä. Ihmiset voivat olla vuorovaikutuksessa keskenään normaalisti, ja työskennellä yhdessä missä tahansa tehtävissä, sillä tietoisuuden tuki on tarjolla ympäristössä itsessään. Tietoisuutta tuetaan rauhallisella ja häiritsemättömällä tavalla, käyttäen hyväksi ihmisten periferaalista aistimiskykyä.

Jokapaikan tietotekniikan ympäristöt asettavat omat haasteensa tietoisuuden tukemiselle. Tiedon kerääminen käyttäjistä, heidän työtehtävistään ja sosiaalisista tilanteistaan on paljon monimutkaisempi asetelma kuin perinteisessä tietokoneavusteisessa ryhmätyön tuessa (CSCW), joka keskittyy henkilökohtaisilla tietokoneilla toimiviin sovelluksiin ja järjestelmiin. Jokapaikan tietotekniikassa on tarjolla suurempi määrä informaatiota, ja aihealueiden ja sovelluskohteiden kokoluokka on suurempi. Tiedon kerääminen ja

päättelymenetelmät ovat myös monimutkaisempia. Tietolähteet ovat yleisesti esimerkiksi kokoelma tunnistustuloksia puheen- ja puhujantunnistimilta, paikannuspalveluilta ja muilta tiedonkeruuohjelmistoilta. Tämän monimutkaisuuden hallinta on jo itsessään suuri haaste.

Eräs keskeisiä haasteita tietoisuuden tukemiselle jokapaikan tietotekniikassa on tiedon esityksen hallinta. Kun informaatio on kerätty, yhdistetty ja tarvittavat päätelmät tehty, tietoisuutta tukevien järjestelmien täytyy päättää mitä tietoa esitetään, koska, kenelle ja missä muodossa. Liiallinen yksityiskohtaisuus ja häiritsevät esitykset helposti kuormittavat kuulijansa kognitiivista kapasiteettia liiaksi. Rauhallisen ja häiritsemättömän tietoisuuden tuen saavuttamiseksi tarvitaan erityisiä esityksen suunnitelun ja hallinnan menetelmiä.

Tässä tutkimuksessa esitän ihmisten tietoisuutta toistensa läsnäolosta toimistoympäristössä tukevan sovelluksen. Sovelluksessa käytetään rauhallisia ja jatkuvia, kuten lintujen laululle ja askelten äänille [Mäkelä et al., 2003] perustuvia äänimaisemia välittämään informaatiota kuitenkaan kuulijoiden kognitiivista kapasiteettia kuormittamatta. Tietoisuutta tukeva sovellus on osa laajempaa jokapaikan tietotekniikan järjestelmää [Kainulainen et al., 2005], joka auttaa ihmisiä toimistoympäristössä monilla tavoin, esimerkiksi opastuksen, puheviestinnän ja muiden palveluiden kautta.

Tehokkaiden läsnäolotietoa esittävien äänimaisemien rakentamisen vuoksi tarvitaan ohjelmistoarkkitehtuuri, jolla on riittävät ominaisuudet äänet ja kompositioiden muokkaamiseen ja hallintaan. Erityisesti täytyy varmistaa esityksen yhtenäisyys ja sopeutuvuus eri tilanteisiin. Tähän vaaditaan erityisiä tekniikoita siirtymien, ajoituksen, äänten yhdistelyn, ulkopuolisiin tietolähteisiin sopeutumisen ja rakenteisen esityksen monimutkaisuuden hallintaan. Tässä tutkimuksessa esitellään arkkitehtuuri, jonka rakenne ja perustoiminnallisuuksien joukko mahdollistavat sovellussuunnittelijoita ja äänimaisemien suunnittelijoita rakentamaan tehokkaita äänen avulla tietoisuutta tukevia sovelluksia. Arkkitehtuuri perustuu kokemuksiimme ääntä käyttävien jokapaikan tietotekniikan sovellusten rakentamisessa [Kainulainen et al., 2005] ja arvioinnissa [Mäkelä et al., 2001]. Teknisesti arkkitehtuuri laajentaa yleisluontoista puhepohjaisten ja jokapaikan tietotekniikan Jaspis-kehysohjelmistoamme [Turunen et al., 2005].

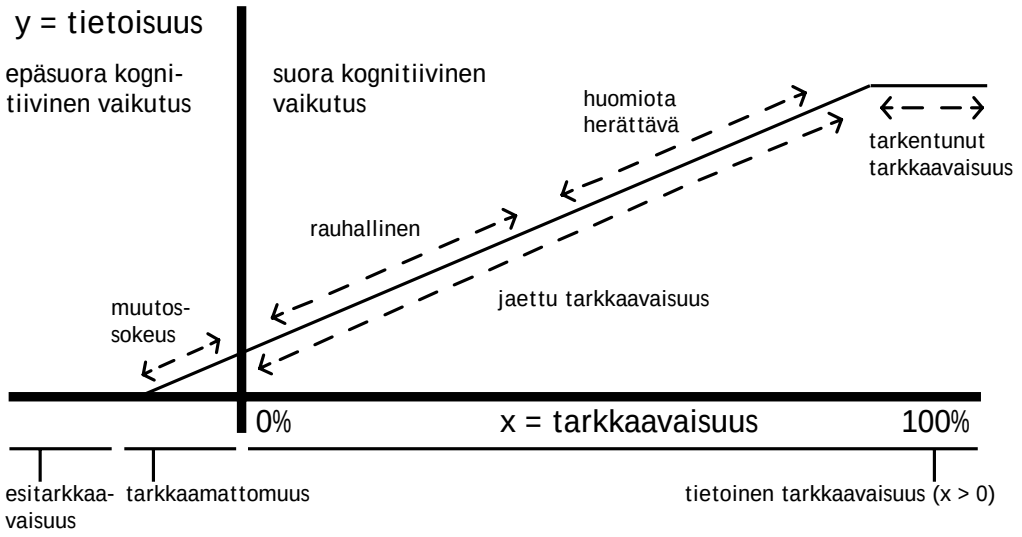
Seuraavaksi esittelen periferaalisen havainnoinnin psykologisia mekanismeja, ja kuinka niitä voidaan käyttää hyväksi. Tämän jälkeen esittelen itse sovelluksen, joka tukee tietoisuutta rauhallisten äänimaisemien kautta. Sen

jälkeen tarkastelemme tarkemmin erilaisia haasteita sellaisten äänimaisemien tuottamisessa: tiedon kuvautumista esitykseksi, esityksen yhtenäisyyden ylläpitoa, ja niiden hallintaa muilla keinoin. Tämän jälkeen kuvaamme arkkitehtuurin joka on suunniteltu ottamaan huomioon nämä vaatimukset. Lopuksi keskustellaan ratkaisuista ja jatkotutkimussuunnitelmista.

2. Tietoisuus ja havaintokyvyn reunamat

Dourish ja Bellotti [Dourish & Bellotti, 1992] määrittelevät tietoisuuden (awareness) ”ymmärrykseksi muiden toiminnasta, joka tarjoaa kontekstin omalle toiminnalle.” Yksilön sosiaalinen vaikutus muihin voidaan nähdä sosiaalisten siteiden, fyysisen läheisyyden ja ajallisen välittömyyden summana. Tietoisuuden rooli, jopa vain läsnäolotietoisuuden, on tärkeätä yhteistyölle ja yksilön sosiaaliselle elämälle. Suurin ongelma tietoisuuden tukemisessa tietokoneen avulla on informaation tarjoaminen häiritsemättömällä tavalla.

Kognitiivisesta näkökulmasta tietoisuus on läheisessä yhteydessä tarkkaavaisuuteen. Tarkkaavaisuus tarkoittaa psyykkisen toiminnan suuntaamista ja valikoivuutta. Tietämisen ja tietoiseksi tulemisen prosessissa tarkkaavaisuuden siirtyminen ja siirtäminen ovat tärkeitä. Tarkkaavaisuuden siirrossa käytetään hyväksi aistien reunamia eli periferaalista havaintokykyä (periphery of sense). Jotkut teoriat [Matthews et al., 2003] jakavat tämän prosessin neljään kategoriaan: esitarkkaavaisuuteen (preattention), tarkkaamattomuuteen (inattention), jaettuun tarkkaavaisuuteen (divided attention) ja tarkentuneeseen tarkkaavaisuuteen (focused attention). Esitarkkaavaisuuden kategorian kohteet eivät välttämättä vaikuta tietoisuuteen lainkaan. Tarkkaamattomuuden kategorian kohteista ei olla tietoisia tiedostetusti, mutta informaatio saattaa vaikuttaa käytökseen alitajuisesti. Välittynyt viesti voi vaikuttaa ihmisen käytökseen ja auttaa häntä tekemään päätöksiä, vaikkei hän itse osaa syitä perustella. Jaetun ja tarkentuneen tietoisuuden kohteet ovat kumpikin ihmisen tietoisuudessa.



Kuva 1. Siirtymät tarkkaavaisuuden kategorioiden välillä.

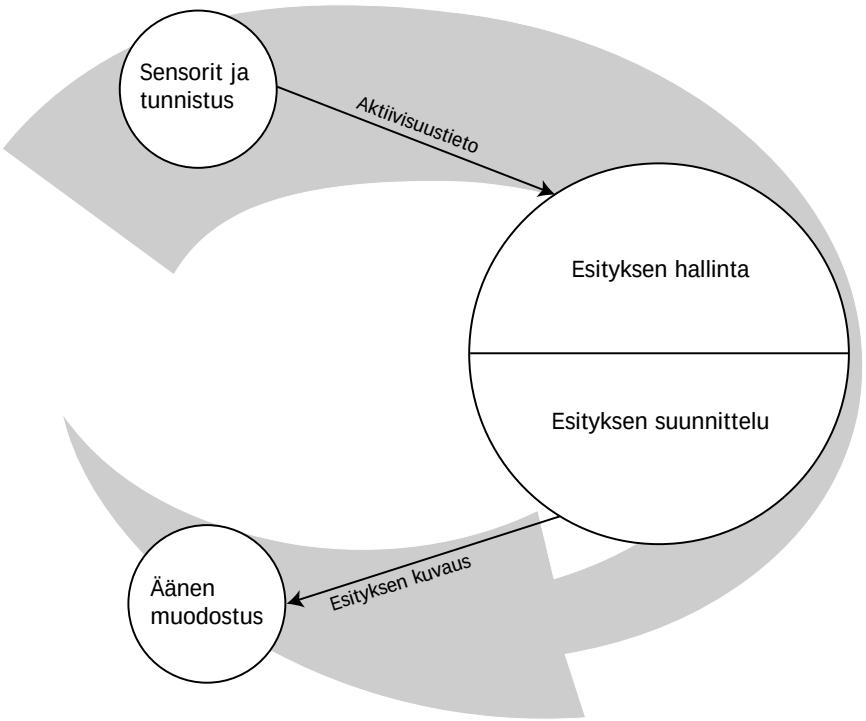
Tarkkaavaisuus siirtyy näiden kategorioiden välillä joko tahdonalaisesti tai tahattomasti riippuen tilanteesta. Tämä tarkoittaa sitä, että äänen avulla tietoisuutta tukevien sovellusten täytyy olla häiritsemättömiä siten, etteivät ne kaappaa huomiota tarpeettomasti katkoisuuden tai huonon äänenlaadun vuoksi. Huomion herättämisen tason täytyy myös sopia sosiaaliseen tilanteeseen ja kulloisiinkin työtehtäviin. Esitysten täytyy olla johdonmukaisia, yhtenäisiä ja jatkuvia.

Jokapaikan tietotekniikan sovellukset, jotka käyttävät hyväksi ihmisten periferaalista havaintokykyä, voivat välittää informaatiota mistä ihmiset ovat tietoisia, mutta joihin eivät ole kohdistaneet tarkkaavaisuuttaan [Wisneski et al., 1998]. Tarkkaamattomuutta ja jaettua tarkkaavaisuutta käytetään hyväksi rauhallisissa ja häiritsemättömissä esityksissä. Tarkkaamattomuus ei kuormita ihmisen kognitiivista kapasiteettia. Tässä tutkimuksessa tarkastellaan kuinka tämä voidaan saavuttaa äänisovelluksilla, jotka on suunniteltu välittämään tietoisuutta tukevaa informaatiota häiritsemättömillä tavoilla. Seuraavaksi kuvaan sovelluksen jossa käytetään rauhallisia äänimaisemia infomraation esitysmuotona.

3. Tietoisuutta tukeva äänisovellus

Tukeaksemme työryhmän tietoisuutta olemme toteuttaneet sovelluksen, joka yrittää pitää ihmiset tietoisina toistensa läsnäolosta toimistoympäristössä. Sovellus on osa laajempaa jokapaikan tietotekniikan ympäristöä, joka auttaa

ihmisiä jokapäiväisissä tehtävissään toimistotyötilanteissa [Kainulainen et al., 2005]. Järjestelmä kerää läsnäolotietoa eri lähteistä, esimerkiksi seuraamalla heidän tietokoneen käyttöään. Läsnäolotieto esitetään ympäristöön upotettujen kaiuttimien kautta.



Kuva 2. Ääntä esittävän sovelluksen yleisrakenne.

Kuvassa 2 esitetään sovelluksen yleisrakenne. Ensimmäinen vaihe on tiedonkeräys eri lähteistä sensorien ja tunnistustekniikoiden avulla, kuten implisiittisen tiedon keräys liikeantureista, ja eksplisiittisen tiedon keräys henkilökohtaisista tietokoneista (esimerkiksi näppäimistön käyttö tai kalenterisovellukset). Toisessa vaiheessa raakatiedosta poimitaan hyödylliset piirteet abstraktimmalla tasolla, esimerkiksi sääntöpohjaisilla tai tilastollisilla menetelmillä. Tässä vaiheessa tehdään myös esityksen hallinnallisia päätöksiä, kuten keiden läsnäolo tulee esittää missäkin järjestyksessä. Kolmannessa vaiheessa läsnäolotieto yhdistetään saatavilla oleviin äänellisiin esitystapoihin. Neljännessä ja viimeisessä vaiheessa edellä tuotettu esityksen kuvaus annetaan äänenmuodostuslaitteelle, joka soittaa lopullisen esityksen.

Tietoisuutta tukevassa sovelluksessamme informaatiota esitetään äänimaisemien muodossa. Äänimaisemilla tarkoitetaan jonkin paikan koko äänienergiaa eli ääniympäristöä siinä määrin kuinka sen kuulijat tulkitsevat ja

sovellus osittain tuottaa. Äänimaisemat koostuvat temaattisesti samankaltaisista äänistä, kuten eri lintujen lauluista. Työryhmän yksittäisen henkilön aktiivisuutta ja läsnäoloa vastaa yhden linnun laulu. Aikaisin aamulla ja myöhään illalla, kun vain muutama ihminen on paikalla, äänimaisema muistuttaa hiljaista järveä luonnon keskellä, jossa vain muutaman linnun laulu kaikuu järvenselän yli. Keskipäivän kiireisinä hetkinä useamman linnun ääntely luo aktiivisemman ilmapiirin. Olemme tehneet kokeita erilaisialla äänillä, kuten askeleiden äänillä [Mäkelä et al., 2003]. Olemme myös kokeilleet erilaisia äänimaisemien ja kompositioiden rakentamismenetelmiä.

Käytännössä sovellus kerää läsnäolo- ja aktiivisuustietoa seuraamalla ihmisten tietokoneenkäyttöä. Tietyn aikaikkunan sisällä tapahtunut toiminta tulkitaan tarpeeksi tuoreeksi aktiivisuudeksi, ja tiedonkeruulähteen sijainti tulkitaan ko. ihmisen sijainniksi.

Sovellus yhdistää ääniä dynaamisesti tiettyjen rajoitteiden mukaisesti. Jotta äänimaisema ei ruuhkautuisi kiireisimmilläkään hetkillä, vain tietty määrä ääniä sallitaan yhtä aikaa soitettavaksi. Äänet sovitetaan osittain päällekkäin soiviksi, ja uusi ääni voidaan lisätä äänimaisemaan vain edellisten äänten ollessa sopivan rauhallisessa kohtaa. Rauhalliset kohdat on merkitty talteen ääninäytekohtaisesti etukäteen. Koska ihmiset sijaitsevat saman käytävän varrella, heidän istumajärjestyksensä ohjaa äänten soittojärjestystä. Nämä säännöt aiheuttavat sen, että äänimaisema muistuttaa ympäristön liikkuvaa silmäilyä.

Sovellusta toteutettaessa törmäsimme moniin haasteisiin. Erityisesti kaivattiin työkalua äänimaiseman kompositiota ja dynaamista hallintaa varten. Sovelluksella tulisi olla helppokäyttöinen rajapinta iteratiiviseen suunnitteluun ja erilaisten kompositioiden kokeilemiseen. Toisaalta taas sovelluksen pitäisi olla joustava muiden sovellusten ja informaatiolähteiden kanssa kommunikointiin. Nämä kaksi ääripäätä pitäisi saattaa yhteen dynaamisella ja vuorovaikutteisella tavalla. Me ratkaisimme ongelma käyttämällä kuvaus- tai ohjauskieltä (markup language, scripting language), jonka avulla äänimaiseman komposition suunnittelija tai säveltäjä voi määritellä esityksen rakenteen ja toiminnallisuuden. Samalla tämä kuvaus- tai ohjauskieli mahdollistaa selkeästi määritellyn muodon, jota sovellussuunnittelijat voivat käyttää toteuttaessaan sovelluksia. Sovellukset käyttävät agentteja, joiden toiminnallisuus on myös konfiguroitavissa samaan tapaan. Tämä mahdollistaa nopeat kokeilut ja komponenttien uudelleenkäytettävyyden. Ohjauskieli ja agentit esitellään tarkemmin kohdassa

5. Ennen sitä, kohdassa 4 tarkastellaan mitä vaatimuksia ja teknisiä haasteita äänentuotanto ja esittäminen asettavat tälle arkkitehtuurille.

4. Äänimaisemat

Äänimaisemien esityksen tuottamisen kolme tärkeintä aluetta ovat tiedon kuvautuminen, jatkuvuuden ja yhtenäisyyden varmistaminen, sekä komposition hallinta. Tieto voi kuvautua konkreettisiin esitysmuotoihin monilla tavoilla, jotka eivät kaikki ole triviaaleja, ja vaativat välillä hieman taiteellista silmää. Ääniesityksen jatkuvuus ja yhtenäisyys vaikuttavat sen rauhallisuuteen ja häiritsemättömyyteen. Kokonaisten kompositioiden määrittellyyn ja esityksen aikaiseen hallinta on monia keinoja. Seuraavissa alikohdissa tarkastelemme erityisesti ääniesitysten adaptiivisuutta ja vuorovaikutteisuutta.

4.1. Käsitteiden kuvautuminen esitysmuotoihin

Äänimaisemat muodostetaan yhdistämällä monia ääniä, sekä määrittelemällä sääntöjä ja vaihtelua niiden ominaisuuksiin, käyttäytymisen ja vuorovaikutukseen. Muutokset äänimaisemassa, siinä liikkuvissa ja toistensa kanssa vuorovaikuttavissa objekteissa voidaan helposti ymmärtää kokonaisuutena. Merkityksellisen informaation esittämiseksi äänen avulla täytyy määritellä kuvaus informaation ja sen äänellisen esitystavan välillä. Tämä kuvaus voi olla:

- Suora yhteys muuttujan ja äänen välillä, missä muuttujan arvon muutos aiheuttaa muutoksen äänessä tai sen ominaisuuksissa suoraan, jatkuvana virtana tai rajallisina ikoneina.
- Staattinen, binäärinen tai dynaaminen, missä staattiset äänet voivat esittää tunnelmallista, oletusarvoista tai muuttumatonta informaatiota, binääriset äänet voivat esittää henkilön läsnäoloa tai muuta tosi/epätosi-tyyppistä informaatiota, ja dynaamiset äänet voivat esittää informaatiota prosesseista tai vuorovaikutuksesta.
- Hierarkkisesti rakentunut, missä esimerkiksi äänet voivat kuulua samanaikaisesti useampaan kategoriaan esimerkiksi melodian, äänensävyn tai harmonian kautta.
- Kokonainen sävellys, kompositio tai virtuaalisten objektien rakennelma, jossa elementit nähdään yksilöllisinä objekteina, joiden käyttäytymistä ja vuorovaikutusta määrittelevät säännöt ja toimintamallit.

Kun äänimaisema on määritelty, se hahmontaa eli tehdä samaan tapaan kuin kolmiulotteinen grafiikka, jatkuvasti ja reaaliaikaisesti. Yksittäisten elementtien vuorovaikutus luo koko esityksen rikkauden. Kaikki tämä vaatii tapoja sitoa eri tyyppistä informaatiota äänielementteihin, niiden ominaisuuksiin, ja sävellyksisiin rakenteisiin eri abstraktiotasoilla. Näitä voidaan muokata dynaamisesti reaaliajassa riippuen syötteistä, niiden toimintaa säätelevien sääntöjen mukaan.

4.2. Ajoittaminen, siirtymät, jatkuvuus ja yhtenäisyys

Yhtenäisten ja jatkuvien esitysten luomisessa äänielementtien ajoittaminen ja synkronointi on tärkeätä. Uudet äänielementit ja vanhojen äänielementtien muutokset täytyy ajoittaa oikein jo soivaan esitykseen, sen elementteihin ja musiikillisiin katkelmiin. Muutos esitettävässä informaatiossa aiheuttaa muutoksen esityksessä, ja näiden muutoskohtien, eli siirtymien, täytyy olla sulavia ja luonnollisia. Siirtymien ajoittaminen ja käsittely vaatii oman notaationsa ja hallintajärjestelmänsä.

Kun on tehty päätös muutoksesta esityksessä, siirtymä aikaisemmasta tilasta seuraavaan täytyy toteuttaa sujuvasti. Siirtymä voi olla karkeimmillaan hiljaisuus kahden peräkkäisen elementin välillä tai niiden suora yhdistäminen, ristikkäin tapahtuva äänenvoimakkuuksien vaihdos tai yhteisen rytmin synkronoitu päällekkäistäminen, efektien käyttö siirtymän helpottamisessa, tai parhaimmillaan täysin saumaton siirtymä. Saumaton siirtymä voi olla vaikea toteuttaa jopa musiikillisesti, saati sitten teknisesti automatisoituna. Haastavimmillaan ääniesityksen ja sitä tuottavan järjestelmän täytyy olla valmiita tekemään siirtymä millä hetkellä hyvänsä. Mitä suuremmasta muutoksesta on kyse, sitä vaikeampi se on toteuttaa saumattomasti, etenkin lyhyessä ajassa.

Teknisestä näkökulmasta siirtymä voidaan toteuttaa hyppäämällä etukäteen merkityssä kohdassa esitystä, joko saman ääniraidan sisällä tai eri esitysten välillä. Siirtymä voidaan toteuttaa myös kerrostettuna, jolloin esitykseen lisätään tai siitä poistetaan ääniä tai raitoja kerros kerrokselta. Siirtymämatriisia voidaan myös käyttää luomalla jokaista mahdollista perättäisten ääniraitojen yhdistelmää varten erillinen siirtymäraita. On myös mahdollista seurata esitettävän musiikin harmoniallista rakennetta [Temperley, 1997], jolloin uusi äänikerros voi alkaa harmoniassa edellisten kanssa. Sointukarttoja [Mugglin, 2005] voidaan käyttää

määrittelemään vieläkin monimutkaisempia sääntöjen tiettyjen sointujen välillä siirtymiseen.

Joskus sävellyksen elementit alkavat toistua usein, mistä saattaa tulla jopa ärsyttävää. Vaihtoehtoisia äänielementtejä voidaan satunnaistaa sääntöjen avulla. Niin kutsuttu generatiivinen musiikki [Eno, 1996] on erikoistunut satunnaisuuteen ja määrittelemättömyyteen.

Äänimaisemasovelluksessa täytyy olla notaatio ja hallintajärjestelmä elementtien ajoittamiseen yhdistämisvaiheessa, ja eri tilojen välillä tapahtuvien siirtymien ohjaukseen. Järjestelmän ja notaation täytyy olla laajennettavissa, sillä on mahdotonta ennustaa kaikkia mahdollisia tapoja, joilla joku haluaa ajoittamista ja siirtymistä suorittaa.

4.3. Ääniesitysten kompositioiden hallinta

Mitä suurempi määrä informaatiota täytyy esittää, sitä vaativampaa on ääniesityksen ajonaikainen hallinta sovelluksissa. Kasvavan monimutkaisuuden vuoksi on hyödyllistä eriyttää sovelluksen suunnittelu sovelluksen ohjelmoijien ja esityksen suunnittelijoiden tai säveltäjien kesken. Ääniesitystä tuottavan ja hallitsevan ohjelmistokomponentin täytyy olla joustava ja monipuolinen, ja sen täytyy tarjota selkeä työkalu esitysten suunnitteluun.

Äänimaisema on esitykseltään jatkuva, mutta se voi reagoida esitettävän informaation muutoksiin reaaliajassa, viiveellä tai jopa ennakoimalla tapahtumia. Suora reagointi on reaaliaikaista, mikä vaatii rinnakkain toimivia itsenäisiä komponentteja. Viiveellinen reagointi tarkoittaa, että muutokset on ajoitettu seuraaviin esteettisesti tai psykologisesti soveltuviin hetkiin. Ennakointi tarkoittaa valmistautumista etukäteen määriteltyihin ja tiedettyihin tapahtumaketjuihin tai tilastollisesti todennäköisiin tilanteisiin. Ennakoimalla tilanteita voidaan esityksessä tapahtuvan muutoksen vaatimaan siirtymää valmistautua jo ennen itse muutoksen tapahtumista, ts. itse siirtymä voi alkaa jo ennen kuin muutos on suoraan havaittu.

Eräs suurista haasteista on määritellä kuinka ääniesitystä ohjaava ohjelmistokomponentti kommunikoi muun järjestelmän kanssa, mitä esityksen elementtejä voidaan muuttaa, ja kuinka järjestelmä laukaisee nämä muutokset. Tietokonepeleistä [Land & McConnell, 1994; Microsoft, 1998; Microsoft, 2005; Grigg, 2003] löytyy hyviä esimerkkejä näiden ongelmien ratkaisuun, sillä niissä on jo pitkään käytetty erilaisia laukaisimia ääniesitysten muutosten ohjaamiseen. Samankaltaisia laukaisintekniikoita voidaan hyödyntää reagoimaan myös

fyysisen maailman ilmiöihin. Tätä tarkoitusta varten ääniesityskomponentti tarvitsee avoimen rajapinnan ja tavan kommunikoida takaisinpäin ulkoisten sovellusten ja tietolähteiden kanssa.

Tapahtumien ja niiden käsikirjoittamisen (scripting) kautta voidaan myös luoda yhteys äänimaisemien ja taustalla olevien tietolähteiden välille. Käsikirjoittaminen tarkoittaa tässä kontekstissa tapaa rakentaa monimutkaisia yhdistelmiä ja tapahtumia yksittäisistä säännöistä ja komennoista yksinkertaisen annotaatiomekanismin tai kielen avulla. Teknisesti käsikirjoittaminen vaatii tarpeeksi yleisen kielen, jotta muita äänimaiseman hallintamenetelmiä ja komponentteja voidaan käyttää, sekä tapahtumia voidaan määritellä selkeällä ja helpolla tavalla. Tämän kaltaista käsikirjoittamista voidaan ajatella perinteisen musiikillisen säveltämisen kanssa rinnakkaisena ja jossain määrin päällekkäisenä tapana määritellä ääniesitys. On olemassa monenlaisia ohjaus- ja kuvauskieliä sekä notaatioita, rinnakkaisista ääniohjelmointikielistä, kuten ChucK [Wang & Cook, 2003], synteesinkuvauskieliin, kuten SSML [Taylor & Isard, 1997].

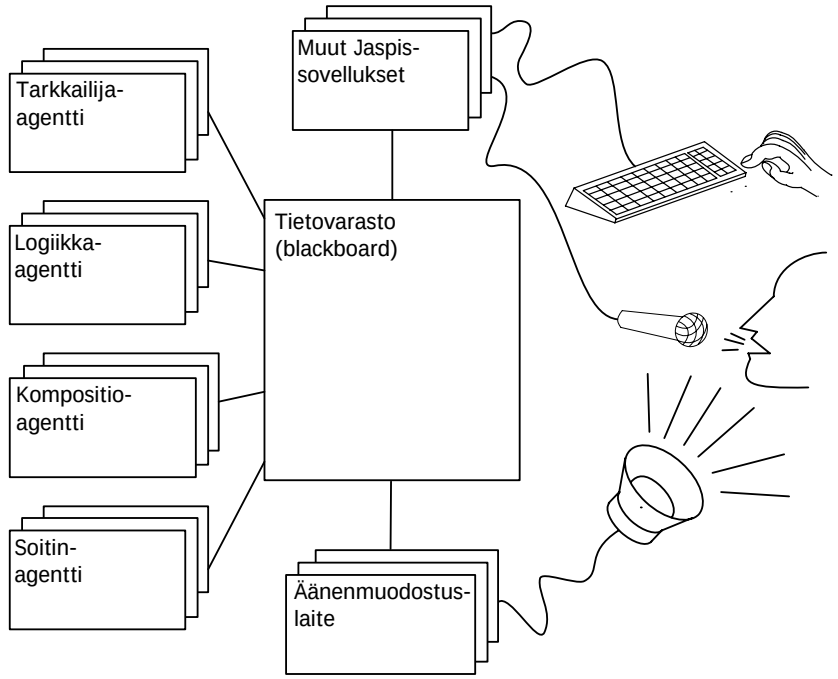
Tehokas äänimaiseman hallintamekanismi vaatii yhteisen abstraktiotason erikoisalueille, jotta sovellussuunnittelijat ja ääniesityksen säveltäjät voivat tehdä yhteistyötä. Saman mekanismin täytyy tarjota keinot suoran, viiveisen tai ennakoidun muutoksen hallintaan. Tarvitaan myös avoin rajapinta ulkoisille informaatiolähteille ja callback-mekanismi. Reaaliaikaista reagoivuutta ja rinnakkain ajettavia komponentteja vaaditaan tiettyihin tilanteisiin. Ohjauskieltä tarvitaan konfiguroitavuuden ja laajennettavuuden saavuttamiseksi. Seuraavassa kohdassa kuvaan arkkitehtuurin, joka vastaa näissä alikohdissa kerättyjä vaatimuksia.

5. Arkkitehtuuri äänimaisemien suunnitteluun, tuottamiseen ja hallintaan

Tässä kohdassa kuvataan arkkitehtuuri, joka on suunniteltu jokapaikan tietotekniikan sovelluksissa käytettävien äänimaisemien suunnitteluun, tuottamiseen ja hallintaan. Arkkitehtuuri ottaa huomioon jokapaikan tietotekniikan ympäristöjen erityisvaatimukset, sillä kohdeyleisö liikkuu tilassa ja sosiaalisesta tilanteesta toiseen, ja työtehtävät ja ympäristön laitteiston tekniset kyvyt vaihtelevat. Lisäksi erilaiset käyttäjäryhmät [Turunen et al., 2005] otetaan huomioon. Esimerkiksi, samasta esityksestä voidaan helposti luoda erilaisia versioita, jotka sopeutuvat vaikkapa kuulorajoitteisten heikentyneisiin kykyihin.

Teknisesti arkkitehtuuri rakentuu puhe- ja jokapaikan tietotekniikan sovelluksiin suunnitellun Jaspis-kehyksen [Turunen et al, 2005] päälle. Koska tässä tutkimuksessa esitellyt arkkitehtoniset ratkaisut perustuvat yhteiseen ohjelmistokehykseen, sen tarjoama toiminnallisuus on tarjolla myös muille sovelluksille, jotka on rakennettu Jaspiksen varaan. Arkkitehtuurin rakenne ja toiminnallisuus on suurelta osin olioajattelun mukainen. Arkkitehtuurissa hajautetaan ääniesityksen tuotanto ja hallinta alitehtäviksi erilaisille agenteille. Agentit ovat pieniä, mutta niitä on runsaasti. Tämä mahdollistaa helpon uudelleenkäytettävyyden ja toiminnallisuuden laajennettavuuden. Agentit sekä perivät toiminnallisuutta, että hyödyntävät ohjauskielen kautta konfiguroitavuutta.

Tietoisuutta tukeva informaatio ja sen äänellinen esitysmuoto ovat tallennettuina jaetussa järjestelmän tietovarastossa eri abstraktiotasoilla. Kaikki agentit voivat muokata tätä esitystä riippuen omista tehtävänkuvauksistaan ja tilanteesta. Agenttien toiminnallisuus on määritelty kovakoodattuna niihin sisäisesti sekä erikseen ohjauskielen avulla erilaisina konfiguraatioina. Agenttien luoma ja hallitsema esitys tuotetaan äänenmuodostuslaitteessa, joka tulkitsee ja jäsentää agenttien antamat komennot ja pyynnöt. Kommunikaatio toimii kumpaankin suuntaan, joten esityksen sisäiset tapahtumat voivat laukaista jatkotoimenpiteitä agenteissa, ja jopa ulkoisissa sovelluksissa yhteisen kehyksen kautta. Arkkitehtuurin yleisrakenne on kuvattu kuvassa 3.



Kuva 3. Arkkitehtuuri koostuu monista agenteista, se kommunikoi Jaspis-kehysohjelmiston kautta, ja käyttää yhteistä Tietovarastoa.

Informaatiota voidaan hankkia julkisten informaatiopalveluiden, samaa kehysohjelmistoa käyttävien muiden sovellusten, tai erityisesti tietoisuutta tukevan sovelluksen rajoitettua tarkoitusta varten suunniteltujen sovellusten kautta. Raaka-informaation muokkaus ja jalostaminen ovat esityksen muodostamisesta erillisiä toimintoja. Tiedon esittämistä käsittelevät komponentit ovat palveluita, jotka voivat jakaa resurssejaan muille sovelluksille. Sama pätee myös tietoa kerääviin komponentteihin.

Tässä tutkimuksessa kuvattu arkkitehtuuri on enemmän äänimaisemaesitysten suunnitteluun, hallintaan ja koordinointiin tarkoitettu sovellus kuin konkreettisten äänten tuottamiseen tarkoitettu sovellus. Vaikka äänen käsittely voi tässä sovelluksessa olla hyvinkin yksityiskohtaista, lopullisen laitteistorajapinnan ja äänenmuodostuslaitteen toteutus ovat tämän tutkimuksen ulkopuolisia asioita. Seuraavaksi kuvaan arkkitehtuurin jokaisen pääkomponentin esimerkkien avulla. Ensiksi tarkastellaan jokaista agenttityyppiä ja niiden tehtäviä. Seuraavaksi kuvaan äänenmuodostuslaitteen ja sen rajapinnan. Ohjaus- ja kuvauskieli käsitellään viimeiseksi.

5.1. Äänimaiseman ohjausagentit

Äänimaiseman suunnittelu, luominen ja hallinta tapahtuu agenttien avulla. Sovelluksen ajon aikana agentit valitaan Jaspis-kehyksen agentti-evaluaattori-hallintaohjelma –periaatteen avulla. Tässä paradigmassa agenttien tehtävät ja velvollisuudet ovat pieniä. Monien agenttien toiminnallisuutta yhdistämällä voidaan rakentaa suuria sovelluksia ja silti saavuttaa helppo hallittavuus, muunneltavuus ja laajennettavuus. Agentteja, joilla on atomiset tehtävät, voivat käyttää muut agentit yhdistelmätehtävien suorittamiseen ja niin edelleen. Agentit voivat myös periä toiminnallisuutta normaalin olioperustaisen ohjelmointitavan malliin. Tämä hajautus on tärkeätä uudelleenkäytettävyyden ja laajennettavuuden kannalta. Informaation kulun vapaus saavutetaan blackboard-tyyppisen yhteisen Tietovaraston avulla. Tietovarastossa oleva informaatio on kaikkien agenttien luettavissa ja muokattavissa. Agentit voidaan jakaa karkeasti neljään ryhmään:

- tarkkailija-agentteihin, jotka välittävät informaatiota ulkopuolisista lähteistä, sensoreista ym.,
- logiikka-agentteihin, jotka määräävät kuinka havaitut tapahtuvat vaikuttavat esitykseen ja sen rakenteeseen käsitteellisellä tasolla,
- kompositioagentteihin, jotka suunnittelevat esityksen rakenteen ja ohjaavat soitinagentteja, sekä
- soitinagentteihin, jotka ohjaavat esityksen elementtejä ja tuottavat lopullisen konkreettisen kuvauksen esityksestä, joka lähetetään äänenmuodostuslaitteelle.

On mahdotonta määritellä selkeitä rajoja eri agenttieryhmien välille, ja kategorisointi onkin enemmän suunnittelua ja toteutusta ohjaava ja helpottava. Jokatapauksessa, tämä jaottelu on tärkeätä tehtävien rinnakkaistamisen, iteratiivisen kehityksen, ja sovellussuunnittelijoiden ja esityksen säveltäjien välisen kommunikoinnin kannalta. Kategorisointi myös vihjaa summittaisesti missä järjestyksessä asiat tulee käsitellä. Seuraavissa alikohdissa käsitellään jokainen agenttityyppi tarkemmin.

5.1.1. Tarkkailija-agentit

Tarkkailija-agentit keräävät informaatiota sovellusten tarpeisiin. Esimerkiksi sovelluksessa, joka esittää läsnäolotietoa, tarkkailija-agentit voivat kerätä informaatiota käyttäjien aktiivisuudesta ja sijainnista eri lähteistä (tietokoneen käyttö, paineherkät lattiamatot, puhujantunnistustulokset), tehtävistä joita

käyttäjät ovat tekemässä, ja kuinka paljon käyttäjiä sopii häiritä (ketkä puhuvat samassa huoneessa, kenellä on kalenterissaan merkittyjä tapahtumia).

Tarkkailija-agentit käyttävät joko yhteistä Tietovarastoa tai ulkoista tietokantaa informaation keruussa. Informaatio voi olla muodostettu ja tallennettu saman järjestelmän muissa tai ulkoisissakin sovelluksissa. Tarpeen vaatiessa, tarkkailija-agentit voivat muuntaa tätä informaatiota käsitteellisempään muotoon, jota muut agentit osaavat paremmin käsitellä. Esimerkiksi tarkkailija-agentit voivat siistiä turhia yksityiskohtia tunnistustuloksista, jotta vain logiikka-agenttien tarvitsemia osia annetaan niille prosessoitaviksi. Laajemman sovelluskokoelman ja kokonaisen kattavan jokapaikan tietotekniikan ympäristön kontekstissa on luonnollista hajauttaa informaation keruu ja esitys erillisiksi ja itsenäisiksi sovelluksikseen. Nämä sovellukset silti käyttäisivät yhteistä Tietovarastoa kommunikointiin.

5.1.2. Logiikka-agentit

Logiikka-agentit käyttävät tarkkailija-agenttien keräämää informaatiota, ja hyödyntävät sitä suunnitellessaan ja aloittaessaan toimia sovellusalueen käsitteellisellä tasolla. Suunnittelulogiikka ei liity sinällään suoraan musiikkiin tai ääniin, vaan asioihin kuten milloin esityksiä tulisi aloittaa, kenelle esitys pitäisi suunnata ja niin edelleen. Yksinkertaistettuna, nämä agentit toimivat tapahtumien laukaisijoina. Siinä missä tarkkailija-agentit käsittelevät informaation välitystä ja muuntamista, logiikka-agentit ovat enemmän vastuussa informaation yhdistämisestä, päätelmien tekemisestä ja sen pohjalta toiminnan aloittamisesta.

Esimerkiksi logiikka-agentit ymmärtävät, että tiedonjyvät kuten ”kello on 08:00”, ”päivämäärä on maaliskuun 13. 2005”, ”Jaakko on tunnistettu ovella”, ja ”Jaakon edellinen aktiivisuus tapahtui maaliskuun 12. 2005 kello 19:50” yhdessä tarkoittavat, että ”Jaakko on saapunut töihin” ja ”Jaakon töihin saapumista esittävä ääni pitäisi lisätä ääniesitykseen.”

5.1.3. Kompositioagentit

Kompositioagenttien tehtävä on luoda käsitteellisiä hahmotelmia esityksestä tai kompositiosta perustuen siihen mitä logiikka-agentit ovat päättäneet esitettävän. Kompositioagentit kuvaavat sovellusalueen kielellä kuvattua informaatiota ääniesityksen kielellä kuvattuun muotoon. Niiden tehtävänä on päättää mitä soittimia ja ääniraitoja käytetään, kuinka ne yhdistyvät toisiinsa sävellyksellisesti

ja mitä muita melodisia päätöksiä tehdään. Kompositioagentit rakentavat kompositioita tai sävellyksiä perustuen logiikka-agenttien tekemiin päätelmiin.

Esimerkiksi kompositioagentit yhdistelevät informaation palasia kuten ”Jaakko on saapunut töihin” ja ”Esa-Pekka on ollut töissä 40 minuuttia”, ja päättävät että äänimaisema koostuu ääniraidoista nimeltä ”Jaakko saapuu töihin, ”siirtymä Jaakon saapumisesta Esa-Pekan läsnäoloon”, ”Esa-Pekan läsnäolo”, ”Esa-Pekka käyttää tekstinkäsittelyä”, ja ”siirtymä Esa-Pekan tekstinkäsittelystä esityksen loppuun”.

Tällä tasolla esitys on edelleen käsitteellisessä muodossa, ja komposition elementtejä ajatellaan sovellusalueen terminologian mukaisin käsittein. Kompositioagenttien käyttämät ääniraidat ja -elementit koostuvat todennäköisimmin pienemmistä palasista, joiden monimutkaisuus ja toiminnallisuus on piilotettuna niiden toisaalla tehtyihin määrityksiin. Siirtymä-elementit ovat hyvä esimerkki tästä.

5.1.4. Soitinagentit

Soitinagenttien tehtävä on muuttaa kompositioagenttien käyttämät käsitteelliset ääniraidat, instrumentit ja efektit konkreettisiksi komennoiksi ja parametreiksi äänenmuodostuslaitteita varten. Soitinagentit käsittelevät asioita kuten pyyntöjä, tiedostonimiä, tarkkaa ajoittamista ja ehtolauseiden kuvauksia, efektejä kuten äänenvoimakkuutta ja jälkikaikua, sekä muita konkreettisia elementtejä ja komentoja. Edelleen, soitinagenttien ja kompositioagenttien tehtävät ovat hieman päällekkäisiä, sillä tiettyjä efektejä kuten äänenvoimakkuutta voidaan käyttää sekä laajemmalla sävellyksellisellä että yksittäisen instrumentin tasolla. Suurin ero sävellyksellisen ja instrumenttitason välillä on ääniesityksen käsitteellisen ja konkreettisen kuvaustavan ero.

Esimerkiksi soitinagentti ymmärtää, että ääniraita nimeltä ”Jaakko saapuu töihin” tarkoittaa, että tiedosto nimeltä ”lintu_13.wav” tulisi voimistaa kuuluviin viidessä sekunnissa, jonka jälkeen sama tiedosto pitäisi toistaa kolme kertaa 100%:n voimakkuudella.

Kun soitinagentit ovat tehneet työnsä, koko ääniesitys on kuvattu äänenmuodostuslaitteen ymmärtämässä muodossa, jolle se lähetetään Tietovaraston kautta esitettäväksi, kuten seuraavaksi alikohdassa 5.2 kuvataan. Äänenmuodostuslaite reagoi näihin esityspyyntöihin oman tilansa ja sisäisten sääntöjensä mukaisesti. Lisäkommunikaatio laitteen ja muun sovelluksen välillä voi tarkoittaa sitä, että esityksen kuvaus suunnitellaan uudelleen soveltumaan

tiettyihin tilanteisiin, kuten esimerkiksi synkronisointiin. Seuraavaksi kuvaan äänenmuodostuslaitteen, jolle agentit loivat ääniesityksen kuvauksen.

5.2. Äänenmuodostuslaitteet

Viimeinen vaihe ääniesityksen muodostusprosessissa on äänenmuodostuslaite-komponentti, joka tuottaa lopullisen äänidatavirran ja lähettää sen äänentuottolaitteistolle toistettavaksi. Äänenmuodostuslaitteet ovat itsenäisiä palveluita, ja mikä tahansa sovellus voi ottaa niihin yhteyden ja lähettää niille esityspyynnön. Jokaista fyysistä resurssia, kuten tietyssä huoneessa sijaitsevaa tiettyä kaiutinta, kohden on oma palvelunsa. Teknisesti tämä tarkoittaa palvelua jokaista järjestelmään kuuluvan tietokoneen äänikortin äänen ulostulokanavaa kohden.

Laitteistorajapinta ja matalan tason toiminnallisuus voidaan toteuttaa jollain kolmannen osapuolen ohjelmistolla, kuten esimerkiksi Java Audio Synthesis System:llä (JASS) [van den Doel & Pai, 2001] tai Microsoft Direct Sound:lla [Microsoft, 2005]. Tässä tutkimusessa määrittelemm vain ne rajapinnat, palvelut ja toiminnallisuuden, jota äänenmuodostuslaite tarjoaa sovelluksille.

Äänenmuodostuslaite on itsenäinen, jatkuvasti toiminnassa oleva palvelu, joka voi ottaa vastaan pyyntöjä ja komentoja joiden mukaan se toimii. Laite on yleinen, mikä tarkoittaa että sen ymmärtämät komennot ja pyynnöt eivät ole sidottuja minkään yksittäisen sovelluksen käsitteistöön, vaan äänen käsittelyyn ja esittämiseen yleisesti. Yleisiä komentoja voivat olla esimerkiksi pyyntö soittaa jokin ääniraita, tai hiljentää jonkin raidan äänenvoimakkuus tietylle tasolle joidenkin millisekuntien sisällä.

Äänenmuodostuslaite voi myös kommunikoida takaisin sovelluksille, jotka käyttävät sitä, eli siinä käytetään takaisinkutsu (callback) rajapintaa. Laite vastaa pyyntöihin riippuen niiden sisällöstä, esimerkiksi ilmoittamalla komennon vastaanottamisesta, tiedottamalla sovellusta mahdollisista soveliaista esityksen muuttamishetkistä, tai kertomalla koska on mahdollista synkronisoida äänisovelluksen ulkopuolisia tapahtumia kuten graafisia animaatioita. Näillä tavoin laite voi olla aktiivisempi sitä käyttävien sovellusten kanssa yhteistyössä toimiessaan. Kaikki takaisinkutsukommunikaatio perustuu muuttujiin, joiden avulla laite seuraa asioista joita se on esittänyt ja joita se tulee esittämään. Muuttuja voi olla esimerkiksi yksittäinen ääniraita koostuen muutamasta äänesta ja hiljennysefektistä, tai koko esityksen aloittamisesta kulunut aika millisekunteina.

Ulkopuolisten komentojen lisäksi äänenmuodostuslaite voi reagoida sisäisiin tapahtumiinsa. Laite näkee vastaukset, musiikilliset muutoshetket, synkroisointikohdat ja muut vastaavat tapahtumat laukaisimina (trigger) itse esityksen kuvauksessa, eli kompositiodatassa. Näiden laukaisimien avulla laite muuttaa käyttäytymistään tai vastaa takaisin sovelluksille. Laukaisimet on upotettuina osiksi kompositioita, komentoja tai pyyntöjä, jotka on lähetetty laitteelle. Ne sisältävät ehtolauseita, joiden toteutumista laite seuraa. Laukaisin koostuu kahdesta osasta: Laukaisimen ehto voi olla esimerkiksi se, että ääniraita on saapunut ennaltamäärättyyn kohtaansa, tai että sävellys soi D-mollissa. Laukaisin sisältää myös komennon, joka voisi olla esimerkiksi pyyntö raportoida takaisin sovellukselle tai muuttaa sovelluksen esitystä jollain tavalla. Laite ei välttämättä osaa itse tulkita kaikkea kaikkea sisältöä, jota sitä pyydetään raportoimaan takaisin, sillä tämä sisältö voi koskea jotain laajennettua toiminnallisuutta, jonka agentit ovat määritelleet, ja joka vain varastoidaan muuttujana laitteessa. Laukaisimet voivat käyttää komentoja ohjaamaan laitteen käyttäytymistä eri syistä, joista monia käsiteltiin kohdassa 4. Laukaisimien sisältämät ehdot eivät ole rajoitettu vain esitystä koskeviin muuttujiin, sillä laitteen sisäistä käyttäytymistä koskevat muuttujat voivat myös olla hyödyllisiä.

Sovellusten vuorovaikutteisuus, eli reagointi ulkomaailman tapahtumiin, käsitellään agenttien puolesta muissa osin sovellusta. Tämä käsiteltiin alikohdassa 5.1. Pyynnöt, komennot ja esityksen kuvaus ovat XML-pohjaisia. Ne sisältävät varsinaiset komentosanat ja tarvittavat lisäparametrit, kuten käytettävät ehdot ja muuttujat. Parametrit voivat sisältää kompositioiden yksityiskohtaisia kuvauksia ja vaadittavia laukaisimia. Äänenmuodostuslaitteen vastatessa takaisin muille sovelluskomponenteille, se käyttää samaa XML-notaatiota. Alikohdissa 4.2 ja 4.3 keskusteltiin siitä miksi agenttien (ja myös äänenmuodostuslaitteen) toiminnallisuus tulisi kuvata ohjauskielellä siten, ettei kaikki toiminnallisuus ole rajattu kovakoodattuihin ominaisuuksiin. Seuraavaksi tarkastelemme kuinka tämä on mahdollista saavuttaa käytännössä.

5.3. Ohjauskieli agenttien hallintaan

Agenttien sisäinen toiminnallisuus on osittain kuvattu ohjaus- tai kuvauskielellä. Tämä kuvaus ladataan konfiguraatiotiedostosta sovellusta käynnistettäessä. Samaa agenttia voidaan käyttää eri tarkoituksiin erilaisten konfiguraatioiden kautta.

On tärkeätä jättää agenteissa tilaa muunneltavuudelle ja vaihtelevuudelle, jotta niitä voidaan helposti muuttaa esitystä tai sovellusta suunniteltaessa ilman lähdekoodin kääntämistarvetta. Esityksen suunnittelija tai säveltäjä voi tarvita suuren kokoelman agentteja kuvaamaan kaiken sen toiminnallisuuden mitä hän toivoo sovellukselta. Ohjauskielen tarkoitus on luoda yhteinen työkalu säveltäjien ja sovellussuunnittelijoiden yhteistyötä varten. Ohjauskielen tulisi olla tarpeeksi helppokäyttöinen, jotta ne joilla ei ole paljoa ohjelmointikokemusta, voivat käyttää sitä kuvaamaan kompositioitaan interaktiivisella tavalla. Lisäksi sovelluksen toteuttajat voivat ymmärtää sävellykselliset elementit niin yksiselitteisesti, että ne voidaan toteuttaa.

Koska ääniesityksen kuvaukset ovat XML-muotoisia, tapa jolla agentti tulkitsee Tietovarastosta löytyneen kuvauksen riippuu sekä kovakoodatusta että konfiguroidusta toiminnallisuudesta. Jokaisen ohjauskielen pätkän lopullinen merkitys on aina kuvattu sekä agenteissa, että äänenmuodostuslaitteessa. Koska ohjauskielellä tehty kuvaus on laajennettavissa, agentit käsittelevät ne osat komposition kuvausta, jotka ne itse osaavat tulkita. Tämä tekee agenteista myös uudelleenkäytettäviä. Aivan kuten agentitkin, ääniesityksen kuvaus on myös rakennettu olioperustaiseen tapaan.

Soitinagentit poislukien, tarkoituksena on kuvata asioita (ääniä, efektejä, muuttujia) käsitteellisin ja symbolisin termein, jotka tulevat sovellusalueen terminologiasta (esimerkiksi ”Esa-Pekka saapuu töihin”), eikä teknisin termein (tiedostonimet jne.), ja jotka kätkevät kompleksisuuden. Tämä on tärkeätä, jotta sovelluksen suunnittelijat ja esityksen säveltäjät voivat työskennellä yhdessä. Tämä on mahdollista tarjoamalla joukko agentteja, jotka toteuttavat perustoiminnallisuuden, ja joita voidaan käyttää eri tarkoituksiin ohjauskielen avulla. Eri tavoin konfiguroituja agentteja voidaan yhdistellä ja ketjuttaa monimutkaisemman toiminnallisuuden saavuttamiseksi. Tämä ei sulje pois uusien agenttien ohjelmointimahdollisuutta. Riippuu täysin sovellusalueesta ja ääniesityksen sävellyksellisistä pyrkimyksistä minkälaisia agentteja tarvitaan. Esimerkiksi monimutkaista signaalinkäsittelyä ei tarvita kaikissa sovelluksissa.

Vaikka suuri määrä toiminnallisuutta voidaan saavuttaa yksinkertaisen konfiguroinnin kautta, tarvitaan myös enemmän tai vähemmän täysikykyistä ohjelmointisyntaksia boolean logiikkoineen ja kontrollirakenteineen. Tämä on jaettua harmaata aluetta agenttien kyvykkyyden ja ohjauskielen kyvykkyyden välillä. On olemassa monia paljonkäytettyjä kuvaus- ja ohjauskieliä, jotka täyttäisivät äänimaiseman kuvaustarpeet, joko musiikillisesta tai toiminnallisesta

näkökulmasta [Wang & Cook, 2003; Grigg, 2003; Haus & Longari, 2002; Taylor & Isard, 1997].

6. Yhteenveto

Tässä tutkimuksessa käsiteltiin tietoisuuden ja yleisemmin sosiaalisten suhteiden tärkeyttä ryhmätyölle. Esitin kuinka jokapaikan tietotekniikkaa voidaan käyttää tukemaan tietoisuutta työtehtävissä ja sosiaalisissa tilanteissa laaja-alaisesti. Tutkimuksessa käsiteltiin myös tietämisen ja tietoiseksi tulemisen prosesseja, ja kuinka tietoisuutta tukevaa informaatiota tulisi esittää rauhallisilla ja häiritsemättömillä tavoilla. Tämä toimi motivaationa äänen avulla tietoisuutta tukevan sovelluksen rakentamiseen toimistoympäristöön. Aikaisempi tutkimus ja iteratiivisen kehityksen aikana kohdatut haasteet johtivat monien sellaisia sovelluksia koskevien vaatimusten keräämiseen.

Tietoisuuden tukeminen jokapaikan tietotekniikan kontekstissa vaatii sovelluksia, jotka on rakennettu arkkitehtuurille, joka osaa hallita suuria datamääriä laaja-alaisista aihepiireistä ja sovelluskohteista. Ympäristön muuttuvuuden vuoksi arkkitehtuurin tulisi sisältää avoin rajapinta ulkoisille tietolähteille, sen tulisi olla helposti laajennettavissa eri ympäristöihin ja resursseille, ja sen tulisi sopeutua muuttuviin tilanteisiin. Arkkitehtuurissa tulisi olla helppoja tapoja kuvata informaatiota esityselementteihin, rakentaa kompositioita näistä elementeistä ja määritellä sääntöjä niiden käyttäytymiselle.

Rauhallisuuden ja häiritsemättömyyden vuoksi esityksen täytyy pysyä yhtenäisenä ja jatkuvana. Yhtenäisyyden ja jatkuvuuden saavuttamiseksi tarvitaan menetelmiä tiedon kuvauksen, siirtymien, ajoittamisen ja äänien yhdistämisen hallintaan. Esityksen häiritsemättömyyden saavuttamiseksi tarvitaan herkkyyttä erilaisille sosiaalisille ja työtehtävätilanteille. Sopeutuvuuden tulisi olla reaaliaikaista, mikä puolestaan vaatii itsenäisiä ja rinnakkain toimivia ohjelmistokomponentteja.

Arkkitehtuurin tulisi sallia sovellusten iteratiivinen suunnittelu, ja tarjota työkalu jonka kautta sovellussuunnittelijat ja äänimaiseman suunnittelijat voivat keskustella yhdessä. Tämä myös mahdollistaisi monimutkaisuuden hallinnan abstrahoimalla ja hajauttamalla tehtävät ja elementtien määritelmät. Toiminnallisuuden tulisi olla konfiguroitavissa, laajennettavissa ja komponenttien tulisi olla uudelleenkäytettäviä.

Esittelin näihin vaatimuksiin perustuvan arkkitehtuurin. Arkkitehtuuri on suunniteltu äänimaisemien avulla tietoisuutta tukevien sovellusten toteuttamiseen. Arkkitehtuuri koostuu agenteista, Tietovarastosta ja äänenmuodostuslaitteista. Agentit ovat pieniä, mutta runsaslukuisia, ja ne hajauttavat tiedonkeruun, loogisen päättelyn, komposition ja soittimien hallinnan tehtävät keskenään. Agentit ovat yleisluontoisia, uudelleenkäytettäviä ja laajennettavia, koska arkkitehtuuri on suunniteltu vahvasti olioperustaiseen ajatteluun perustuen. Agenttien toiminnallisuus on konfiguroitavissa ohjaus- ja kuvauskielen avulla. Komponentit käyttävät ohjauskieltä kommunikointiin, mikä on myös suunniteltu tukemaan hajautusta, uudelleenkäytettävyyttä ja laajennettavuutta. Monimutkaisuus hallitaan hierarkkisen abstrahoinnin ja hajauttamisen avulla. Ääniesitysarkkitehtuurin ja ulkoisten sovellusten välinen vahva yhteys tarjoutuu Jaspis-kehysohjelmiston kautta, ja kommunikaatio on helppoa yhteisen Tietovaraston kautta.

Jatkossa arkkitehtuuria on suunniteltu kehitettävän eteenpäin mm. integroimalla se useampien sovellusten ja tietolähteiden kanssa. Muutamia ohjaus- ja kuvauskieliä on myös vertailtu esityksen rakenteen ja toiminnallisuuden kuvaamisen näkökulmasta, ja suunnitelmissa on tarkentaa tässä tutkimuksessa alustettua ohjauskielen kuvausta ottamaan huomioon niiden monissa konteksteissa kokeiltuja ja testattuja parhaita ominaisuuksia.

Viiteluettelo

[Kainulainen et al., 2005] Kainulainen, A., Turunen, M., Hakulinen, J., Salonen, E.-P., Prusi, P., and Helin, L., *A Speech-based and Auditory Ubiquitous Office Environment.* Proceedings of the 10th International Conference on Speech and Computer (SPECOM 2005), 2005.

[Mäkelä et al., 2003] Mäkelä, K., Hakulinen, J., and Turunen, M., *The Use Of Walking Sounds In Supporting Awareness.* In Proceedings of ICAD 2003, 2003, 144-147.

[Mäkelä et al., 2001] Mäkelä, K., Salonen, E.-P., Turunen, M., Hakulinen, J., and Raisamo, R., *Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman.* In Proceedings of the In-ternational Workshop on Information Presentation and Natural Multimodal Dialogue, 2001, 115 – 119.

[Turunen et al., 2005] Turunen, M., Hakulinen, J., Räihä, K.-J., Salonen, E.-P., Kainulainen, A., and Prusi, P., *An architecture and applications for speech-based accessibility systems.* IBM Systems Journal, Vol. 44, No 3, 2005, 485-504.

[Dourish & Bellotti, 1992] Dourish, P., and Bellotti, V., *Awareness and Coordination in Shared Workspaces.* In Proceed-ings of the ACM CSCW Conference on Computer-Supported Cooperative Work, 1992, 107-114.

[Matthews et al., 2003] Matthews, T., Rattenbury, T., Carter, S., Dey, A., and Mankoff, J., *A Peripheral Display Tool-kit.* Intel Research Berkeley, Technical Report IRB-TR-03-018, 2003.

[Temperley, 1997] Temperley, D., *An Algorithm for Harmonic Analysis.* In: Music Perception, 15/1, 1997, 31-68.

[Mugglin, 2005] Mugglin, S.: *Music Theory for Songwriters.* Webpage: http://members.aol.com/chordmaps/ [3.6.2005], 2005.

[Eno, 1996] Eno, B.: *Generative Music.* A talk delivered at the Imagination Conference, San Francisco, June 8th, 1996. Available in: http://www.inmotionmagazine.com/eno1.html [3.6.2005]

[Land & McConnell, 1994] Land, M. Z., and McConnell, P. N., *Method and Apparatus for Dynamically Composing Music and Sound Effects Using a Computer Entertainment System.* U.S. Patent 5,315,057, May 24th, 1994.

[Microsoft, 1998] Microsoft Corporation, *Composing Music for Interactive Titles: An Overview of Direct-Music Producer.* 1998. Webpage: http://msdn.microsoft.com/library/deafult.asp?url=/library/en-us/dnmusic/htm/interact.asp [3.6.2005]

[Microsoft, 2005] Microsoft Corporation: *Microsoft Direct Sound.* 2005. Webpage: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/htm/directsound.aps [3.6.2005]

[Wang & Cook, 2003] Wang, G., and Cook, P. R., *ChucK: A Concurrent, On-the-fly, Audio Programming Language.* In Proceedings of the 2003 International Computer Music Conference, 2003.

[Grigg, 2003] Grigg, C., *Preview: Interactive XMF – A Standardized Interchange File Format for Ad-vanced Interactive Audio Content.* In 115th Audio Engineering Society Convention, Octo-ber 10-13, 2003.

[van den Doel & Pai, 2001] van den Doel, K., and Pai, D., *JASS: An Audio Synthesis System for Programmers.* In Proceedings of the 2001 International Conference on Auditory Display, Espoo, Finland, July 28-August 1, 2001.

[Haus & Longari, 2002] Haus, G., and Longari, M., *Towards a Symbolic/Time-Based Music Language Based on XML.* In Proceedings of MAX2002, The First International Conference on Musical Applications using XML, 2002, 38-46.

[Wisneski et al., 1998] Wisneski, C., Ishii, H., Dahley, A., Gorbett, M., Brave, S., Ullmer, B., and Yarin, P., *Ambient Displays: Turning Architectural Space into an Interface between People and Digital Information.* In Proceedings of CoBuild'98 Proceedings of CoBuild'98, 1998, 22–32.

[Taylor & Isard, 1997] Taylor, P., and Isard, A., *SSML: A speech synthesis markup language.* Speech Communication 21, 1997, 123-133.

Standardit multimodaalisissa käyttöliittymissä

Tanja Malmberg

Tiivistelmä.
Multimodaaliset käyttöliittymät mahdollistavat useamman aistin käytön ihmisen ja tietokoneen välisessä vuorovaikutuksessa, jolloin kommunikoinnin tulisi olla helpompaa. Samalla yhä useampien on mahdollista käyttää tietotekniikkaa hyväkseen erilaisissa tilanteissa. Multimodaalisia sovelluksia varten onkin suunniteltu ohjelmistoarkkitehtuureja, jotka pyrkivät huomioimaan alueen monimuotoisuuden. Sovellusten kehitystyössä auttavat myös useat standardit ja suositukset, joita on määritelty varsinkin puhekäyttöliittymiä varten. Niiden avulla voidaan sekä tehostaa työskentelyä että vähentää virheiden määrää.

Avainsanat ja -sanonnat: Multimodaalisuus, käyttöliittymät, ohjelmistoarkkitehtuurit, standardit.

1. Johdanto

Perinteisesti ihmisen ja tietokoneen välinen vuorovaikutus on ollut lähinnä käskyjen antamista näppäimistön ja hiiren avulla sekä palautteen saamista visuaalisesti näytön kautta. Tämä on hankaloittanut tai jopa estänyt tiettyjen erityisryhmien tietokoneen käytön ja toisaalta osoittautunut epäkäytännölliseksi monien nykypäivän tietokonesovellusten, kuten kämmenmikrojen, kanssa. Tällaisten vuorovaikutusongelmien ratkaisemiseksi on kehitetty useampaa aistia hyödyntäviä, multimodaalisia, käyttöliittymiä.

Multimodaaliset käyttöliittymät mahdollistavat vuorovaikutuksen esimerkiksi puheen, kirjoittamisen, eleiden ja ilmeiden avulla. Nykyisin multimodaalisuutta hyödynnetään muun muassa matkapuhelimissa ja muissa pienikokoisissa laitteissa, WWW-selaimissa ja autoissa.

Multimodaalisia sovelluksia varten on esitetty oma ohjelmistoarkkitehtuuri, joka toimii sovelluksen suunnittelun ja toteutuksen pohjana. Arkkitehtuuri erottaa toisistaan suunnitteluvaiheen ja ajonaikaisen näkymän, joista ensimmäinen esittää sovelluksen merkintäkielten ja jälkimmäinen ohjelmistokehyksen sekä eri komponenttien tasolla.

Jotta eri valmistajien multimodaaliset sovellukset voisivat toimia keskenään, tarvitaan standardeja luomaan niille yhteinen pohja. Standardien avulla voidaan sekä tehostaa työskentelyä yhteisten käsitteiden ja määritelmien kautta että vähentää virheitä ja väärinkäsityksiä.

Tässä tutkielmassa tutustutaan ensin tarkemmin multimodaalisuuteen yleensä ja sen hyödyntämiseen käyttöliittymissä. Tämän jälkeen kuvataan World Wide Web Consortiumin esittämä multimodaalinen arkkitehtuuri. Lopuksi luodaan ensin yleiskatsaus standardeihin ja tämän jälkeen käydään tarkemmin läpi muutamia multimodaalisten käyttöliittymien standardeita ja suosituksia.

2. Multimodaaliset käyttöliittymät

Multimodaalisuus *(multimodality)* on käsitteenä melko uusi ja alan tutkimus tällä hetkellä vilkasta. Yhä useammat uudet laitteet käyttävät hyväkseen multimodaalisuuden tarjoamia mahdollisuuksia, vaikka toisaalta perinteinen näyttö-näppäimistö-yhdistelmä näyttää vieläkin olevan suosituin vuorovaikutuksen muoto ihmisen ja tietokoneen välillä.

Mitä multimodaalisuudella sitten tarkoitetaan? Ihmisen ja tietokoneen välisessä vuorovaikutuksessa sanalla viitataan mahdollisuuteen käyttää useampaa yhtäaikaista kanavaa, joilla kommunikointi osapuolten välillä on mahdollista [Kanninen, 2003]. Näitä kanavia ovat kontekstista ja käyttäjästä riippuen esimerkiksi näkö, kuulo ja tunto.

Itse asiassa jo sanan 'multimodaalisuus' alkuosa kertoo, että käytettävissä on useampi kuin yksi modaliteetti samanaikaisesti. Modaliteetti on määritelty määrittelijästä ja aihealueesta riippuen eri tavoin, mutta puhuttaessa ihmisen ja tietokoneen vuorovaikutuksesta sillä tarkoitetaan eri aistien kautta tapahtuvaa viestintää [Adage, 2004]. Keskeisinä voidaan pitää visuaalista, auditiivisista, haptista (kosketukseen liittyvää) ja tasapainon modaliteettia.

Ihmisten keskinäinen vuorovaikutus elävässä elämässäkin on multimodaalista. Kommunikointi muiden kanssa tapahtuu niin puheen kuin eleiden, jäljittelyn ja äännähdysten avulla. Eri modaliteetteja ei käytetä itsenäisesti toisistaan riippumattomina, vaan yhden modaliteetin avulla viestitetyn tiedon merkitys on riippuvainen myös enemmän tai vähemmän yhtäaikaisesti käytetystä toisesta modaliteetista. Esimerkiksi auditiivista modaliteettia tukee usein visuaalinen, kun toisen ihmisen puheen kuuntelemisen

lisäksi tarkkailemme myös hänen sanatonta viestintäänsä eli esimerkiksi eleitä [Bunt, 1998].

Bunt [1998] toteaa ihmisten käyttävän luonnollisessa kommunikoinnissaan niin sanottua maksimointiperiaatetta *(multimax principle)*: viestintätilanteessa molemmat osapuolet käyttävät kaikkia modaliteetteja ja kanavia, jotka ovat sillä hetkellä mahdollisia. Tämä lisää sekä viestinnän tehokkuutta että mahdollisuutta vaikuttaa maksimaalisesti toiseen osapuoleen. Lopullinen viestinnän onnistuminen on kuitenkin kiinni siitä pystyvätkö osapuolet käsittelemään kaikkea eri modaliteettien tarjoamaa tietoa.

Kun on kysymys ihmisen ja tietokoneen välisestä kommunikoinnista, Bunt [1998] erottaa karkeasti toisistaan kaksi erilaista vuorovaikutustapaa. Ensinnäkin käyttäjä muotoilee ohjeita tai kyselyitä, jotka hän ilmaisee tietokoneelle joko kirjoittamalla komentoja tai valikosta valitsemalla. Toisaalta käyttäjä säätää esimerkiksi ikkunan tai tekstin kokoa sekä näytön kirkkautta. Näitä ei yleensä voi tehdä komentoja kirjoittamalla, vaan nappuloiden ja hiiren avulla, jolloin niiden viestinnällinen luonne häviää ja tietokonetta käsitellään kuten mitä tahansa kodinkonetta.

Tällaiseen vuorovaikutukseen ei maksimoinnin periaate enää päde. Tietokoneen ja ihmisen välisestä vuorovaikutuksesta monet kahden ihmisen välisen viestinnän modaliteetit puuttuvat tai ne ovat mukana vain hyvin yksinkertaisessa muodossa [Bunt, 1998]. Esimerkiksi keskusteleminen tavanomaiseen tapaan ei ole mahdollista, vaikka nykyiset puhesovellukset pystyvätkin melko hyvin tunnistamaan erilaisia sanoja ja sanontoja.

Multimodaaliset käyttöliittymät ovat yritys parantaa ihmisen ja tietokoneen välistä vuorovaikutusta. Uusien vuorovaikutuskanavien lisäämisellä pyritään luomaan tietokoneesta yhä ihmismäisempi, jotta kommunikaatio koneen ja ihmisen välillä muuttuisi yhä enemmän luonnollisen, ihmisten välisen, viestinnän suuntaan [Kanninen, 2003]. Tällöin myös tietokoneen käytöstä tulisi entistä helpompaa edellyttäen tietenkin, että käyttöliittymä olisi riittävän helppo oppia ja käyttää.

Kun tietokoneen käyttö aiemmin on rajoittunut lähinnä paikallaan oleviin pöytätietokoneisiin, on nykyisin tietokonesovelluksia monissa muissakin laitteissa kuten matkapuhelimissa ja autoissa. Näitä sovelluksia käytetään usein liikkuessa, jolloin käsien käyttö ei välttämättä ole mahdollista ja yleensä pienen kokonsa vuoksi tiedonsyöttömahdollisuudet ovatkin rajoittuneet [Kanninen, 2003]. Vuorovaikutus on toteutettava perinteisten modaliteettien sijaan jonkin

muun, esimerkiksi auditiivisen modaliteetin avulla, joka soveltuukin hyvin nopeaan, yksinkertaisten ja lyhyiden viestien lähettämiseen käyttäjän ollessa liikkeellä [Paternò, 2004].

Koska kielen käyttö on keskeistä ihmisten välisessä viestinnässä, puheella on usein merkittävä osa, kun tietoa välitetään multimodaalisissa käyttöliittymissä käyttäjältä tietokoneelle ja toisinpäin. Koska täysin luonnolliseen keskusteluun ei tietokoneen kanssa vielä pystytä, toteaakin Sà [2001] vuorovaikutuksen koostuvan ihmisen osalta lähinnä keinotekoisista erityiskäskyistä tai hyvin rajoitetusta luonnollisesta puheesta ja tietokoneen osalta synteettisestä puheesta, josta puuttuvat kaikki normaalille puheelle tyypilliset variaatiot (esimerkiksi äänenpainot).

Puheen lisäksi multimodaalisissa käyttöliittymissä myös kasvojen ilmeillä ja osoituseleillä on tärkeä osa. Videoinnin avulla on mahdollista välittää tieto käyttäjän toimista tietokoneelle, jolloin se vastaa niihin määrätyllä tavalla. Aiemmin mainittujen ilmeiden ja eleiden lisäksi tällä tavoin voidaan yrittää tulkita myös käyttäjän tunteita, seurata silmänliikkeitä tai lukea puhetta huulilta [Sà, 2001]. Vastavuoroisesti tietokone voi simuloida sanatonta viestintää esimerkiksi niin sanotun keinopään avulla, joka on ihmisen päätä ja kasvoja jäljittelevä sovellus.

3. Multimodaalinen arkkitehtuuri

Ohjelmistoarkkitehtuuria voidaan pitää toteutettavan järjestelmän perustuslakina, joka määrittelee sen ytimen. Tämän ydin pysyy samana läpi kehityksen ja ylläpidon. Osat, jotka eivät kuulu ytimeen, ovat vapaasti muunneltavissa. Arkkitehtuuri ottaa myös kantaa keskeisiin järjestelmäratkaisuihin ja sen avulla pyritään parantamaan sekä ohjelmiston laatua että ohjelmistokehityksen tuottavuutta [Koskimies ja Mikkonen, 2005].

World Wide Web Consortium (W3C) on esittänyt arkkitehtuurirakenteen multimodaalisia käyttöliittymiä varten. Arkkitehtuurin päämääränä voidaan pitää sisäisten komponenttien toteutuksen kapselointia, ohjelmiston osien hajautusta, laajennettavuutta ja rekursiivisuutta sekä modulaarisuutta eli ohjelmiston jakamista eri yksiköihin [Barnett, 2005]. Pohjana W3C:n kuvaamalle arkkitehtuurille on ohjelmistoteollisuudessa paljon käytetyt viestinvälitys- *(message dispatcher)* ja malli-näkymä-ohjain *(model-view-controller*) -arkkitehtuurit.

Seuraavissa luvuissa käydään läpi W3C:n multimodaalista arkkitehtuurimallia, joka on esitelty huhtikuussa 2005 julkaistussa teknisessä dokumentissa [Barnett, 2005].

3.1. Suunnitteluvaiheen näkymä

Käsiteltäessä multimodaalisia käyttöliittymiä on tärkeää huomata ero suunnitteluvaiheen *(design-time)* ja ajonaikaisen *(run-time)* näkymän välillä. Näkymällä tarkoitetaan järjestelmästä riippuvaa kuvausta, joka on jonkin näkökulman mukainen [Koskimies ja Mikkonen, 2005].

Suunnittelutasolla multimodaaliset sovellukset ovat yleensä muodoltaan dokumentteja, jotka sisältävät erilaisia merkintätapoja. Usein eri merkintätavat vastaavat eri modaliteetteja, vaikka tämä ei olekaan pakollista. Merkintätasolla arkkitehtuurissa on ylemmän tason kieli, joka on dokumentin juuri. Muut kielet on sulautettu siihen osana rekursiivista prosessia, jossa mikä tahansa kieli voi toimia ylemmän tason kielenä itseensä sisällytettyyn toiseen kieleen nähden. Näillä kaikilla on oma nimiavaruutensa.

Merkintäkielet ovat siis näin yhtä aikaa olemassa kuitenkaan vaikuttamatta suoraan toisiinsa. Niiden välillä ei ole yhteisiä muuttujia, jotka voisivat välittää tietoja eri kielten kesken. Suunnitteluvaiheen merkintäkielinä voidaan käyttää jo olemassa olevia kieliä kuten XHTML:ää, joka sopii sekä ylemmän että alemman tason kieleksi.

3.2. Ajonaikainen näkymä

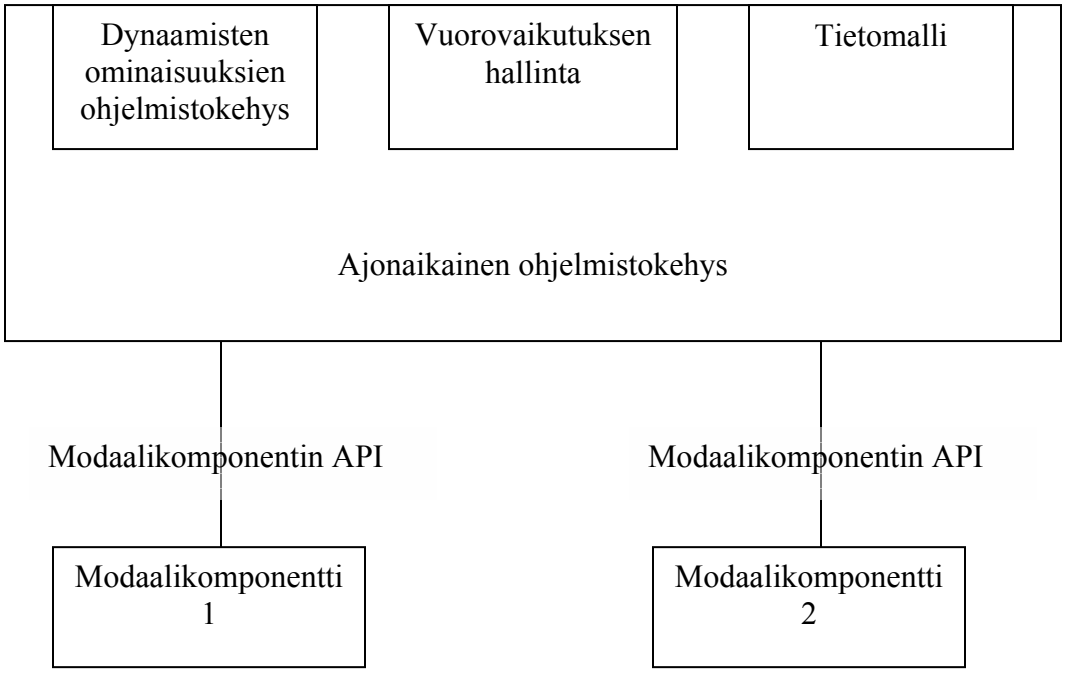
Multimodaalisten käyttöliittymien ajonaikainen arkkitehtuuri perustuu ohjelmistokehykseen *(framework)* ja komponentteihin *(components)*, jotka vastaavat esimerkiksi eri modaliteettien tuottaman tiedon välityksestä. Koska komponenttien toteutus on kapseloitu, voivat ohjelmistokehys ja osa komponentteja toimia alemman tason komponenttina ylemmän tason ohjelmistokehykselle, kuten merkintäkielet suunnitteluvaiheessa.

Komponentit voidaan jakaa kahteen osaan: modaalikomponentteihin, jotka toteuttavat vuorovaikutuksen käyttäjän kanssa ja palvelukomponentteihin, jotka sisältävät esimerkiksi tiedot käytetystä laitteesta ja käyttäjän mieltymyksistä. Eräs erittäin tärkeä palvelukomponentti on vuorovaikutuksenhallinta *(interaction manager)*, joka vastaa eri modaalikomponenttien vuorovaikutuksesta.

Tässä multimodaalinen arkkitehtuuri seuraa julkaisija/tilaaja-tapahtumamallia *(publish/subscribe event model)* eli komponenttien tuottamat

tapahtumat välitetään ohjelmistokehyksen kautta vuorovaikutuksenhallinnalle, joka vuorostaan päättää mitä niille tehdään. Jos vuorovaikutuksenhallinta puuttuu, ohjelmistokehys toimittaa tapahtuman kaikille komponenteille, joille se on merkitty. Samoin toimii vuorovaikutuksenhallinta tuottaessaan itse tapahtuman.

3.3. Arkkitehtuurikaavio



Kuva 1. Multimodaalisen ohjelmistoarkkitehtuurin rakenne [mukaillen Barnett, 2005].

Ajonaikainen ohjelmistokehys on vastuussa monista perustoiminnallisuuteen kuuluvista asioista, kuten sovelluksen käynnistämisestä, ohjelmistokomponenttien alustamisesta, nimiavaruuksien hallinnasta ja perustapahtumasilmukan ylläpitämisestä. Se myös tulkitsee ylimmän tason merkintäkielen.

Vuorovaikutuksenhallinta on valinnainen komponentti, joka siis vastaa kaikista muiden komponenttien tuottamista tapahtumista. Yleensä siihen liittyy erityinen merkintätapa, joka kertoo miten tapahtumiin tulee vastata. Jos vuorovaikutuksenhallinta ei sisällä selkeää käsittelijää tietylle tapahtumalle, se joko vastaa oletusarvon mukaisesti tai tämän puuttuessa sivuuttaa tapahtuman. Muilla komponenteilla voi olla omat vuorovaikutuksenhallintansa, jotka eivät kuitenkaan näy ylimmälle tasolle.

Dynaamisten ominaisuuksien ohjelmistokehys *(dynamic properties framework)* mahdollistaa nopean mukautumisen käyttäjän mieltymyksiin, ympäristön olosuhteisiin ja laitteen ominaisuuksiin. Tietomalli *(data model)* on valinnainen ohjelmistokomponentti, joka hallinnoi muiden kanssa jaettavaa tietoa ja on vastuussa muun muassa tiedon varastoinnista. Modaalikomponentit vuorostaan hoitavat kaiken vuorovaikutuksen käyttäjän kanssa ja ovat hyvin vahvasti sidoksissa sovellukseen, jossa niitä käytetään.

Ajonaikaisen ohjelmistokehyksen ja modaalikomponenttien välissä toimiva rajapinta vastaa asynkronisten DOM 3 tapahtumien välittämisestä. Rajapinnan kautta modaalikomponentit kommunikoivat sekä ohjelmistokehyksen että toisten komponenttien kanssa. Komponenttien odotetaan aloittavan tapahtumia automaattisesti (osana niiden toteutusta) tai merkintäkielen kautta. Näiden hyvin sovellusriippuvaisten tapahtumien välittämisestä huolehtii ohjelmistokehys ja siinä erityisesti vuorovaikutuksenhallinta.

W3C:n arkkitehtuurirakenne on yksinkertaistettu näkemys mahdollisesta multimodaalisten sovellusten ohjelmistoarkkitehtuurista. Arkkitehtuuri kuvataan hyvin yleisellä tasolla ja siihen jää vielä paljon tulkinnanvaraisia kohtia. W3C:n arkkitehtuuria voidaankin pitää enemmän hahmotelmana kannustamaan multimodaalisten sovellusten suunnittelua tietyn mallin mukaisesti kuin käytännössä testattuna arkkitehtuuriratkaisuna. On kuitenkin huomattava, että multimodaalinen arkkitehtuuri on tällä hetkellä vielä alkutekijöissään ja että sitä tullaan tulevaisuudessa edelleen kehittämään.

4. Standardit

Standardien ja suositusten avulla on mahdollista luoda yhteinen pohja ohjelmistoteollisuuden tuottamille erilaisille sovelluksille. Aktiivinen standardien kehittäjä ja hyväksyjä on W3C, jonka työryhmistä kaksi (ääniselainryhmä ja multimodaalinen vuorovaikutus-ryhmä) työskentelevät aktiivisesti standardien kehittämiseksi multimodaalisille käyttöliittymille.

Standardi voidaan Korpelan [2005] mukaan laajasti otettuna ymmärtää jonkin organisaation esittämäksi suositukseksi, joka kertoo, miten jokin asia tulisi tehdä. Yleensä kyseessä on tällöin jokin laaja kokonaisuus. Tiukasti määriteltynä standardi on puolueettoman, yleensä kansainvälisen standardointijärjestön vahvistama normi. Tiukempaa määritelmää käytettäessä on mahdollista puhua erikseen myös suosituksista, joista ajan mittaan saattaa kehittyä standardeja.

Keskeistä standardin vaikutuksen kannalta on miten laajasti se hyväksytään yhteisössä käyttöön. Standardista, jota kukaan ei käytä tai jonka vain hyvin harvat ovat omaksuneet, ei ole juurikaan hyötyä. Yleiseen käyttöön tulleella standardilla voidaan sen sijaan saavuttaa monia etuisuuksia, kun yhteiset käsitteet ja määritelmät tehostavat työskentelyä ja vähentävät virheitä sekä väärinkäsityksiä [SFS, 2005].

Standardien kautta voidaan varmistaa, että tuotteet ja menetelmät soveltuvat sellaiseen käyttöön ja olosuhteisiin, joihin ne on tarkoitettukin. Samalla voidaan vähentää kaupallisesti ja teknisesti merkityksettömiä erilaisuuksia eri tuotteiden välillä ja luoda sekä tuotteita että järjestelmiä, jotka ovat yhteensopiva [SFS, 2005]. Standardien ansiosta käyttäjillä on myös suurempi valinnanvapaus eri tuotteiden kesken. Ne myös edistävät teknologioiden uudelleenkäyttöä ja siirrettävyyttä [Larson, 2005].

Toisaalta Larson [2005] kutsuu standardeja myös kaksiteräiseksi miekaksi. Vaikka niiden hyödyt ovat kiistattomat, saattaa standardien käyttö joskus tukahduttaa suunnittelijan luovuutta. Koska standardit ovat vaikeasti laajennettavissa, saattaa sovelluksen joustavuus kärsiä, jos standardin toiminnallisuus ei tuekaan suunnittelijan tarpeita.

Seuraavaksi perehdytään lyhyesti multimodaalista käyttöliittymistä annettuihin standardeihin ja yleisiin suosituksiin.

4.1. SALT

Speech Application Language Tags eli SALT on puhekäyttöliittymiä varten luotu merkintäkieli, joka koostuu muutamasta XML-elementistä. Nämä elementit sijoitetaan johonkin toiseen kieleen, kuten HTML:ään tai XHTML:ään, koska SALT ei itsessään sisällä mitään kontrollirakenteita. Isäntäkieli määrittelee sovelluksen toiminnot ja vastaa sen suorittamisesta, SALT vuorostaan puheentunnistamisesta ja synteettisestä puheesta [Larson, 2005].

SALT on käyttökelpoinen sekä pelkästään puheeseen pohjautuvissa että multimodaalisissa käyttöliittymissä. Kielen kehittämisestä vastaa SALT Forum, jossa osallisena on monia tunnettuja tietotekniikkayrityksiä. SALT ei kuitenkaan ole vielä virallinen standardi, vaan avoin spesifikaatio, joka on toimitettu W3C:n käsiteltäväksi.

SALT Forumin [2005] määritelmän mukaisesti SALT rakentuu kolmesta pääelementistä, joita ovat <listen>, <prompt> ja <dtmf>. Muita elementtejä ovat

<grammar>, <bind> ja <record>. Näistä kaikki kolme voivat sisältyä <listen> elementtiin ja kaksi ensimmäistä myös <prompt> elementtiin.

Elementeistä <listen> toimii puheen vastaanottajana eli sen avulla yritetään tunnistaa puhutut sanat ja lauseet. Elementti määrittää myös tavan, jolla tunnistettua puhetta käsitellään ja on käytössä silloinkin, kun halutaan nauhoittaa puhetta. Näissä tehtävissä mukana ovat kaikki aiemmin mainitut alaelementit.

Käyttäjän mahdolliset sanat ja lauseet eli kielioppi on määritelty <grammar> elementissä, joita voi olla useitakin yhden <listen> lauseen sisällä. Vaikka SALT ei määritäkään mitään tiettyä kielioppia, johon <grammar> lauseessa tulisi viitata, on suositeltua tukeutua SRGS-standardin XML-malliin (katso 4.6.). Elementtiä <bind> voidaan käyttää tunnistetun puheen tutkimiseen ja sen merkittävien osien välittämiseen muille elementeille. Puheen ja esimerkiksi myös musiikin sekä muiden äänien nauhoittamisessa käytetään <record> elementtiä.

Vuorovaikutus ulospäin tapahtuu <prompt> elementin kautta eli se siis vastaa esimerkiksi ääninauhoitusten ja synteettisen puheen esittämisestä käyttäjälle. Tässä yhteydessä käytettäväksi suositellaan SSML-standardia (katso 4.6.). Elementtiä <dtmf> käytetään puheeseen pohjautuvissa sovelluksissa, joissa <grammar> jälleen määrittää sopivan kieliopin ja <bind> käsittelee näppäinpainallukset sekä muut tapahtumat.

4.2. VoiceXML

Voice Extensible Markup Language eli VoiceXML on merkintäkieli, joka on suunniteltu tukemaan esimerkiksi synteettistä puhetta sisältäviä äänivuoropuheluita, puheen tunnistamista ja nauhoitusta [McGlashan *et al.*, 2004]. Toisin kuin SALT, VoiceXML tukee vain puheeseen pohjautuvia sovelluksia.

VoiceXML sai alkunsa vuonna 1999 muutaman tietotekniikkayrityksen perustaessa VoiceXML Forumin. Tällöin esiteltiin VoiceXML 1.0, jota W3C myöhemmin käytti pohjanaan VoiceXML 2.0-standardissaan, joka saavutti standardointiprosessin viimeisen vaiheen keväällä 2004. Nyttemmin on esitelty myös VoiceXML 2.1, joka tuo mukanaan useita uusia ominaisuuksia. Tämä versio ei kuitenkaan vielä ole W3C:n virallinen suositus vaan vasta julkisen testaamisen ensimmäisessä vaiheessa.

VoiceXML perustuu käyttäjän ja sovelluksen väliseen vuoropuheluun, jossa käyttäjä on koko ajan yhdessä keskustelutilassa eli dialogissa kerrallaan

[McGlashan *et al.*, 2004]. Tämä tila määrittää sen mihin tilaan seuraavaksi siirrytään. Dialogeja on kahdenlaisia: lomakkeita ja valikoita. Lomakkeet määrittävät vuorovaikutustavan, jolla kerätään tietoja käyttäjältä. Lomakekenttien yhteydessä voidaan käyttää kielioppia, joka sisältää sallitut syötteet. Valikko esittää käyttäjälle toimintavaihtoehdot ja suorittaa sitten siirtymisen toiseen dialogiin valitun vaihtoehdon pohjalta.

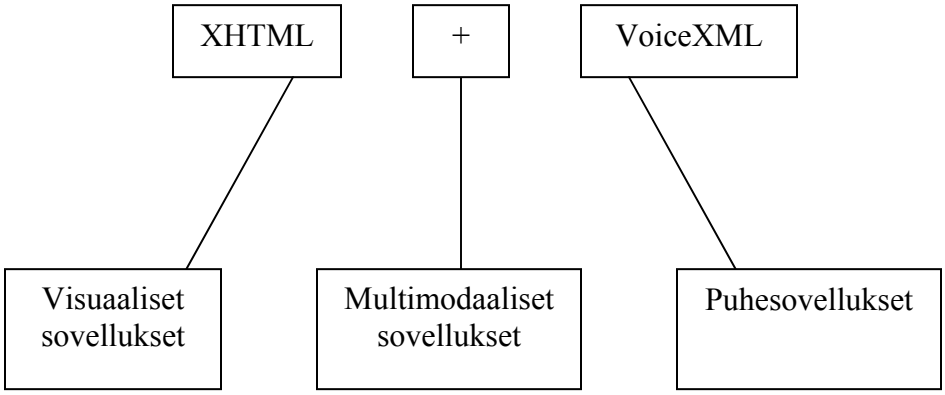
VoiceXML-standardi määrittää useita erilaisia elementtejä, joista <vxml> on juurielementti [McGlashan *et al.*, 2004]. Myös VoiceXML:n pitää tukea yleistä kielioppiformaattia eli SRGS:n XML-mallia. Osa VoiceXML:n elementeistä kuuluu SSML-standardiin, jonka avulla vaikutetaan synteettisen puheen esittämiseen.

4.3. XHTML + Voice

XHTML + Voice eli lyhyemmin X+V on merkintäkieli multimodaalisia WWW-sivustoja varten. Sen avulla voidaan luoda WWW-sivuja, joilla on mahdollista käyttää sekä perinteistä visuaalista vuorovaikutusta että puhetta kommunikoitaessa käyttäjän kanssa [IBM, 2004]. Nimensä mukaisesti X+V käyttää visuaaliseen esittämiseen XHTML:ää ja puheen ilmaisemiseen VoiceXML 2.0 peruselementtejä (Kuva 2). Näiden kahden puolen yhdistämiseen käytetään XML Events-standardia [IBM, 2004].

Myös X+V:n takana on ryhmä tietotekniikkayrityksiä, jotka toimittivat X+V spesifikaation W3C:n multimodaalisen työryhmän harkintaan vuonna 2002. Kyseessä ei siis vielä ole virallinen standardi, vaan SALTin tavoin avoin spesifikaatio, joka kuitenkin pohjautuu W3C:n jo hyväksymiin standardeihin. Tämä voidaankin laskea X+V:n eduksi vertailtaessa sitä SALTin kanssa [IBM, 2004].

X+V jakaa VoiceXML-kielen useisiin moduuleihin, joita on esimerkiksi synteettistä puhetta, kielioppeja ja dialogeja varten [Larson, 2005]. Ottamalla VoiceXML mukaan voidaan vuorovaikutus sovelluksen kanssa kokonaisuudessaan tai paikoitellen hoitaa joko kommunikoimalla puheen avulla tai käyttäen perinteisiä keinoja, kuten näppäimistöä. Erillisiä ohjelmia ei siten tarvita kattamaan eri modaliteetteja, vaan niiden yhdistäminen onnistuu käyttäen <sync> elementtiä [Larson, 2005].



Kuva 2. XHTML ja VoiceXML mahdollistavat yhdistyessään multimodaalisuuden.

4.4. EMMA

Extensible MultiModal Annotation eli EMMA on merkintäkieli, joka pohjautuu XML-kieleen. EMMA on tarkoitettu käytettäväksi järjestelmissä, jotka tulkitsevat useiden eri syötteiden merkityksiä. Tällaisia syötteitä voivat olla esimerkiksi puhe, puhekieleen perustuva teksti, graafisen käyttöliittymän syötteet tai digitaalinen muste [Johnston *et al.*, 2005]. W3C julkaisi syyskuussa 2005 EMMAsta viimeisen teknisen dokumentin kommentoitavaksi, joten virallisen standardin asemaa ei sekään vielä omaa.

EMMA sisältää useita elementtejä ja attribuutteja, joiden avulla eri lähteistä saatu syöte voidaan yhdistää yhtenäiseksi syötteeksi [Larson, 2005]. Näin tietojen välittäminen multimodaalisen järjestelmän eri komponenttien välillä on mahdollista. Komponentteja, jotka tuottavat EMMAa ovat esimerkiksi puheen ja käsinkirjoituksen tunnistajat, näppäimistö ja osoitinlaitteet kuten hiiri [Johnston *et al.*, 2005]. EMMAa käyttävä komponentti puolestaan on vuorovaikutuksenhallinta, jolle EMMAn avulla toimitaan yhtenäinen syöte [Larson, 2005].

4.5. InkML

Ink Markup Language eli InkML on XML-pohjainen merkintäkieli kynäkäyttöliittymille. Tällaiset käyttöliittymät mahdollistavat käsinkirjoittamisen, jolloin tieto voidaan esittää sellaisena kuin käyttäjä sen kirjoittaa. Täten mukana tekstin seassa saattaa esimerkiksi olla muidenkin kielten kirjaimia tai vaikkapa piirroksia. Koska käsinkirjoittaminen on modaliteettina

hyvin tuttu kaikille, pyrkivät käyttäjät käyttämään sitä tiedon syöttämiseen ja hallintaan aina kun mahdollista [Chee *et al.*, 2004].

InkML on W3C:n kehittämä, mutta ei ole vielä ole virallinen standardi, vaan vasta teknisen dokumentin asteella. Kun aiemmin laitteistojen ja sovellusten kehittäjät joutuivat käyttämään digitaalisen musteen esittämiseen ja tallentamiseen omia muiden kanssa yhteensopimattomia formaatteja, mahdollistaa InkML yksinkertaisen ja laitteistoriippumattoman vuorovaikutuksen erilaisten kynäkäyttöliittymien välille.

InkML tukee tarkkaa esitystä digitaalisesta musteesta ja sen muuttujiin voidaan kynän liikkeiden lisäksi tallentaa tietoa myös kirjoitusalustana toimivasta laitteesta [Chee *et al.*, 2004]. Multimodaalisissa sovelluksissa kynän kautta saatavaan tietoon voidaan yhdistää puhe, jolloin käyttäjän on mahdollista esimerkiksi valita kohde kynällään ja tämän jälkeen antaa tarvittava käsky ääntä käyttäen.

InkML määrittelee joukon peruselementtejä, jotka ovat riittäviä kaikille tavallisille kynäsovelluksille. Keskeisimmät elementit ovat juurielementtinä toimiva <ink> ja elementti <trace>, joka pitää sisällään tiedot kynän yhtäjaksoisista liikkeistä alustalla eli koordinaateista, joiden kautta kynä kulkee [Chee *et al.*, 2004]. Yhden <trace> elementin sisältämät tiedot vastaavatkin usein yhtä kirjainta tai sanaa.

4.6. SRGS ja SSML

Speech Recognition Grammar Specification eli SRGS on kielioppiformaatti puheentunnistusta varten. Jotta sovellus osaisi toimia käyttäjän puheen mukaisesti, täytyy puhe ensin analysoida. Tämän hoitaa puheentunnistus, joka perustuu johonkin kielioppiin tai tunnistajan hyväksymiin muodollisiin määrityksiin [Dahl, 2005]. SRGS onkin W3C:n määrittelemä standardi kielioppien kirjoittamiseen.

SRGS:stä on kaksi eri muunnelmaa: XML ja ABNF *(Augmented Backus-Naur Format)*. XML-malli on vaikeaselkoisempi, mutta helpommin koneellisesti manipuloitavissa, kun taas ABNF-malli on suppeampi ja helpompi ihmisen lukea [Dahl, 2005]. Kuten aiemmin mainittiin suositellaan SALTin ja edellytetään VoiceXML:n tukevan SRGS:n XML-mallia. Täydentämään SRGS-standardia W3C on julkaissut SISR *(Semantic Interpretation for Speech Recognition)* -spesifikaation, jonka avulla on mahdollista esimerkiksi täsmentää kieliopissa olevia samantyyppisiä ilmauksia.

Speech Synthesis Markup Language eli SSML on XML:n perustuva merkintäkieli, jonka avulla tekstiin voidaan liittää synteettistä puhetta varten ohjeita siitä miltä tekstin tulisi äänen luettaessa kuulostaa esimerkiksi ääntämisen, äänenvoimakkuuden ja äänenkorkeuden suhteen [Dahl, 2005]. Myös SSML on W3C:n kehittämä standardi, jota voidaan käyttää SALTin yhteydessä ja jota VoiceXML:n tulee tukea.

4.7. SMIL

Synchronized Multimedia Integration Language eli SMIL on HTML:n tapainen merkintäkieli multimedian esittämistä varten. Sen avulla on mahdollista helposti määritellä ja synkronisoida multimediaelementtejä, kuten videota, kuvaa, ääntä ja tekstiä, esitettäväksi halutussa järjestyksessä [Kujanpää, 2000].

SMILin toiminta perustuu aikajanan käyttöön. Jokaiselle elementille määritellään oma aika-arvonsa eli kesto ja ajankohta, jonka aikana sitä esitetään [Kujanpää, 2000]. Määrätyn ajan päätyttyä esitys pysähtyy ja toinen alkaa, joskaan samaan aikaan esittävien elementtien määrää ei ole rajoitettu.

SMIL on W3C:n luoma kieli, josta ensimmäinen julkinen hahmotelma julkaistiin jo vuonna 1997. Nyt SMIL 2.0 on virallinen standardi ja seuraava versio, SMIL 2.1, on jo ehdotetun suosituksen asteella.

SMIL sisältää runsaasti elementtejä, joiden avulla voidaan esittää, koordinoida ja synkronisoida erilaisten mediaelementtien toimintaa. Juurielementtinä on <smil>. Multimediaelementteihin viitataan URL:n avulla, jolloin niiden ei tarvitse sijaita samassa paikassa SMIL-dokumentin kanssa. SMIL mahdollistaa myös sopivan mediaelementin version valinnan, jolloin käyttäjä voi esimerkiksi valita parhaimman version käyttötilanteeseen ja laitteeseensa nähden [Kujanpää, 2000].

5. Standardien arviointia

Edellä esitellyistä suosituksista ja standardeista SALT, X+V, VoiceXML, SRGS ja SSML koskevat lähinnä puhetta hyödyntäviä multimodaalisia käyttöliittymiä. Ainoastaan InkML ja SMIL käsittelevät muitakin vuorovaikutustapoja, kun taas EMMA liittyy eri modaliteettien kautta tulleiden syötteiden yhdistämiseen.

Tästä jakautumisesta on helppo päätellä puheen olevan perinteisen visuaalisen esittämisen lisäksi melko keskeisessä osassa multimodaalisissa käyttöliittymissä. Puheen suosion ymmärtää, kun ajattelee, miten tärkeää se on ihmisten välisen vuorovaikutuksen kannalta. Toisaalta puhejärjestelmiä on myös

ollut olemassa jo kauan verrattuna esimerkiksi käsinkirjoittamiseen, joka on vasta viime vuosina yleistynyt yhtenä vuorovaikutuksen muotona.

Huomattavaa on, ettei yksikään mainituista suosituksista tai standardeista käsittele kosketusta tai tasapainoa. Nämä modaliteetit ovat ahkeran tutkimuksen kohteena eikä niitä hyödyntäviä sovelluksia ole vielä kovinkaan paljon. Ehkä yllättävämpää on se, ettei näköön perustuvia järjestelmiä varten ole standardeja. Kuitenkin on hyvin todennäköistä, että myös näitä modaliteetteja varten tullaan tulevaisuudessa määrittelemään standardeja.

Standardien joukosta puuttuvat myös multimodaalisia sovelluksia kokonaisuudessaan määrittävät standardit. Ainoastaan EMMA ottaa kantaa eri syötteiden yhdistämiseen, mutta muuten yleisiä standardeja ei ole. Jokaiselle modaliteetille on luotu yksi tai useampia standardeja: multimodaalisen sovelluksen toteutuksesta voi näin tulla melkoinen yhdistelmä erilaisia merkintäkieliä. Useiden erilaisten standardien sijaan olisikin pyrittävä yhdistämään multimodaalisuuden eri puolia hallittavaksi kokonaisuudeksi, jolloin standardisoinnin hyödyt parhaiten tulisivat esiin.

6. Yhteenveto

Multimodaalisten käyttöliittymien avulla voidaan varmistaa, että kaikilla käyttäjillä on mahdollisuus valita paras vuorovaikutuksen muoto käyttämänsä laitteen kanssa. Tämä mahdollistaa sekä laitteen käytön monessa erilaisessa tilanteessa että vuorovaikutuksen huolimatta käyttäjän vammoista kuten sokeudesta. Eri modaliteetteja hyödyntäen voidaan vuorovaikutus ihmisen ja tietokoneen välillä toteuttaa vaikkapa visuaalisesti, puheen ja/tai kosketuksen avulla.

Koska multimodaalisuutta hyödyntäviä laitteita on monia, tarvitaan standardeja, joiden avulla eri alustoilla toimivien sovellusten yhteensopivuus voidaan varmistaa. Toisaalta tarvitaan myös arkkitehtuuri, joka antaa puitteet multimodaalisten käyttöliittymien suunnittelulle ja toteutukselle. Tässä kehitystyössä on aktiivisesti toiminut W3C, joka on sekä esittänyt multimodaalisen arkkitehtuurirakenteen että määrittänyt useita suosituksia ja standardeja.

Tulevaisuudessa multimodaalisuuden tarjoamia etuja tullaan hyödyntämään yhä enemmän, kun pienikokoiset mobiililaitteet edellyttävät uusia käyttötapoja perinteisten rinnalle. Parhaillaan W3C työskenteleekin ahkerasti InkML- ja

EMMA-merkintäkielien parissa kehittääkseen niistä virallisia standardeja. Myös multimodaalista arkkitehtuurirakennetta kehitetään edelleen toteuttamaan hyvinmääriteltyjä komponenttirajapintoja ja tapahtumamalleja.

Viiteluettelo

[Adage, 2004] Adage Oy, Käytettävyyssanasto. Maaliskuu 2004.
http://www.adage.fi/artikkelit/kaytettavyyssanasto.html
(tarkistettu 7.10.2005)

[Barnett, 2005] Jim Barnett (ed.), Multimodal Architecture and Interfaces. W3C Working Draft, April 2005.
http://www.w3.org/TR/2005/WD-mmi-arch-20050422/
(tarkistettu 8.10.2005)

[Bunt, 1998] Harry Bunt, Issues in Multimodal Human-Computer Communication. In: Harry Bunt, Robbert-Jan Beun and Tijn Borghuis (eds.), *Multimodal Human-Computer Communication: Systems, Techniques, and Experiments*. Springer, Berlin, 1998, 1-12.

[Chee *et al.*, 2004] Yi-Min Chee (ed.), Max Froumentin (ed.), Jose-Antonio Magaña, Katrin Franke, Gregory Russell, Sriganesh Madhvanath, Giovanni Seni, Christopher Tremblay and Larry Yaeger, Ink Markup Language. W3C Working Draft, September 2004.
http://www.w3.org/TR/2004/WD-InkML-20040928/
(tarkistettu 19.10.2005)

[Dahl, 2005] Deborah Dahl, Guide to Speech Standards. March/April 2005.
http://www.speechtechmag.com/issues/9_8/cover/11619-1.html
(tarkistettu 20.10.2005)

[IBM, 2004] IBM, Multimodal Tools V4.1.2/4.1.2.2. Frequently Asked Questions, July 2004.
ftp://ftp.software.ibm.com/software/pervasive/info/multimodal/multimodal_faq.pdf
(tarkistettu 19.10.2005)

[Johnston *et al.*, 2005] Michael Johnston (ed.), Wu Chou, Deborah A. Dahl, Gerry McCobb and Dave Raggett, EMMA: Extensible MultiModal Annotation markup language. W3C Working Draft, September 2005.
http://www.w3.org/TR/2005/WD-emma-20050916/
(tarkistettu 19.10.2005)

[Kanninen, 2003] Matti Kanninen, *Multimodaalisuus käyttöliittymäsuunnittelijan näkökulmasta: graafisen ja puhekäyttöliittymän symbioosi*. Taideteollinen korkeakoulu, Medialaboratorio, 2003.
http://mlab.uiah.fi/www/projects_and_publications/final_thesis/pdf/kanninen_loppututyo
(tarkistettu 6.10.2005)

[Korpela, 2005] Jukka Korpela, Standardi, mikä se on? Elokuu 2005.
http://www.cs.tut.fi/~jkorpela/stand.html
(tarkistettu 9.10.2005)

[Koskimies ja Mikkonen, 2005] Kai Koskimies ja Tommi Mikkonen, *Ohjelmistoarkkitehtuurit*. Gummerus, Jyväskylä, 2005.

[Kujanpää, 2000] Tomi Kujanpää, SMIL –ohjauskieli. Oulun yliopisto, Tietojenkäsittelytieteiden laitos, Internet ja tietoverkot-opintojakson harjoitustyö, helmikuu 2000.
http://www.student.oulu.fi/~tkujanpa/itv/smil.html
(tarkistettu 20.10.2005)

[Larson, 2005] James A. Larson, Standard Languages for Developing Multimodal Applications. *11th International Conference on Human-Computer Interaction*, 2005.
http://www.larson-tech.com/Writings/multimodal.pdf
(tarkistettu 9.10.2005)

[McGlashan *et al.*, 2004] Scott McGlashan (ed.), Daniel C. Burnett, Jerry Carter, Peter Danielsen, Jim Ferrans, Andrew Hunt, Bruce Lucas, Brad Porter, Ken Rehor and Steph Tryphonas, Voice Extensible Markup Language (VoiceXML) Version 2.0. W3C Recommendation, March 2004.
http://www.w3.org/TR/2004/REC-voicexml20-20040316/
(tarkistettu 17.10.2005)

[Paternò, 2004] Fabio Paternò, Multimodality and Multi-device Interfaces. W3G Workshop of Multimodal Interaction, Position Paper, 2004.
http://www.w3.org/2004/02/mmi-workshop/paterno-cnr.pdf
(tarkistettu 7.10.2005)

[Sà, 2001] Vìtor J. Sà, User Interfaces for Anyone Anywhere. *2nd Conference of the Portuguese Association of Information Systems*, 2001.
https://repositorium.sdum.uminho.pt/retrieve/759/User+Interfaces+for+Anyone+Anywhere.pdf
(tarkistettu 7.10.2005)

[SALT Forum, 2005] SALT Forum, Speech Application Language Tags (SALT) Technical White Paper. 2005.
http://www.saltforum.org/whitepapers/whitepapers.asp
(tarkistettu 17.10.2005)

[SFS, 2005] Suomen Standardisoimisliitto SFS ry, Tietoa standardeista. 2005.
http://www.sfs.fi/standardisointi/tietoa_standardeista
(tarkistettu 9.10.2005)

Multimodaalisten non-WIMP-käyttöliittymien arkkitehtuurit

Satu Mäkitammi

Tiivistelmä.

Perinteiset graafiset käyttöliittymät ovat rakentuneet työpöytä-metaforan mukaisesti ja ovat perustuneet pitkälti suorakäyttöisyyteen sekä yksimodaalisuuteen. Teknologian kehittymisen sekä uudenlaisten käyttötarpeiden myötä ovat multimodaalisuutta hyödyntävät non-WIMP-tyyliset käyttöliittymät kuitenkin lisääntyneet. Niitä käytetään muun muassa virtuaalitodellisuussovelluksissa, uudenlaisissa peleissä, sulautetuissa järjestelmissä sekä mobiilikäytössä. Tällaisten käyttöliittymien erityispiirteet – esimerkiksi jatkuva vuorovaikutus, reaaliaikaisuuden vaatimus sekä useat vuorovaikutuskanavat – asettavat suuren haasteen käyttöliittymän suunnittelulle ja arkkitehtuurille. Tässä tutkielmassa käsitellään non-WIMP-käyttöliittymiin – erityisesti niiden multimodaalisuuteen – liittyviä ominaispiirteitä sekä niiden vaikutusta käyttöliittymäarkkitehtuuriin.

Avainsanat ja -sanonnat: non-WIMP -käyttöliittymä, multimodaalisuus, moniagenttiarkkitehtuuri.
CR-luokat: H.5.2, I.2.11

1. Johdanto

Tällä hetkellä laajimmin käytössä ovat edelleen perinteiset graafiset, WIMP-tyyliset (Window, Icon, Menu, Pointer) käyttöliittymät, joissa vuorovaikutus käyttäjän ja tietokoneen välillä tapahtuu pääasiassa yhden syöte- ja tulostevirran välityksellä ja on luonteeltaan jaksottaista ja vuorottelevaa. Teknologisen kehityksen ja uusien käyttötarpeiden myötä ovat uudenlaiset multimodaaliset ja non-WIMP-tyyliset käyttöliittymät kuitenkin yleistyneet kasvavaa tahtia. Tällaisten ns. "uuden sukupolven käyttöliittymien" perusominaisuudet ja vuorovaikutustavat – mm. jatkuvat syötteet, vaatimus reaaliaikaisuudesta, useat vuorovaikutuskanavat sekä vaihtelevat modaliteettien integrointitavat – eroavat suuresti perinteisistä WIMP-käyttöliittymistä.

Aikaisemmin käyttöliittymä on saatettu nähdä yksiulotteisesti, ainoastaan ohjelmiston yhtenä pienenä osasena tai pelkkänä visuaalisena pintana, joka noudattaa muun ohjelmiston arkkitehtuuria ja joka vain liitetään varsinaisen toiminnallisuuden päälle. Perinteisiä WIMP-tyylisiä käyttöliittymiä varten on olemassa myös useita vakiintuneita ja standardoituja arkkitehtuuriratkaisuja, suunnittelumalleja sekä työkaluja. Uudenlaiset multimodaaliset, non-WIMP-tyyliset käyttöliittymät ovat asettaneet käyttöliittymäsuunnittelijat kuitenkin aivan uuden haasteen eteen. Tällaiset käyttöliittymät joutuvat käsittelemään määrältään suurempia ja monimutkaisempia tietoja sekä samalla vastaamaan tiukkoihin aikavaatimuksiin. WIMP-käyttöliittymissä hyvin toimivien arkkitehtuurien soveltaminen suoraan ei olekaan mielekästä tai kannattavaa, jossain tapauksessa se voi olla jopa mahdotonta. Tämän vuoksi multimodaalisia non-WIMP-käyttöliittymiä varten tulee suunnitella ja kehitellä uudenlaisia arkkitehtuuriratkaisuja, kuvauskieliä ja muita suunnittelutyökaluja, jotka ottavat paremmin huomioon niiden erityispiirteet ja tarpeet. Käyttöliittymän arkkitehtuurisuunnittelun rooli heti ohjelmistoprojektin alussa korostuu, sillä käyttöliittymän toteutusarkkitehtuurilla saattaa olla vaikutusta koko muun sovelluksen arkkitehtuuriratkaisuihin.

Tässä tutkimuksessa tarkastelen multimodaalisten, non-WIMP-tyylisten käyttöliittymien ohjelmistoarkkitehtuureja. Aluksi käsittelen non-WIMP-käyttöliittymien erityispiirteitä ja niille tyypillistä vuorovaikutusta. Lähempään tarkasteluun otan käyttöliittymän multimodaalisuuden ja sen asettamat haasteet käyttöliittymäsuunnittelulle. Tämän jälkeen esittelen moniagenttisuuteen perustuvaa arkkitehtuuria, joka sopii multimodaalisiin non-WIMP-käyttöliittymiin erityisen hyvin joustavuutensa, jaettavuutensa sekä multimodaalisuutensa ansiosta. Lopuksi esittelen vielä mallin, joka jakaa non-WIMP-käyttöliittymässä tapahtuvan vuorovaikutuksen kahteen erilliseen komponenttiin vuorovaikutuksen jatkuvuuden ja erillisyyden perusteella.

2. Non-WIMP-käyttöliittymät

Perinteisiin työpöytä-metaforan mukaisiin ja tavallisesti suorakäyttöisyyteen perustuviin käyttöliittymiin viitataan usein nimityksellä WIMP-käyttöliittymä, joka tulee sanoista Window, Icon, Menu ja Pointer. Monesti WIMP-käyttöliittymä ja graafinen käyttöliittymä (GUI) tulkitaan samaksi käsitteeksi. Uudenlaisiin työpöytä-metaforasta poikkeaviin käyttöliittymiin puolestaan

viitataan termillä non-WIMP-käyttöliittymä. Tällaisten käyttöliittymien piiriin kuuluvat mm. virtuaalitodellisuussovellukset, sulautettujen järjestelmien käyttöliittymät sekä muut uudenlaisia syöte- tai tulostetapoja hyödyntävät käyttöliittymät [Green and Jacob, 1991].

2.1. Non-WIMP-käyttöliittymän ominaispiirteet

Non-WIMP-käyttöliittymät eroavat suuresti perinteisistä käyttöliittymistä käyttäjän ja tietokoneen välisen vuorovaikutuksen osalta. WIMP-tyylisissä käyttöliittymissä vuorovaikutus muodostuu yleensä vuorottaisesta dialogista yksittäisten ja erillisten syöte- ja tulostevirtojen välityksellä, kun taas non-WIMP-käyttöliittymissä vuorovaikutus on rinnakkaista, jatkuvaa ja yleensä myös multimodaalista [Jacob et al., 1999].

Green and Jacob [1991] ovat määritelleet viisi non-WIMP-käyttöliittymille ominaista piirrettä, joiden huomioonottaminen on erityisen tärkeää käyttöliittymää suunniteltaessa ja toteutettaessa. Ensimmäiseksi piirteeksi he mainitsevat paljon kaistanleveyttä vaativat syötteet ja tulosteet. Käyttöliittymän on pystyttävä käsittelemään suuria syötemääriä ja vastaamaan niihin tarkoituksenmukaisella tavalla. Esimerkiksi näkymän päivitys virtuaalitodellisuussovelluksessa tai kynän sijainnin mittaaminen käsialasyötteessä täytyy suorittaa monia, mahdollisesti monia kymmeniä, kertoja sekunnissa.

Toinen ominaispiirre liittyy tarjolla olevien vaihtoehtojen laajuuteen sekä syöte- ja tulostelaitteiden että sovelluksen toiminnassa. Käyttöliittymän on silti pystyttävä säilyttämään sovelluksen helppokäyttöisyys. Kolmas ominaispiirre puolestaan sisältää vaatimuksen siitä, että käyttäjän toimintoihin on reagoitava reaaliajassa. Esimerkiksi virtuaalitodellisuussovelluksissa käyttäjän päänliikkeen ja sen mukaisesti päivitetyn näkymän välinen viive saa olla enintään 0,4 sekuntia, jotta käyttäjällä säilyy todenmukainen tuntuma kolmiulotteiseen ympäristöön. Samoin kynäsyötettä vastaavan piirtojäljen tulee näkyä välittömästi käyttäjälle, jotta syötetapa tuntuisi luontevalta ja todenmukaiselta.

Neljäs non-WIMP-käyttöliittymien ominaispiirre määrittelee, että syötteisiin reagoinnin ja palautteenannon tulee olla jatkuvaa. Suurin osa non-WIMP-käyttöliittymissä annetuista syötteistä on jatkuvia, toisin kuin WIMP-käyttöliittymille tyypilliset erilliset komentotyyliset syötteet, minkä vuoksi käyttöliittymän on myös jatkuvasti vastattava käyttäjän syötteisiin. Esimerkiksi käyttäjän liikkuessa virtuaalitodellisuudessa käyttöliittymä ei voi odottaa

käyttäjän liikkeen loppumista, vaan näkymää täytyy päivittää jatkuvasti liikkeen aikana ja käyttöliittymän tulee olla jatkuvasti valmis reagoimaan liikkeen aiheuttamiin tilanteisiin (esimerkkinä törmääminen seinään tai uuden esineen tuleminen tartuntaetäisyydelle).

Listan viimeinen ominaispiirre on todennäköisten syötteiden käyttäminen. Non-WIMP-käyttöliittymissä palautteen tulee olla reaaliaikaista, mutta samalla monet tunnistukseen perustuvat modaliteetit saattavat tuottaa epätarkkoja syötteitä. Tämän vuoksi käyttöliittymä saattaa joutua arvaamaan käyttäjän tarkoittaman syötteen tai valitsemaan todennäköisimmän tunnistustuloksen. Jos käyttöliittymä huomaa myöhemmin, oikean merkityksen varmistuessa, valinneensa väärän tulkinnan, annettu palaute joudutaan korjaamaan. Esimerkiksi käyttäjän kirjoittaessa syötteensä kynän avulla käyttöliittymä tulostaa aluksi todennäköisimmät kirjaimet ja koko sanan valmistuttua mahdollisesti korjaa väärin tulkittuja kirjaimia.

Taulukkoon 1 on koottu vertailua varten sekä nykyisille WIMP-tyylisille käyttöliittymille että uudenlaisille non-WIMP-käyttöliittymille ominaisia piirteitä [Shaer and Jacob, 2005]. Vertailemalla näitä ominaispiirteitä voidaan huomata uudenlaisten käyttöliittymien mahdollisesti tarjoavan käyttäjälle helppokäyttöisemmän, luonnollisemman ja monipuolisemman tavan olla vuorovaikutuksessa tietokoneen kanssa. Toisaalta sellaiset piirteet, kuten multimodaalisuus tai vaatimus reaaliaikaisuudesta, tekevät monesti non-WIMP-käyttöliittymien suunnittelusta haastavampaa ja toteuttamisesta vaikeampaa.

Nykyiset WIMP-käyttöliittymät	Uuden sukupolven käyttöliittymät
Yksisäikeinen dialogi	Samanaikaiset, synkronoimattomat dialogit
Erilliset ja täsmälliset merkit	Jatkuvat sekä erilliset syötteet ja tulosteet, todennäköiset syötteet
Järjestys on merkityksellistä, ei aika	Vaatimus reaaliaikaisuudesta, määräaikaan perustuvat laskelmat
Selvät komennot	Tarkoituksellinen sekä passiivinen vuorovaikutus
Ennalta määritellyt interaktiiviset kohteet ja niiden käyttäytyminen	Uudenlaiset kohteet ja interaktiiviset käyttäytymistavat
Yksittäinen tulostekanava	Rinnakkaiset digitaaliset ja fyysiset tulostekanavat
Standardi laitteisto syötteiden ja tulosteiden antamiseen	Hyödyntää laajaa valikoimaa laitteita ja eri aisteihin perustuvia teknologioita
Toimii pysyvässä ja eristetyssä laskennallisessa ympäristössä	Toimii dynaamisessa, ad hoc -tyyppisessä, heterogeenisessa ympäristössä

Taulukko 1: Vertailussa nykyisten ja uuden sukupolven käyttöliittymien ominaispiirteet [Shaer and Jacob, 2005].

3. Multimodaalisuus non-WIMP-käyttöliittymässä

Non-WIMP-käyttöliittymien vuorovaikutus on yleensä multimodaalista eli syötteiden ja tulosteiden antamisessa käytetään useampaa erilaista tapaa ja laitetta. Monet uusista syötetavoista – mm. eleet, puhe, käsiala ja katse – perustuvat tunnistukseen ja sen todennäköisyyteen. Käyttäjillä on lisäksi erilaisia tapoja sekä mieltymyksiä yhdistellä tarjolla olevia syötemodaliteetteja, ja lisäksi käyttäjien – niin taidolliset kuin fyysisetkin – ominaisuudet sekä erilaiset käyttötilanteet tai -tarkoitukset ohjaavat käytettyjä vuorovaikutustapoja. Kun nämä seikat yhdistetään non-WIMP-käyttöliittymille luontaiseen jatkuvaan vuorovaikutukseen ja reaaliaikaisuuden vaatimukseen, joudutaan käyttöliittymäarkkitehtuuria suunniteltaessa entistä suuremman haasteen eteen.

3.1. Multimodaalisuus käsitteenä

Käyttöliittymän multimodaalisuudella tarkoitetaan yleisesti ihmisen ja tietokoneen välillä tapahtuvaa vuorovaikutusta, jossa hyödynnetään useaa eri kommunikaatiokanavaa. Syötteiden tai tulosteiden antamisessa yhdistetään siis vähintään kaksi eri modaliteettia, jotka tuotetaan vähintään kahdella eri laitteella [Schomaker et al., 1995]. Nigay and Coutaz [1993] ovat rajanneet tätä multimodaalisuuden määritelmää merkityksen käsitteen avulla. He ovat

määritelleet multimodaalisuuden järjestelmäsuuntautuneesta näkökulmasta niin, että sillä tarkoitetaan järjestelmän kykyä olla vuorovaikutuksessa käyttäjän kanssa erityyppisten kommunikaatiokanavien kautta sekä automaattisesti tulkita ja välittää merkityksiä tässä vuorovaikutusprosessissa. Määritelmän perusteella syötteen antaminen sekä puheen että eleen avulla on multimodaalista. Toisaalta valmiin kuva- ja äänimateriaalin pelkkä toistaminen ei tee tulosteesta multimodaalista, vaikkakin tulosteen antamiseen käytetään kahta eri modaliteettia. Tällaiset tulosteet kuuluvat multimedian piiriin, sillä multimodaalisen tulosteen avulla tulee välittää jotakin merkitystä. Esimerkiksi käyttäjän suorittamasta virhetoiminnasta kertominen sekä äänimerkin että visuaalisen virheilmoituksen avulla täyttää multimodaalisuuden vaatimukset.

Modaliteetilla tarkoitetaan tässä tutkimuksessa Nigay and Coutazin [1993] määritelmän mukaisesti sekä informaatiota välittävän kommunikaatiokanavan tyyppiä että sitä tapaa, jolla sanoma on ilmaistu tai vastaanotettu. Näin ollen modaliteettien avulla välitetään merkitystä sisältävää informaatiota, jonka käyttäjä vastaanottaa jonkin aistinsa välityksellä tai tietokone vastaanottaa ihmisaistia vastaavalla tavalla kuten "keinonäkönä" toimivan kameran kautta. Ihmisaistien mukaisesti modaliteetit voidaan jakaa kuuteen kategoriaan: visuaalinen (näköaisti), auditiivinen (kuuloaisti), haptinen (tuntoaisti), vestibulaarinen (tasapaino- ja liikeaisti), olfaktorinen (hajuaisti) ja gustatoorinen (makuaisti) modaliteetti. Näiden lisäksi omaksi modaliteetikseen voidaan erottaa käyttäjän aivotoimintaan perustuva vuorovaikutus, jossa käyttäjä antaa syötteensä aivokäyttöliittymälle pelkkien ajatustensa avulla.

3.2. Multimodaalisuuden edut

Multimodaalisuuden avulla pyritään tekemään käyttäjän ja tietokoneen välinen vuorovaikutus luonnollisemmaksi, helpommaksi ja joustavammaksi. Käyttäjälle tarjoutuu tilaisuus valita haluamansa syötemodaliteetti sekä vaihdella ja yhdistellä eri modaliteetteja omien mieltymystensä, fyysisten rajoitustensa, suorittamiensa tehtävien tai käyttöympäristön mukaan. Tällä tavalla voidaan saavuttaa laajempi kirjo erilaisia käyttäjiä ja käyttötilanteita. Myös käyttäjien tietoturvaa ja yksityisyyden suojaa voidaan parantaa tarjoamalla vaihtoehtoisia syöte- tai tulostetapoja tilanteissa, joissa käsitellään henkilökohtaisia tunnistetietoja tai muuta yksityistä informaatiota. Käyttäjien luotettavaa tunnistusta voidaan puolestaan parantaa yhdistämällä yhtäaikaisesti useita erilaisia biometrisia tunnisteita eli ihmisen fysiologisia tai käyttäytymiseen

liittyviä piirteitä (mm. sormenjäljet, kasvonpiirteet, silmän iiris, allekirjoitus tai ääni) hyödyntäviä syötemodaliteetteja [Jain and Ross, 2004].

Multimodaalisuuden avulla voidaan saavuttaa etuja myös järjestelmän kannalta. Sen suorituskykyä, kestävyyttä ja luotettavuutta voidaan saada parannettua monella tavalla. Ensinnäkin multimodaalisen käyttöliittymän joustavuus estää yksittäisen modaliteetin ylikuormittumisen sekä auttaa virheiden ennaltaehkäisyssä sekä niistä toipumisessa [Oviatt, 1999]. Virheiden ennaltaehkäisyyn auttaa suuresti se, että käyttäjälle voidaan tarjota mahdollisuus valita omiin ominaisuuksiinsa, tehtävään tai tilanteeseen parhaiten soveltuva modaliteetti. Käyttäjä voi myös pyrkiä varmistamaan syötteensä oikean tulkinnan antamalla sen useamman modaliteetin avulla, varsinkin jos edes toinen modaliteeteista on tulkintaan perustuva tai jos tilanne sisältää häiriötekijöitä. Yhdistetyllä tulkinnalla saadaan lisättyä oikean tulkinnan todennäköisyyttä [Kaiser et al., 2003; Oviatt, 2001]. Tulkintavirheitä ennaltaehkäisee myös multimodaalisen kielen erilaisuus verrattuna yksimodaalisiin puhesyötteisiin. Multimodaalisissa syötteissä lauserakenteet ovat lyhyempiä ja lauseopillisesti yksinkertaisempia [Oviatt *et al.,* 1997] ja lisäksi pronominien käyttö on tehokasta ja luonnollista [Bolt, 1980]. Nämä seikat parantavat syötteen tunnistustarkkuutta ja vähentävät puheentunnistustekniikalle asetettavia vaatimuksia. Jos tulkintavirhe kuitenkin pääsee tapahtumaan, siitä toipumista edesauttaa käyttäjän mahdollisuus vaihtaa kokonaan toiseen modaliteettiin tai yhdistää useita modaliteetteja tulkintatuloksen parantamiseksi.

3.3. Multimodaalisuuden asettamat haasteet käyttöliittymäarkkitehtuurille

Multimodaalisen käyttöliittymän suunnittelussa on otettava huomioon useita erilaisia näkökohtia, jotta käyttöliittymästä saadaan helppokäyttöinen, tehokas ja luotettava. Oviattin [1999] mukaan onnistunut suunnittelu pohjautuu perusteelliseen tietoon eri modaliteettien ominaisuuksista, multimodaalisen kielen erityispiirteistä, syötteiden yhdistelytavoista sekä käyttäjien vaihtelevista vuorovaikutustavoista. Hänen mukaansa multimodaalisten käyttöliittymäarkkitehtuurien suurimpia haasteita onkin käsitellä rinnakkaisia ja toisiinsa, niin ajallisesti kuin merkitykseltään, liittyviä syötteitä sekä optimoida luotettavuus ja kestävyys.

3.3.1. Käyttäjien vuorovaikutustavat

Ihmisten välinen vuorovaikutus on luonnostaan multimodaalista, sillä siinä käytetään useita aisteja viestien välittämiseen ja vastaanottamiseen sekä merkityksen tulkitsemisen apuna. Ihmisten on todettu olevan mielellään multimodaalisessa vuorovaikutuksessa myös tietokoneiden kanssa, mutta heillä on lisäksi tapana vaihdella multimodaalisen ja yksimodaalisen syötetavan välillä [Oviatt, 1999]. Syötetavan valintaan vaikuttaa niin tehtävän luonne, käyttäjän ominaisuudet kuin käyttöympäristökin.

Karttakäyttöliittymän avulla suorittamissaan kokeissa Oviatt *et al.* [1997] huomasivat syötetavan riippuvan selkeästi suoritettavana olevan toiminnon luonteen mukaan. Käyttäjät ilmaisivat syötteensä multimodaalisesti todennäköisimmin spatiaalisissa eli tilallista kuvausta vaativissa komennoissa, kuten "add" tai "move". Sen sijaan pelkkään näkyvän kohteen tunnistamiseen ja valintaan liittyvät komennot, kuten "label" tai "delete" suoritettiin multimodaalisesti vain 14-36%:n todennäköisyydellä ja yleisissä ohjauskomennoissa, kuten "print", käytettiin lähes pelkästään yksimodaalisia syötteitä. Multimodaalisten syötteiden käytön on havaittu lisääntyneen merkittävästi myös tehtävän vaikeustason noustessa sekä uutta käsitettä tai asiayhteyttä luotaessa [Oviatt *et al.*, 2004]. Multimodaalisen syötteen avulla käyttäjät voivat osittaa syöteinformaationsa usealle eri modaliteetille, ja tällä tavalla he voivat paremmin hallita työmuistinsa rajoituksia ja kasvanutta kognitiivista kuormitusta.

Käyttäjien tavassa yhdistää eri modaliteetteja on myös havaittavissa eroja, jotka vaikuttavat suuresti syötteenkäsittelyyn. Oviatt *et al.* [2004] tutkimusten mukaan käyttäjät voidaan jakaa kahteen ryhmään pääasiallisen yhdistelytapansa mukaan. Toinen ryhmä käyttää lähes yksinomaan samanaikaista yhdistelytapaa, jossa eri modaliteettien syötteet ovat ajallisesti ainakin osittain päällekkäisiä. Toinen ryhmä puolestaan antaa syötteensä peräkkäisesti niin, että eri modaliteeteilla annettujen syötteiden välissä on pieni viive. Lisäksi eri modaliteettien käyttöjärjestys vaihtelee käyttäjien välillä. Esimerkiksi kirjoitettu syöte edeltää yleensä puhesyötettä käyttäjän yhdistelytavasta riippumatta [Oviatt *et al.*, 1997].

Muita multimodaalisuuden hyödyntämiseen vaikuttavia tekijöitä ovat käyttäjien, niin fyysiset kuin taidolliset, ominaisuudet sekä käyttötilanteen luonne. Fyysiset rajoitukset, kuten puhekyvyttömyys tai motoriset rajoitteet,

saattavat vaikeuttaa tai estää kokonaan jonkin modaliteetin käytön. Tällainen käyttäjä hyötyy multimodaalisen käyttöliittymän tarjoamista vaihtoehtoisista modaliteeteista, mutta toisaalta hän ei välttämättä pysty hyödyntämään modaliteettien yhdistelymahdollisuuksia. Käyttötilanne voi määritellä myöskin multimodaalisuuden astetta. On tilanteita, joissa yksittäisen modaliteetin hetkellinen vaihtaminen tai useamman modaliteetin yhdistäminen on välttämätöntä. Esimerkkeinä voidaan mainita puhesyötteeseen siirtyminen käsien ollessa varattuina tai yhdistetyn puhe- ja elesyötteen käyttäminen häiriöllisessä ympäristössä tunnistustuloksen parantamiseksi.

3.3.2. Modaliteettien integrointi

Multimodaalisen käyttöliittymän suunnitteluun ja arkkitehtuuriratkaisuihin vaikuttaa suuresti se, millaisia modaliteettien yhdistely- ja integrointitapoja tullaan käyttämään. Valintaan vaikuttavat monet tekijät: järjestelmän aihealue, käyttötarkoitus, suoritettavien tehtävien laatu, käytettävien modaliteettien erityispiirteet, käyttöympäristö sekä oletettujen käyttäjien ominaisuudet. Näitä tekijöitä analysoimalla voidaan valita parhaiten kyseiseen käyttöliittymään soveltuva integrointitapa. Toisaalta käyttöliittymän eri osien välillä voidaan vaihdella käytettävää integrointitapaa, tai multimodaalisuutta voidaan myös hyödyntää ainoastaan joissakin käyttöliittymän osissa.

Kaaviossa 1 on esitelty Nigay and Coutazin [1993] luokittelu, jossa erityyppiset modaliteettien käyttötavat on jaettu neljään kategoriaan olettaen, että multimodaalisuus määritellään tulkintaa sisältäväksi vuorovaikutukseksi eli abstraktiotasoltaan merkitys-luokkaan kuuluvaksi. Eri kategorioiden välillä modaliteettien yhtäaikainen käyttömahdollisuus ja niiden yhdistetyn tulkinnan mahdollisuus vaihtelevat. Kaaviossa modaliteettien käyttö -ulottuvuudella tarkoitetaan sitä, ovatko eri modaliteetit käytettävissä samanaikaisesti vai ainoastaan yksi kerrallaan. Modaliteettien yhdistettävyys -ulottuvuus puolestaan jaottelee käyttöliittymät sen mukaan, onko eri modaliteettien kautta tuotetut syötteet tulkittavissa yhdessä vai erillään toisistaan riippumattomina syötteinä.

		Modaliteettien käyttö	
		Peräkkäinen	Rinnakkainen
yhdistettävyys	Yhdistetty	Vuorottainen (alternate)	Yhteisvaikutteinen, synerginen (synergistic)
	Riippumaton	Poissulkeva (exclusive)	Samanaikainen (concurrent)
		Merkitys/merkityksetön	Merkitys/merkityksetön
		Abstraktiotaso	

Kaavio 1: Multimodaalisten käyttöliittymien luokittelu modaliteettien integrointitapojen mukaan [Nigay and Coutaz, 1993].

Jokainen kaavion kategoria asettaa omat haasteensa syötteiden käsittelylle. Toteutukseltaan poissulkeva-integrointitapa on yksinkertaisin ja vastaa pöytäkoneiden syötteidenantoa hiiren ja näppäimistön välityksellä. Toisaalta synerginen-integrointitapa asettaa suurimmat vaatimukset järjestelmän suunnittelulle, toteutukselle ja tekniikalle. Vaikka synerginen tapa tarjoaa yleisesti ottaen luonnollisimman ja tehokkaimman vuorovaikutustavan, täytyy ottaa kuitenkin huomioon, että eri käyttäjillä on taipumus luonnostaan suosia joko peräkkäistä tai rinnakkaista syötteiden antotapaa. Oman ongelmansa muodostaa puolestaan se, miten järjestelmä osaa rajata peräkkäisistä syötteistä kulloinkin käyttäjän tarkoittaman syötekokonaisuuden. Ovatko peräkkäiset syötteet tulkittava irrallisina ja toisistaan täysin riippumattomina vai toisiinsa liittyvinä syötekokonaisuuden osina? Eri modaliteeteilla annettujen syötteiden keskinäisestä järjestyksestä sekä niiden välisestä keskimääräisestä viiveestä voidaan ennustaa todennäköisyyttä sille, onko kysymyksessä yhdistetty multimodaalinen syöte vai kaksi erillistä yksimodaalista syötettä [Oviatt et al., 1997].

3.3.3. Vaihtuva laiteympäristö

Monet uudenlaisissa käyttöliittymissä käytettävistä modaliteeteista, kuten luonnollinen puhe ja katse, kehittyvät jatkuvasti teknisesti, ja sitä myötä myös niiden käytettävyys sekä hyödynnettävyys paranee. Multimodaalisen

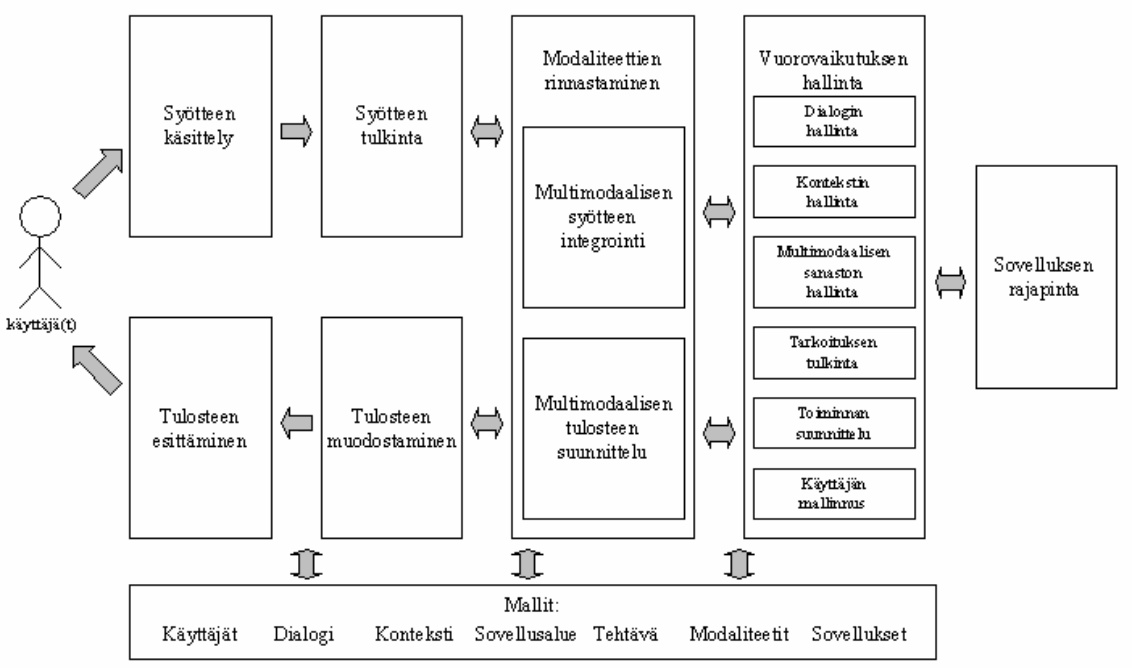
käyttöliittymän tulisikin olla helposti laajennettavissa niin, että siihen on helppo lisätä uusia modaliteetteja sekä kehittyneempiä versioita modaliteettien käsittelyyn. Multimodaaliseen käyttöliittymään liitettyjen syöte- ja tulostelaitteiden kokoonpano saattaa muuttua myös dynaamisesti ohjelman suorittamisen aikana. Laitteet saattavat myös liittyä käyttöliittymään vain hetkellisesti. Tässä tapauksessa käyttöliittymien pitäisi kaiken aikaa sopeutua hyödyntämään kullakin hetkellä tarjolla olevia syöte- ja tulostelaitteita.

4. Multimodaalisten non-WIMP-käyttöliittymien arkkitehtuurit

Koska multimodaalisten non-WIMP-käyttöliittymien ominaispiirteet ja niiden asettamat vaatimukset poikkeavat niin merkittävästi perinteisten WIMP-käyttöliittymien ominaisuuksista, ei niiden pohjalle rakennettuja valmiita suunnittelumalleja ja käyttöliittymäarkkitehtuureita voida soveltaa suoraviivaisesti. Soveltaminen vaatisi ad hoc -tyylistä koodin muokkaamista, jolloin arkkitehtuurisuunnittelun tavoitteet, kuten helppo ylläpidettävyys, laajennettavuus ja uudelleenkäytettävyys, eivät olisi enää helposti saavutettavissa. Tämän vuoksi on kehitettävä uudenlaisia käyttöliittymän organisointi- ja toteutustapoja sekä arkkitehtuuriratkaisuja, jotka huomioivat paremmin ko. käyttöliittymien erityispiirteet ja tarpeet sekä soveltuvat näin ollen paremmin niiden kuvaamiseen. Lisäksi tarvitaan kuvauskieliä ja muita suunnittelutyökaluja, joiden avulla arkkitehtuurin mukaisia käyttöliittymiä voidaan helposti rakentaa. Laaja-alaisesti hyödynnettävät ja toimivat ratkaisut tulisi pyrkiä standardoimaan, jotta myös multimodaalisille non-WIMP-käyttöliittymille saataisiin tulevaisuudessa luotua yhtenäiset suunnittelukäytännöt.

Kuvassa 1 on esitetty korkean tason käyttöliittymäarkkitehtuuri, joka on mukailtu Maybury and Wahlsterin [1998] esittämästä älykkäisiin multimodaalisiin käyttöliittymiin tarkoitetusta arkkitehtuurikaaviosta. Arkkitehtuurin mukaisesti käyttäjän antama syöte käsitellään aluksi modaliteettikohtaisesti kahdessa vaiheessa: syötelaitteen antaman raakasyötteen käsittely sekä syvempi merkityksen tulkinta perustuen modaliteettiin liittyviin malleihin, kuten luonnolliseen kieleen tai eleisiin. Tämän jälkeen eri modaliteeteilla annetut yhteenkuuluvat syötteet integroidaan ja niistä muodostetaan yhdistetty tulkinta. Integroidun tulkinnan muodostamisessa käytetään apuna tietoja dialogin tilasta, käyttäjämalleista sekä syötehetkellä

vallinneesta kontekstista, kuten ajasta, paikasta, sovellusalueesta tai suoritettavana olevasta tehtävästä. Kun käyttäjän tarkoittama toiminta on saatu tulkittua multimodaalisesta syötteestä, käyttöliittymä välittää mahdolliset toimintapyynnöt taustajärjestelmälle ja lopuksi suorittaa asianmukaisten tulosteiden ja palautteenannon käyttäjälle. Multimodaaliset tulosteet muodostetaan kolmivaiheisesti. Ensin suunnitellaan multimodaalisen tulosteen sisältö, käytettävät modaliteetit, tulosteen jako modaliteettien kesken sekä tulosteen yhtenäinen "ulkomuoto". Suunnittelun perusteella tuloste muodostetaan ja esitetään käyttäjälle modaliteettikohtaisesti.



Kuva 1: Korkean tason arkkitehtuuri älykkäille multimodaalisille käyttöliittymille. Mukautettu Maybury and Wahlsterin [1998] esittämästä arkkitehtuurista.

Kuvan 1 arkkitehtuuri soveltuu mainiosti multimodaalisten käyttöliittymien kuvaamiseen, koska se ottaa huomioon useiden eri modaliteettien kautta annettujen syötteiden integroimisen sekä multimodaalisten tulosteiden muodostamisen. Arkkitehtuurissa otetaan myös huomioon niin käyttäjien, tehtävien, kontekstin, tarjolla olevien modaliteettien kuin sovellusalueenkin vaikutus multimodaaliseen vuorovaikutukseen. Jotta arkkitehtuuri soveltuisi

non-WIMP-tyylisiin multimodaalisiin käyttöliittymiin, on sen mahdollistettava jatkuva syötteiden antaminen sekä välittömän palautteen tarjoaminen. Eräs tehokas ratkaisu tähän on yllä olevan arkkitehtuurin muuntaminen moniagenttiarkkitehtuuriksi, jota käsitellään tarkemmin seuraavassa luvussa.

5. Moniagenttiarkkitehtuuri

Moniagenttiarkkitehtuurit, kuten Open Agent Architecture (OAA), soveltuvat hyvin käytettäväksi multimodaalisissa non-WIMP-käyttöliittymissä, koska ne ovat joustavia ja pystyvät käsittelemään tehokkaasti usean eri modaliteetin kautta tulevia samanaikaisia syötteitä. Moniagenttiarkkitehtuuri koostuu nimensä mukaisesti useista itsenäisistä agenteista, joilla on omat rajatut tehtävänsä. Agentti saa syötteensä joko suoraan käyttäjältä tai toiselta agentilta, käsittelee tai analysoi sitä oman tehtävänsä mukaisesti – pyytäen mahdollisesti lisäsyötteitä muilta agenteilta – ja lopulta antaa tulosteensa toisen agentin syötteeksi.

5.1. Agentit

Moniagenttiarkkitehtuurin mukainen multimodaalinen non-WIMP-käyttöliittymä muodostuu useista agenteista, jotka ovat erikoistuneet suorittamaan oman, tarkoin rajatun tehtävänsä. Agenttien lukumäärä kasvaa sitä mukaa kuin uusia modaliteetteja lisätään, sillä jokainen modaliteetti tarvitsee omat agenttinsa syötteen käsittelyyn ja analysointiin. Arkkitehtuurin sisältämät agentit vaihtelevat sovelluksen ja modaliteettivalikoiman mukaan. Agenttivalikoimaan vaikuttaa myös sallittu vuorovaikutus agenttien välillä eli toimiiko vuorovaikutus keskeisen välittäjä-agentin (*facilitator agent*) kautta vai sallitaanko agenttien olla vuorovaikutuksessa suoraan toistensa kanssa. Vaihtelusta huolimatta on mahdollista hahmottaa muutamia agenttityyppejä, jotka ovat keskeisiä kaikille multimodaalisille non-WIPM-käyttöliittymille.

5.1.1. Käyttöliittymäagentti

Käyttöliittymäagentti toimii keskeisessä ohjaus- ja koordinointitehtävässä käyttäjän ja muiden agenttien välillä. Se vastaanottaa kaikki käyttäjältä tulevat syötteet ja välittää ne eteenpäin timestamp-tietojen kera kyseisten modaliteettien käsittelystä ja analysoinnista vastaaville agenteille. Käyttöliittymäagentti voi vastaanottaa sekä yksimodaalisia että multimodaalisia syötteitä.

Jos käyttäjä esimerkiksi antaa samanaikaisesti syötteet sekä puheen että eleen avulla, käyttöliittymäagentti välittää puhesyötteen puheentunnistusagentille ja elesyötteen puolestaan eleentunnistusagentille, jotka palauttavat käsittelemänsä syötteen takaisin käyttöliittymäagentille. Tarvittaessa tämä "raakasyöte" välitetään edelleen agenteille, jotka muuntavat syötteen loogiseen muotoonsa. Esimerkiksi puhesyötteen tulkinnassa agentti muuntaa raakatekstin luonnollisen kielen mukaisiksi merkityksiksi. Multimodaalisissa syötteissä käyttöliittymäagentti välittää yksittäisten modaliteettien analysoidut syötteet tämän jälkeen modaliteettien integrointi -agentille, jonka palauttama yhdistetty tulkinta syötteestä jatkaa välittäjäagentille.

5.1.2. Syötteen käsittely ja tulkinta-agentit

Jokaiselle eri modaliteetille on olemassa omat käsittely- ja tulkinta-agenttinsa, jotka ottavat vastaan vain oman modaliteettinsa kautta annettuja syötteitä. Ensiksi aktivoituu modaliteetin mukainen käsittelyagentti, esimerkiksi puheentunnistusagentti tai eleentunnistusagentti, jonka toiminnan tuloksena syntyy tyypillisesti lista parhaiten täsmäävistä tunnistustuloksista yhdistettynä timestamp-muotoisiin syötteen antamisaikoihin. Raakatulkinnat välitetään tämän jälkeen agenteille, jotka analysoivat yksittäisen syötteen, esimerkiksi luonnollisen kielen lauseen tai eleen, merkitystä syötteenantoajankohdan mukaisessa kontekstissa.

5.1.3. Syötteen integrointi -agentti

Syötteen integrointi -agentilla on keskeinen tehtävä multimodaalisissa käyttöliittymissä, ja sen toiminta vaikuttaa oleellisesti käyttöliittymän käytettävyyteen ja luotettavuuteen. Agentista käytetään englanninkielisissä teksteissä vaihtelevia nimityksiä: modality coordination agent [Moran et al., 1997] tai multimodal integration agent [Cohen et al., 1997; Kaiser et al., 2003]. Agentin tehtävä on kuitenkin nimestä huolimatta aina sama: samanaikaisten tai muuten yhteenkuuluvien syötteiden modaliteettikohtaisten tulkintojen yhdistäminen yhdeksi yhteiseksi multimodaaliseksi tulkinnaksi. Agentin vastuulla on viittausten ratkaiseminen (esimerkiksi pronominit "tuo", "ne", "tänne"), puuttuvien tietojen täydentäminen sekä monitulkintaisuuden ratkaiseminen kontekstin, vastaavuuden tai tarpeettomien syötteen osien huomiottajättämisen avulla.

5.1.4. Välittäjäagentti

Käytän tästä agentista nimitystä välittäjäagentti sen toiminnan luonteen mukaan (englanninkielisessä kirjallisuudessa agentista käytetään nimitystä facilitator agent). Välittäjäagentti saa syötteenään yksimodaalisesta syötteestä muodostetun tulkinnan tai multimodaalisen syötteen integroidun tulkinnan. Tämän jälkeen välittäjäagentti huolehtii tarvittavien tehtävien jakamisesta aktivoimalla muita agentteja, jotka vastaavat syötteen aiheuttaman toiminnon suorittamisesta ja palautteen antamisesta käyttäjälle. Moran et al. [1997] mainitsevat esimerkkinä kahden kohteen välisen etäisyyden kysymisen multimodaalisessa karttapohjaisessa käyttöliittymässä. Multimodaalisen syötteen yhdistetyn tulkinnan pohjalta välittäjäagentti aktivoi ensin kaksi tietokannanhallinta-agenttia, jotka etsivät haluttujen kohteiden sijainnit erillisistä tietokannoista. Nämä tulokset välittäjäagentti siirtää edelleen etäisyyden laskemisesta vastaavalle agentille, ja lopuksi aktivoi agentin, joka hoitaa tulosteiden ja palautteen antamisesta käyttäjälle.

5.2. Moniagenttiarkkitehtuurilla saavutetut edut

Moniagenttiarkkitehtuurin avulla voidaan vastata moniin multimodaalisille non-WIMP -käyttöliittymille tyypillisiin vaatimuksiin. Moran et al. [1997] ovat jakaneet OAA:n avulla saavutetut hyödyt seitsemään kategoriaan:

- *avoimuus:* agenttien toteutuksessa voidaan käyttää vaihtelevia ohjelmointikieliä ja alustoja,
- *jaettavuus:* agentit voivat toimia useilla eri alustoilla,
- *laajennettavuus:* uusia agentteja voidaan dynaamisesti lisätä tai poistaa. Lisättäessä ne tulevat välittömästi muiden agenttien saataville,
- *mobiilisuus:* sovelluksia voidaan käyttää myös kannettavissa laitteissa, joiden suorituskapasiteetti on alhainen (esim. mobiililaitteet, PDA), koska vain käyttöliittymä-agentin täytyy sijaita kyseisessä laitteessa. Käyttöliittymä-agentti hoitaa yhteydet muihin agentteihin, jotka voivat toimia jossakin toisessa alustassa,
- *Yhteistoiminnallisuus:* koska käyttäjä nähdään ikään kuin yhtenä agenttina, on arkkitehtuurin avulla helppo kehittää sellaisia käyttöliittymiä, joissa on useita samanaikaisia käyttäjiä,
- *Useat modaliteetit:* käyttöliittymä tukee useita modaliteetteja (käsiala, eleet, puhe, hiiri ja näppäimistö), ja

- *multimodaalinen vuorovaikutus:* käyttäjä voi antaa syötteensä yhdistämällä modaliteetteja.

Käyttöliittymän jakaminen moniin itsenäisiin agentteihin tarjoaa laajat mahdollisuudet agenttien uudelleenkäytettävyyteen eri sovelluksissa. Toisistaan riippumattomien agenttien toteutus on myös helppo jakaa usealle taholle, varsinkin kun agentit voidaan ohjelmoida eri kielillä ja alustoilla. Agentti voidaan myös helposti korvata uudella versiolla joko sovelluksen kehitysasteen mukaan tai saatavilla olevien järjestelmien kehittyessä. Järjestelmää voidaan testata jo varhaisessa vaiheessa esimerkiksi käyttämällä aluksi yksinkertaista luonnollisen puheen tunnistusjärjestelmää ja vaihtamalla toteutuksen myöhemmässä vaiheessa tilalle kehittyneempi tunnistusjärjestelmä [Moran *et al.*, 1997]. Monien modaliteettien, kuten luonnollisen puheen tai katseen, kohdalla uuden teknologian ja tutkimuksen aikaansaama kehitys aiheuttaa tarpeen vaihtaa agentin toteutusta.

Uusien agenttien lisääminen on helppoa, koska agentit ovat toisistaan riippumattomia. Tällä tavalla voidaan multimodaalisissa käyttöliittymissä vaivattomasti laajentaa tarjolla olevien syöte- tai tulostemodaliteettien valikoimaa. Tarve uusien modaliteettien lisäämiselle voi tulla joko uudenlaisen teknologian kehittyessä, esimerkiksi katseella tai vartalonliikkeillä annettujen syötetapojen yleistyessä, tai jos käyttöliittymän tulee sopeutua uudenlaisiin käyttötilanteisiin tai käyttäjiin. Koska agentteja voidaan lisätä ja poistaa myös dynaamisesti järjestelmän ajonaikana, voidaan käyttöliittymä helposti ja joustavasti sopeuttaa kulloiseenkin toimintaympäristöönsä. Käytettävissä olevien modaliteettien valikoima saattaa olla rajoittunut niin teknisten puutteiden ja ongelmien vuoksi kuin käyttöympäristön tai syötteen laadun vaikutuksesta [Moran *et al.*, 1997].

Yksi multimodaalisten non-WIMP-käyttöliittymien sovellusalueista on pienet kannettavat laitteet, kuten mobiililaitteet, kämmenmikrot sekä kannettavat pelikonsolit ja mediasoittimet. Moniagenttiarkkitehtuurin jaettavuus mahdollistaa tällaisten laitteiden suorituskapasiteetin sekä virrankulutuksen optimoimisen. Ainoastaan käyttöliittymä-agentin täytyy sijaita käyttäjän päätelaitteessa ja loput agentit, jotka vaativat enemmän suoritustehoa ja muistikapasiteettia, voivat sijaita muilla koneilla tai palvelimilla. Tämä mahdollistaa myös sen, että agentteja voidaan samanaikaisesti jakaa useiden eri sovellusten käyttöön. Lisäksi jaettua sovellusta on mahdollista käyttää, vaikka

käyttöympäristössä tapahtuisi muutoksia, kuten alustan vaihtuminen siirryttäessä käyttämään toista laitetta tai syöte- ja tulostelaitevalikoiman muutokset. Tällaisissa tapauksissa sovelluksen toimivuus vaatii ainoastaan kyseiseen laiteympäristöön soveltuvan käyttöliittymä-agentin ja kaikki muut agentit säilyvät muuttumattomina.

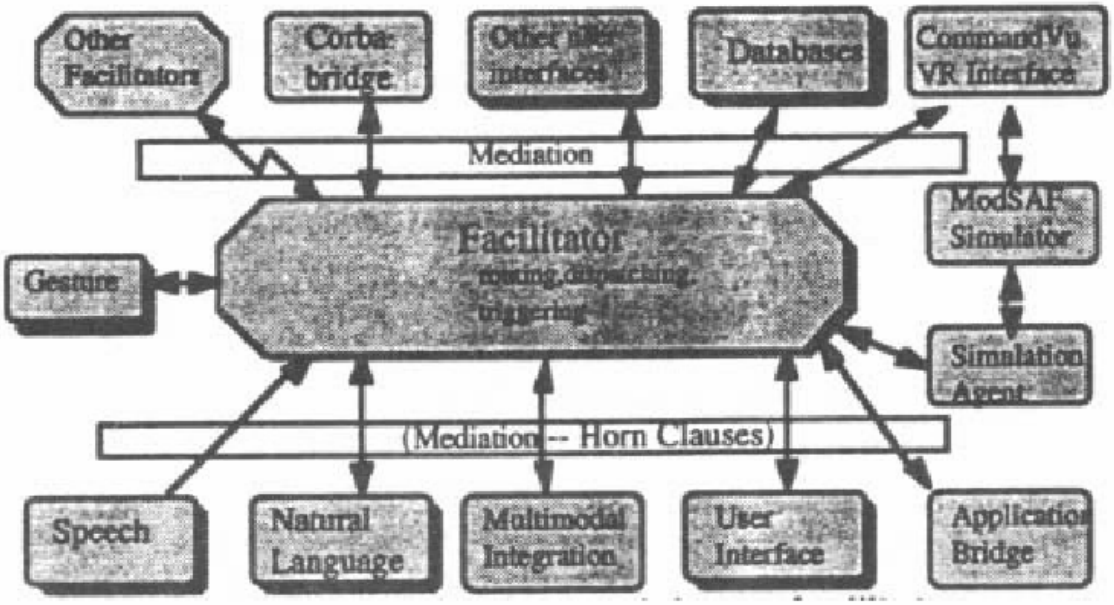
Moniagenttiarkkitehtuurin avulla voidaan tehokkaasti ja nopeasti käsitellä monimutkaisia multimodaalisia syötteitä. Koska jokaiselle modaliteetille on olemassa omat agenttinsa syötteen käsittelyyn ja tulkintaan, voidaan syötteen käsittelyprosessi jakaa pieniin osiin, joiden suorittaminen voi olla samanaikaista sekä ajankäytöltään optimoitua. Tämä edesauttaa sitä, että suuresta ja monimutkaisesta syöteinformaatiosta huolimatta, käyttöliittymä pystyy vastaamaan reaaliaikaisuuden vaatimukseen. Ongelman saattaa kuitenkin muodostaa välittäjäagentti, joka on arkkitehtuurin eräänlainen pullonkaula. Jos käyttöliittymässä joudutaan jatkuvasti välittämään suuria tietomääriä välittäjäagentin kautta, voi tämä vaikuttaa käyttöliittymän reaaliaikaisuuteen. Tästä syystä sovelluksissa, esimerkiksi virtuaalitodellisuussovelluksissa, joissa käsitellään jatkuvasti valtavia tietomääriä ja samanaikaisesti kohdataan tiukat reaaliaikaisuusvaatimukset, saattaa olla tarpeellista rakentaa arkkitehtuuri hieman väljemmäksi välittäjäagentin osalta. Tämä tarkoittaa sitä, että agenteille annetaan vapaus kommunikoida myös suoraan keskenään. Esimerkiksi puhesyötteen käsittely -agentti siirtää tulosteensa suoraan tulkinta-agentille ilman välittäjäagentin kautta kierrättämistä.

5.3. QuickSet

Cohen *et al.* [1997] esittelevät Quickset-järjestelmän, joka on alunperin suunniteltu toimimaan yhdessä Yhdysvaltain armeijan harjoitussimulaattori-järjestelmän kanssa. Käyttöliittymä on multimodaalinen ja tarkoitettu jaettuihin interaktiivisiin simulaatioihin. Varsinainen Quickset-järjestelmä toimii kädessä pidettävissä laitteissa, mutta se voidaan liittää myös suureen virtuaaliseen näyttöön. Käyttäjä voi antaa syötteitä samanaikaisesti sekä puheen että kynällä tuotettujen eleiden avulla.

Kuvassa 2 on kuvattu Quickset-järjestelmässä käytetty OAA-pohjainen arkkitehtuuri, joka soveltuu suurilta osin yleistettäväksi muihinkin kannettavissa laitteissa käytettäviin multimodaalisiin käyttöliittymiin. Keskeisessä asemassa arkkitehtuurissa on välittäjäagentti (*facilitator agent*), joka aktivoi muita agentteja ja välittää näille tehtäviä. Välittäjäagentin käyttäminen mahdollistaa helpon

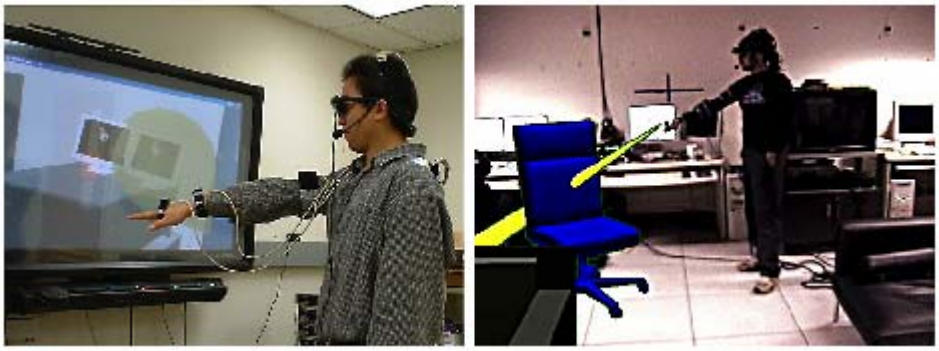
laajennettavuuden, uudelleenkäytettävyyden, jaettavuuden, agenttien itsenäisen kehittämisen sekä monimutkaisten syötteiden hallitsemisen. Arkkitehtuurin ongelmana on kuitenkin välittäjäagentin kuormittuminen, jos sen kautta välittyy jatkuvasti suuria määriä tietoa. Tällainen tilanne tulee vastaan esimerkiksi virtuaalitodellisuussovelluksissa, joissa syöteinformaatiota otetaan vastaan useita kymmeniä kertoja sekunnissa, useista erityyppisistä laitteista ja samalla käyttöliittymän pitäisi pystyä tuottamaan jatkuvaa, reaaliaikaista palautetta käyttäjälle.



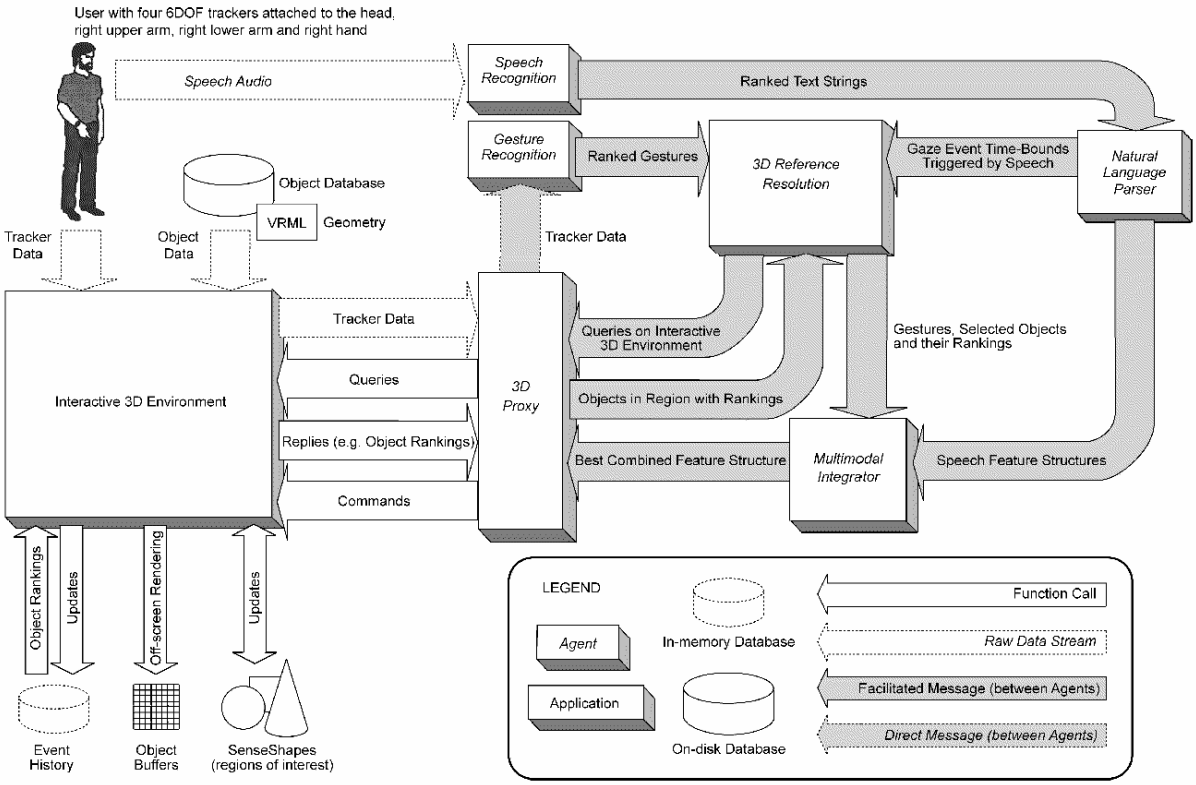
Kuva 2: Quickset-sovelluksen arkkitehtuuri [Cohen *et al.*, 1997].

5.4. Kolmiulotteinen virtuaalitodellisuus ja lisätty todellisuus -sovellus

Kaiser et al. [2003] ovat kehittäneet moniagenttisuuteen pohjautuvan arkkitehtuurin, joka on tarkoitettu kolmiulotteista, multimodaalista vuorovaikutusta sisältäviin virtuaalitodellisuus ja lisätty todellisuus (*augmented reality*) -sovelluksiin. Käyttäjä voi olla vuorovaikutuksessa järjestelmän kanssa yhdistämällä puhesyötteensä joko kädellä tehtävään elesyötteeseen tai pään asentoon perustuvaan katsesyötteeseen. Kuvassa 3 on esimerkki suoritetusta koejärjestelystä, jonka perusteella tutkittiin kehitetyn arkkitehtuurin toimivuutta. Itse arkkitehtuuri on esitetty kuvassa 4.



Kuva 3: Esimerkki koejärjestelystä, jossa testattiin kuvassa 4 esitetyn arkkitehtuurin toimivuutta sekä virtuaalisessa että lisätyn todellisuuden ympäristössä [Kaiser *et al.*, 2003].



Kuva 4: Moniagenttiarkkitehtuuri multimodaalista, kolmiulotteista vuorovaikutusta sisältäviin käyttöliittymiin virtuaali- ja lisätty todellisuus -sovelluksissa käytettäväksi [Kaiser *et al.* , 2003].

Arkkitehtuurissa ei ole käytössä varsinaista välittäjäagenttia, vaan erilliset agentit pystyvät kommunikoimaan suoraan toistensa kanssa. Tällainen ratkaisu tukee käyttöliittymätyypille luontaisen jatkuvan ja valtavan syöteinformaation käsittelyä sekä jatkuvan reaaliaikaisen palautteen tuottamista. Tämänkaltaisessa arkkitehtuuriratkaisussa joudutaan kuitenkin tekemään kompromisseja agenttien erillisyyden ja näin ollen myös uudelleenkäytettävyyden kustannuksella.

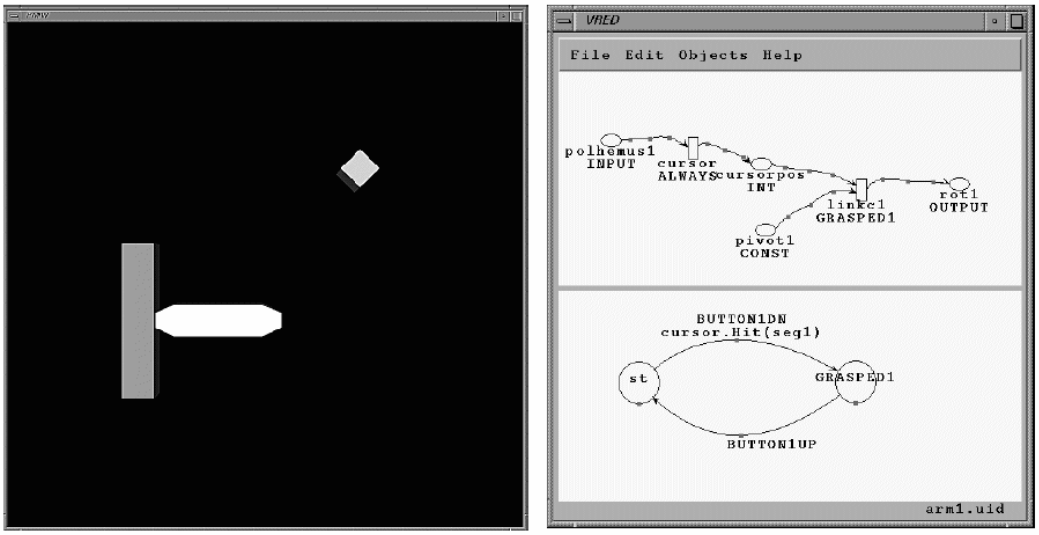
6. Kaksijakoisen vuorovaikutuksen malli

Jacob et al. [1999] ovat kehittäneet non-WIMP-tyylisen käyttöliittymän kuvaamiseen ja ohjelmointiin soveltuvan mallin, jossa otetaan huomioon tällaisille käyttöliittymille tyypillisen vuorovaikutuksen asettamat vaatimukset. Heidän mielestään vuorovaikutuksen ydin ovat jatkuvat vuorovaikutussuhteet, joista suurin osa on vain hetkellisiä. Malli jakaa käyttäjän ja tietokoneen välisen vuorovaikutuksen kahteen keskenään kommunikoivaan komponenttiin. Toinen komponenteista käsittelee jatkuvia vuorovaikutussuhteita tietovuo-tyyppisesti, kun taas toinen tapahtumiin perustuva komponentti käsittelee erillisiä vuorovaikutustapahtumia. Jälkimmäisen komponentin tehtävänä on ohjelman tilasta riippuen mahdollistaa tai estää yksittäisiä jatkuvaan vuorovaikutukseen perustuvia vuorovaikutussuhteita. Malli mahdollistaa myös samanaikaisen vuorovaikutuksen. Lisäksi malli on riippumaton konkreettisista syöte- ja tulostelaitteista, koska se ei ota kantaa niiden erityisominaisuuksiin. Tämä helpottaa uusien modaliteettien lisäämistä.

Malli jakaantuu kolmeen olio-ohjelmoinnin mukaiseen luokkaan: jatkuvaa vuorovaikutusta käsitteleviin muuttujiin ja linkkeihin sekä erillisiin syötteisiin vastaaviin tapahtumankäsittelijöihin. Jatkuviin vuorovaikutussuhteisiin perustuvat muuttujat tallettavat arvoja, jotka liittyvät saatuihin syötteisiin, annettaviin tulosteisiin, mallin sisäiseen toimintaan tai väliaikaisten tulosten säilyttämiseen. Näitä muuttujia voidaan yhdistellä toisiinsa toiminnallisuutta sisältävillä linkeillä, jotka voivat olla joko aktiivisia koko ohjelman suorittamisen ajan, tai joiden aktivointia voidaan säädellä tilanteen mukaan. Säätely tapahtuu asettamalla linkille jokin käyttäjän antamaan syötteeseen perustuva tilanne-ehto, jonka mukaan linkki joko aktivoidaan tai sen toiminta estetään. Tämä säätelymahdollisuus on yksi mallin keskeisimmistä piirteistä, sillä sen avulla

voidaan määritellä tilanteen mukaan, mitkä jatkuvat vuorovaikutussuhteet ovat kulloinkin voimassa.

Mallin pohjalta toteutettiin kuvauskieli sekä visuaalinen VRED-editori sen tuottamiseen. Lisäksi PMIW-nimisen käyttöliittymän hallintajärjestelmän toteuttamisella todennettiin, että malli ei heikennä reaaliaikaisuutta. Kuvassa 5 on kuvattuna yksinkertainen esimerkki liikuteltavasta kädestä, johon käyttäjä voi tarttua ja tämän jälkeen liikutella kolmiulotteisesti. Lisäksi esitetään vastaavan käden toiminnan määrittely malliin perustuen graafisesti VRED-editorissa.



Kuva 5: Esimerkki mallin mukaisesta vuorovaikutuksen kaksijakoisuudesta [Jacob et al., 1999].

7. Yhteenveto

Multimodaalisten non-WIMP-käyttöliittymien hyödyntäminen lisääntyy jatkuvasti teknologisen kehityksen ja uudenlaisten käyttötarpeiden myötä. Ennen ihmiset olivat käyttöliittymän kanssa tekemisissä pääasiassa vain työpöytänsä ääressä, mutta nykyisin ihmiset kohtaavat käyttöliittymiä yhä erilaisimmissa yhteyksissä, esimerkiksi mobiililaitteissa ja sulautetuissa järjestelmissä. Tulevaisuudessa myös virtuaalitodellisuussovellukset saattavat hyvinkin tulla kaikkien ihmisten ulottuville. Tällaisten käyttöliittymien vuorovaikutus eroa suuresti tavallisissa pöytätietokoneissa käytettyjen WIMP-käyttöliittymien vuorovaikutuksesta. Käyttäjälle tarjotaan ensinnäkin uudenlaisia syöte- ja tulostemodaliteetteja sekä mahdollisuus niiden yhdistämiseen. Toisaalta vuorovaikutus ei ole enää vuoroittaista, yksittäisten ja selkeiden komentojen antamista, vaan käyttöliittymän on pystyttävä

vastaanottamaan jatkuvia, samanaikaisia ja osittain passiivisesti annettuja syötteitä. Myös tulosteiden antamiseen käytettyjen modaliteettien kirjo on usein suurempi, ja niitä voidaan muodostaa yhdistämällä useampia modaliteetteja. Lisäksi tulosteiden antamisen tulisi olla jatkuvaa ja mahdollisimman reaaliaikaista sovellustyypistä riippuen.

Multimodaaliset non-WIMP-käyttöliittymät ovat vuorovaikutukseltaan ja laitteistoltaan huomattavasti monimutkaisempia perinteisiin WIMP-käyttöliittymiin verrattuna. Tämän vuoksi arkkitehtuurin suunnittelu nousee huomattavan tärkeään rooliin sovelluksen toimivuuden ja käytettävyyden takaamiseksi. Edellä esitettyjen vuorovaikutuksen erityispiirteiden ja vaatimusten vuoksi WIMP-käyttöliittymiin kehitellyt, toimivat mallit ja arkkitehtuurit eivät sovellu käytettäväksi multimodaalisissa non-WIMP-käyttöliittymissä. Niitä varten tuleekin suunnitella uudenlaisia arkkitehtuuriratkaisuja, suunnittelumalleja sekä määrittelykieliä. Keskityin tutkimuksessani esittelemään moniagenttiarkkitehtuurin, joka sopii mielestäni non-WIMP-käyttöliittymien kuvaamiseen erityisen hyvin joustavuutensa, jaettavuutensa sekä multimodaalisuutensa vuoksi. Arkkitehtuurissa otetaan kattavasti huomioon kaikki non-WIMP -käyttöliittymien keskeiset vaatimukset ja ongelmalliset erityispiirteet.

Multimodaalisten non-WIMP-käyttöliittymien kehittely ja hyödyntäminen on kuitenkin vasta alussa, joten ajan myötä nykyiset arkkitehtuuriratkaisut joutuvat haasteen eteen. Pystyvätkö ne osoittamaan toimivuutensa entistä kehittyneemmissä ja monimutkaisemmissa käyttöliittymissä? Toinen tulevaisuuden haaste tulee olemaan laajasti käytettävien ja toimivien arkkitehtuureiden standardointi, jotta myös non-WIMP-käyttöliittymien pariin saataisiin yhtenevät suunnittelumallit ja -käytännöt.

Viiteluettelo

[Bolt, 1980] Richard Bolt, Put that there: Voice and gesture at the graphics interface. *ACM Computer Graphics* **14**, 3 (1980), 262–270.

[Cohen *et al.*, 1997] Cohen, P. R., Johnston, M., McGee, D., Smith, I., Oviatt, S., Pittman, J., Chen, L., and Clow, J. QuickSet: multimodal interaction for simulation set-up and control. In: *Proceedings of the fifth conference on Applied natural language processing* (1997), 20 - 24. Also available as

www.cse.ogi.edu/CHCC/Publications/quickset_multimodal_interaction_simulation_setup_and_control_cohen.pdf

[Green and Jacob, 1991] Mark Green and Robert Jacob, Software Architectures and Metaphors for Non-WIMP User Interfaces. *ACM SIGGRAPH Computer Graphics* **25**, 3 (July 1991), 229 - 235.

[Jacob *et al.*, 1999] Robert J. K. Jacob, Leonidas Deligiannidis, and Stephen Morrison, A software model and specification language for non-WIMP user interfaces. *ACM Transactions on Computer-Human Interaction* **6**, 1 (March 1999), 1 - 46.

[Jain and Ross, 2004] Anil K. Jain and Arun Ross, Multibiometric systems. *Communications of the ACM* **47**, 1 (Jan. 2004), 34-40.

[Kaiser et al., 2003] Ed Kaiser, Alex Olwal, David McGee, Hrvoje Benko, Andrea Corradini, Xiaoguang Li, Phil Cohen, and Steven Feiner, Mutual disambiguation of 3D multimodal interaction in augmented and virtual reality. In: *Proceedings of the 5th international conference on Multimodal interfaces* (2003), 12 - 19. Also available http://www.cse.ogi.edu/CHCC/Publications/Mutual_Disambiguation_of_3D_Multimodal_Interaction_in_Augmented_and_Virtual_Reality.pdf

[Maybury and Wahlster, 1998] Mark T. Maybury and Wolfgang Wahlster (eds.), Readings in Intelligent User Interfaces. Morgan Kaufmann Publishers, 1998.

[Moran et al., 1997] Douglas B. Moran, Adam J. Cheyer, Luc E. Julia, David L. Martin, and Sangkyu Park**,** Multimodal user interfaces in the Open Agent Architecture. In: *Proceedings of the 2nd international conference on Intelligent user interfaces* (1997), 61 - 68.

[Nigay and Coutaz, 1993] Laurence Nigay and Joelle Coutaz, A design space for multimodal systems: concurrent processing and data fusion. In: *Proceedings INTERCHI'93*, 172-178.

[Oviatt, 1999] Sharon Oviatt, Ten myths of multimodal interaction. *Communications of the ACM* **42**, 11 (Nov. 1999), 74-81.

[Oviatt *et al.*, 2004] Sharon Oviatt, Rachel Coulston, and Rebecca Lunsford, When Do We Interact Multimodally? Cognitive load and multimodal communication patterns. In: *Proceedings of the 6th ICMI* (2004), 129-136.

[Oviatt *et al.*, 1997] Sharon Oviatt, Antonella DeAngeli, and Karen Kuhn, Integration and synchronization of input modes during multimodal human-computer interaction. In: *Proceedings of the SIGCHI conference on Human factors in computing systems* (1997), 415-422.

[Schomaker *et al.*, 1995] Schomaker et al., A taxonomy of multimodal interaction in the human information processing system. *Report of the Esprit Project 8579 MIAMI (1995),* Available as http://hwr.nici.kun.nl/~miami/taxonomy/taxonomy.html (23.5.2005).

[Shaer and Jacob, 2005] Orit Shaer and Robert Jacob, Toward a Software Model and a Specification Language for Next-Generation User Interfaces. *ACM CHI 2005 Workshop* (April 2005). Available as http://hci.stanford.edu/srk/chi05-ui-tools/Shaer.pdf (25.10.2005).

Käsinkosketeltavien käyttöliittymien arkkitehtuuriratkaisut

Antti Nyman

Tiivistelmä.
Käsinkosketeltavat käyttöliittymät hyödyntävät ihmisten taitoja käsitellä jokapäiväisiä esineitä ja ovat osa pyrkimystä siirtyä kohti intuitiivisempia käyttöliittymiä. Uusi vuorovaikutustapa asettaa arkkitehtuureille kuitenkin vaatimuksia, joita aikaisemmin käytetyt ratkaisumallit eivät täysin täytä. Uudenlaisten ratkaisutapojen löytämiseksi esittelen käsinkosketeltavien käyttöliittymien toteutuksissa käytettyjä arkkitehtuureja. Vertailin viittä erilaista arkkitehtuuria keskenään ja havaitsin niissä käytettyjen ratkaisujen eroavan toisistaan. Mikään arkkitehtuuri ei sisällä vastauksia kaikkiin esitettyihin ongelmiin, mutta eri ratkaisut soveltuvat toisistaan eroaviin tarkoituksiin ja niitä yhdistelemällä on mahdollista kehittää yhä parempia arkkitehtuureja.

Avainsanat ja -sanonnat: Ohjelmistoarkkitehtuuri, käsinkosketeltava käyttöliittymä.
CR-luokat: D.2.11, H.5.2

1. Johdanto

Käsinkosketeltavissa käyttöliittymissä (tangible user interface, TUI) ihmisen ja tietokoneen vuorovaikutus perustuu aineellisiin, kosketeltaviin asioihin ja esineisiin. Niissä käyttäjillä on Ishiin ja Ullmerin [1997] esittämän vision mukaisesti mahdollisuus olla vuorovaikutuksessa tietokoneen kanssa tarttumalla käyttöliittymän osana oleviin jokapäiväisiin esineisiin tai arkkitehtonisiin pintoihin ja käsittelemällä niitä. Näihin aineellisiin objekteihin voidaan tietokoneen avulla liittää virtuaalisia ominaisuuksia, mikä puolestaan mahdollistaa digitaalisen informaation esittämisen ja muokkaamisen objektien avulla. Rakennuksen pienoismallia siirtämällä voidaan esimerkiksi siirtää vastaavaa virtuaalista rakennusta. Oleellista on, että aineelliset objektit ovat yhtäaikaisesti sekä esitys, joka vastaa virtuaalista informaatiota että ohjain, jonka avulla informaatiota voidaan muokata. Niissä yhdistyvät siis sekä syöte- että tulostuskanavan ominaisuudet. Tämä erottaa käsinkosketeltavat käyttöliittymät

esimerkiksi hiirestä, joka sekin on yhtä lailla aineellinen ja kosketeltava, mutta ei mahdollista informaation esittämistä sijaintinsa tai ominaisuuksiensa kautta.

Ihmisten luonnollisia taitoja käsitellä aineellisia objekteja hyödynnetään nykyisissä käyttöliittymissä vain vähän. Esineiden havainnointiin ja käsittelemiseen liittyviä kykyjä ei voida käyttää hyväksi graafisissa käyttöliittymissä (graphical user interface, GUI) lainkaan niin hyvin, kuin niitä voidaan hyödyntää aineellisia objekteja käsiteltäessä. Ishiin ja Ullmerin [1997] mukaan nykyisten yleiskäyttöisten tietokoneiden ja yksipuolisten vuorovaikutustapojen myötä onkin hävinnyt osa siitä rikkaudesta, joka liittyy ihmisten luonnollisiin tapoihin käyttää esineitä ja työkaluja.

Käsinkosketeltavat käyttöliittymät ovat osa pyrkimystä siirtyä eteenpäin ikkuna, ikoni, valikko ja osoitin -vuorovaikutusmallista (window, icon, menu and pointer, WIMP) kohti intuitiivisempia ja ihmisen luonnollisia kykyjä paremmin hyödyntäviä käyttöliittymiä. Siinä missä aikaisemmat käyttöliittymät ovat hyödyntäneet pääosin yhtä syöte- ja yhtä tulostuskanavaa sekä peräkkäistä ja vuoropohjaista vuorovaikutusta, uudet käyttöliittymät hyödyntävät jatkuvaa vuorovaikutusta, jossa on käytössä useita rinnakkaisia ja tahdistamattomia kanavia [Jacob *et al.*, 1999]. Shaer [2005] on käsinkosketeltavien käyttöliittymien hallintajärjestelmää kehittäessään havainnut uudenlaisten vuorovaikutustapojen suurimmiksi eroiksi vanhoihin verrattuna juuri useat samanaikaiset syötteet yhdeltä tai usealta käyttäjältä, rinnakkaisen fyysisen ja digitaalisen tulostuskanavan, jatkuvat (continuous) ja erilliset (discrete) syötteet sekä standardien puutteet syöte- ja tulostuslaitteissa. Nämä seikat eivät ainoastaan tee käyttöliittymien käyttämisestä erilaista, vaan myös asettavat uudenlaisia vaatimuksia niiden kehittämiselle. Vanhat arkkitehtuuriratkaisut eivät enää vastaa uusia vaatimuksia niin hyvin, että niiden käyttäminen olisi järkevää. Tarvitaan siis uudenlaisia ratkaisuja, jotka mahdollistavat uusien vuorovaikutustapojen tehokkaan ja erityisesti käyttäjien mielikuvia vastaavan hyödyntämisen.

Uudenlaisia vuorovaikutustapoja hyödyntävien käyttöliittymien toteutusta tulisi myös pyrkiä helpottamaan luomalla työkaluja, jotka siirtävät kehitystyötä korkeammalle abstraktiotasolle. Työkalut voivat parantaa käyttöliittymä-ohjelmoijien työn laatua ja tuottavuutta esimerkiksi lisäämällä yhdenmukaisuutta käyttöliittymien välillä, tukemalla laajennettavuutta ja piilottamalla ympäristöstä riippuvaa monimutkaisuutta [Betts *et al.*, 1987]. Vaikka yleiskäyttöisiä tuoterunkoarkkitehtuureja ei käsinkosketeltaville

käyttöliittymille vielä lähiaikoina onnistuttaisikaan toteuttamaan, voidaan uudelleenkäyttöä lisätä hyvin tarkoitukseensa sopivilla arkkitehtuureilla, joita voidaan käyttää uudelleen muiden sovellusten toteutuksissa.

Seuraavassa luvussa esittelen käsinkosketeltavien käyttöliittymien toteutuksissa käytettyjä arkkitehtuuriratkaisuja tarkoituksenani antaa yleiskuva aihealueesta. Kolmannessa luvussa vertailen ratkaisuja keskenään ja lopuksi esitän yhteenvedon.

2. Käsinkosketeltavien käyttöliittymien arkkitehtuuriratkaisuja

Tässä luvussa kuvailen viisi erilaista käsinkosketeltavien käyttöliittymien arkkitehtuuria tai arkkitehtuurissa käytettyä ratkaisua, joiden taustalla ovat erilaiset ongelmat. Ratkaisuista on kuitenkin löydettävissä lukuisia yhtäläisyyksiä, mikä saattaa osaltaan helpottaa hyvien toimintatapojen havaitsemista.

Ensimmäisessä kohdassa esittelen ohjelmistokehyksen, joka mahdollistaa käsinkosketeltavien käyttöliittymien lisäksi myös useiden muiden vuorovaikutustapojen käyttämisen. Sen jälkeen kuvaan erityisesti käsinkosketeltavia käyttöliittymiä varten kehitetyn alusta-arkkitehtuurin, joka toteuttaa useita käsinkosketeltaviin käyttöliittymiin liittyviä teorioita käytännössä. Kolmannessa kohdassa esiteltävä työkalukokoelma on tarkoitettu käsinkosketeltavien käyttöliittymien kehittämiseen ja erityisesti helpottamaan syötelaitteiden kanssa työskentelyä. Kohdissa neljä ja viisi kuvattavat ratkaisut ovat suunniteltu helpottamaan fyysisten syötelaitteiden liittämistä osaksi sovelluksia. Ensimmäinen niistä on alun perin tarkoitettu jokapaikan tietotekniikan (ubiquitous computing) sovelluksiin, mutta soveltuu myös käsinkosketeltavissa käyttöliittymissä käytettäväksi. Toinen tarjoaa kehittäjien käyttöön valmiita laitekomponentteja sovelluksiin liitettäviksi.

2.1. DWARF

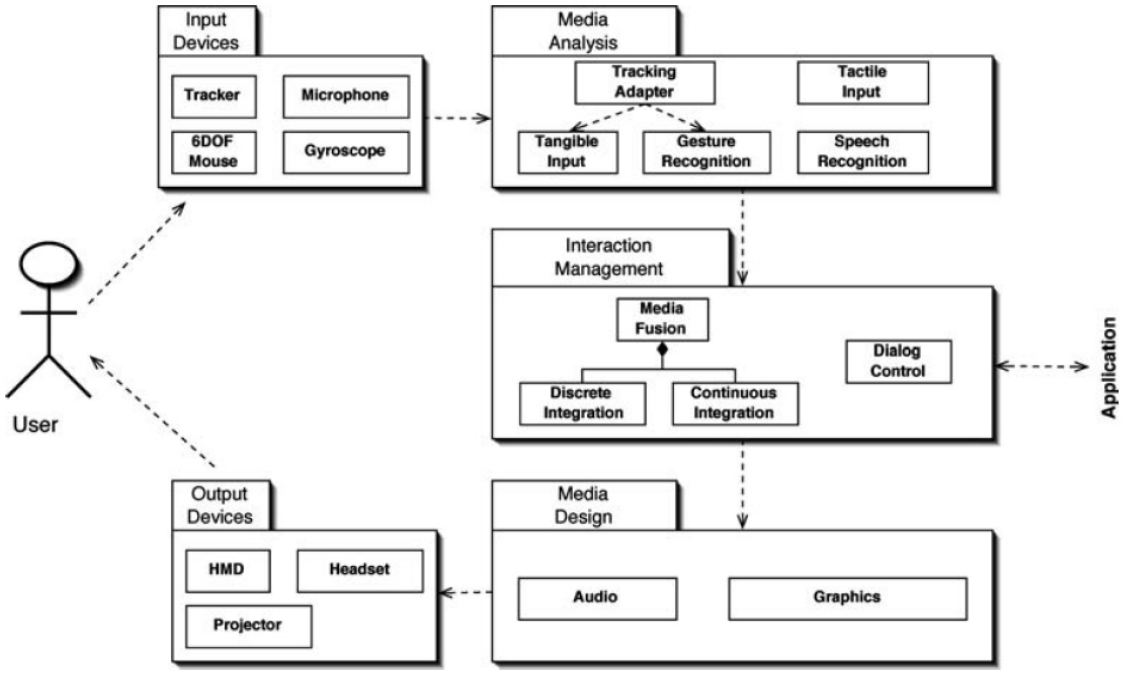
DWARF (distributed wearable augmented reality framework) [Sandor and Klinker, 2005] on useita uusia vuorovaikutustapoja ja niiden nopeaa vaihtamista tukeva ohjelmistokehys [1]. Sandor ja Klinker kutsuvat tutkimaansa uusien käyttöliittymien aluetta jokapaikan lisätyksi todellisuudeksi (ubiquitous

[1] Tietoa ohjelmistokehyksen saatavuudesta on projektin kotisivulla osoitteessa http://campar.in.tum.de/Chair/ProjectDwarf.

augmented reality) ja sanovat siihen kuuluvan multimediaa, moniaistisuutta (multimodality), lisättyä todellisuutta (augmented reality) sekä puettavaa (wearable) ja jokapaikan tietotekniikkaa. Arkkitehtuuria käyttäen voidaan muiden vuorovaikutustapojen lisäksi toteuttaa myös käsinkosketeltavien käyttöliittymien piirteitä.

DWARF-kehyksen suunnittelu perustuu ongelma-alueen kartoitukselle. Eräs suunnitteluperusteista on resurssien liikkuvuus (mobility), josta aiheutuvaa joustavuutta tavoitellaan kehyksen modulaarisuuden avulla. Toinen suunnitteluperuste on moniaistisuuden järjestelmältä vaatima useiden kommunikointikanavien hyödyntäminen. Käsinkosketeltavien käyttöliittymien kannalta mielenkiintoisin suunnitteluperuste on aineellisten ja virtuaalisten objektien vuorovaikutuksen hallinta, jonka tekee mahdolliseksi arkkitehtuurin toteuttama aineellisten objektien jatkuva jäljittäminen (tracking).

Kehyksen päälle on mahdollista rakentaa monenlaisia sovelluksia. Sandor ja Klinker [2005] esittelevät tutkimuksessaan neljä kehyksen päälle rakennettua sovellusta, jotka kaikki toteuttavat käsinkosketeltavien käyttöliittymien piirteitä. Käytettäviä ominaisuuksia ovat esimerkiksi objektien tunnistaminen ja paikannus, olemassa olevien työkalujen hyödyntäminen sekä tarttumisen mahdollistavien linssien käyttö. Nämä ominaisuudet osoittavat, että huolimatta tuestaan muille vuorovaikutustavoille, tukee kehys myös käsinkosketeltavia käyttöliittymiä.



Kuva 1. DWARF arkkitehtuuri. [Sandor and Klinker, 2005]

Kuvassa 1 on esitetty kehyksen arkkitehtuuri, joka koostuu viidestä alijärjestelmästä (syötelaitteet, media-analyysi, vuorovaikutuksen hallinta, mediasuunnittelu ja tulostuslaitteet). Jokainen niistä voi sisältää erilaisia komponentteja kulloinkin toteutettavasta järjestelmästä riippuen. Alijärjestelmät toimivat pääosin itsenäisesti muodostamalla omasta syötteestään tulosteen.

Syötelaitteita voi olla järjestelmässä useita ja niiden avulla voidaan toteuttaa moniaistisia järjestelmiä. Myös tulostuslaitteita voi olla käytössä samanaikaisesti useita, jolloin voidaan hyödyntää multimedian tarjoamia mahdollisuuksia.

Media-analyysin tehtävä on muuntaa syötelaitteilta saamansa käyttäjän antamat syötteet abstrakteiksi, järjestelmän sisäisiksi ilmauksiksi (token). Samassa alijärjestelmässä tapahtuu myös aineellisten syötteiden analysointi, joka voi tarkoittaa esimerkiksi aineellisten objektien paikannusta. Tulostuspuolella vastaava alijärjestelmä on mediasuunnittelu, jossa puolestaan valitaan informaatiota esitettäväksi käyttäjälle vaikkapa äänen tai kuvan muodossa.

Kehyksen sydän on vuorovaikutuksen hallinta, joka yhdistää eri syötelaitteilta tulevat ilmaukset ja päättelee niistä käyttäjän aikomukset. Ilmaukset voivat olla *jatkuvia* (continuous), eli saada mitä tahansa lukuarvoja, tai *erillisiä* (discrete), eli saada vain tiettyjä ennalta määrättyjä arvoja. Esineen kiertoa ilmaiseva astemäärä on esimerkki jatkuvasta ilmauksesta, kun taas

puheentunnistuksen tunnistama sana on erillinen ilmaus. Jatkuvat ilmaukset voivat kuulua rajatulle välille (rotation) tai saada mitä tahansa arvoja (translation). Erilliset ilmaukset taas voivat olla totuusarvo- (boolean) tai merkkijonotyyppisiä (string).

Saamistaan ilmauksista vuorovaikutuksen hallinta päättelee käyttäjälle näytettävän tulostuksen ja välittää sen eteenpäin komento-suunnittelumallia (command design pattern) [Gamma *et al.*, 1995] käyttäen. Multimodaalinen vuorovaikutus on toteutettu Petri-verkkojen (Petri net) avulla. Järjestelmän törmäystunnistus (collision detection) tunnistaa virtuaalisten objektien, todellisten objektien sekä virtuaalisten ja todellisten objektien törmäykset. Jäljitystieto voidaan välittää joko suoraan jäljityskomponenteilta näyttökomponenteille tai käsitellä ensin putki-suodatin-suunnittelumallia (pipes and filters design pattern) [Buschmann *et al.*, 1996] käyttäen.

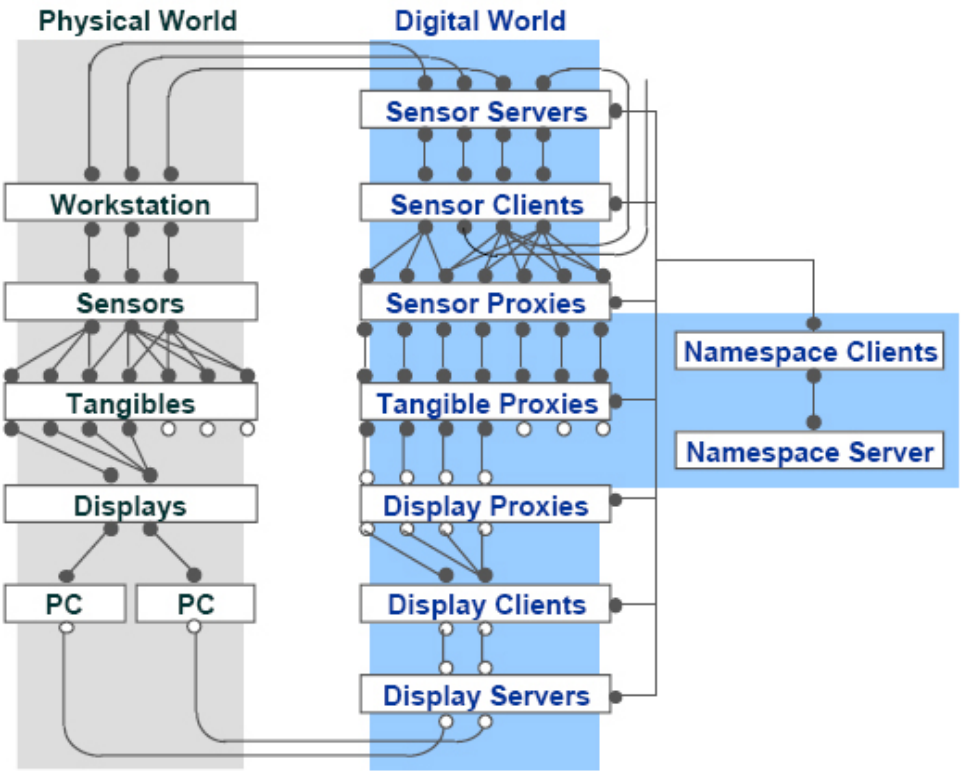
2.2. metaDESK

metaDESK [Ullmer and Ishii, 1997] on käyttöliittymäalusta, joka toteuttaa tekijöidensä visiota [Ishii and Ullmer, 1997]. Siinä suunnittelun lähtökohtana on ollut tuoda tietokoneen työpöydältä tuttuja metaforia ihmisille tuttuun aineelliseen todellisuuteen. Aineelliset ikonit, työpöydän ikkunoita vastaavat linssit ja muut aineelliseen todellisuuteen tuodut graafisen käyttöliittymän elementit ovat alustassa käyttäjien kosketeltavissa ja käsiteltävissä. Niiden kautta voidaan myös tuoda aineelliseen todellisuuteen monia tietokoneen mahdollistamia lisäyksiä rikastuttamaan toimintaympäristöämme.

Arkkitehtuuri tukee vision metaforia antamalla kaikille aineellisille objekteille kyvyt tunnistaa oma tilansa, kommunikoida muiden objektien kanssa ja mahdollisesti näyttää tulosteita. Vain osassa objekteista sijaitsee todellisuudessa sensoreita, joilla edellä mainitut kyvyt voidaan itsenäisesti toteuttaa. Loput objekteista ovat täysin passiivisia, eivätkä sisällä mitään elektroniikkaa. Tästä johtuen objektit ovat sovelluksen näkökulmasta erilaisessa asemassa. Ongelmaksi toteutuksessa tuli vaikeus säilyttää halutut vuorovaikutustapaan liittyvät metaforat ja samalla hallita järjestelmän monimutkaisuus. Ratkaisuksi tähän Ullmer ja Ishii [1997] kehittivät hajautettuihin edustajiin (proxy-distributed, proxdist) perustuvan arkkitehtuurin sekä alustariippumattoman meta-kielen [Ullmer, 1997], joka tarjoaa aineellisia objekteja vastaavat edustajaluokat.

Alustan fyysinen arkkitehtuuri koostuu useista esineistä ja laitteista. Alhaalta heijastettava, lähes vaakasuora taso toimii työpöytänä koko järjestelmälle. Sen yläpuolella varren varassa liikkuva aktiivinen linssi tarjoaa vaihtoehtoisen näkymän sovelluksen informaatioon. Passiivista linssiä ja aineellisia objekteja käyttäjä voi liikutella vapaasti tason pinnalla. Mainittujen osien lisäksi järjestelmän fyysiseen arkkitehtuuriin kuuluvat objektien sijainteja aistivat sensorit sekä tietokoneet, joille alusta on hajautettu.

Proxdist-arkkitehtuuri mahdollistaa tarvittavien kykyjen toteuttamisen passiivisten objektien puolesta, jolloin niiden varsinaiset toteutukset voivat sijaita hajautettuina järjestelmän tietokoneisiin. Kaikkia järjestelmän objekteja voidaan kuitenkin käsitellä samalla tavalla ja jokaiselle objektille voidaan määrittää rajapinta, joka määrää sen kyvyt. Palvelujen käyttäjät eivät kuitenkaan tiedä tarjoaako objekti itsessään palvelun vai tarjoaako sen objektin edustaja.



Kuva 2. metaDESK arkkitehtuuri. [Ullmer and Ishii, 1997]

Alustan arkkitehtuuri on kerroksittainen. Kuvan 2 kaavio esittää kerrokset eriteltyinä fyysiseen arkkitehtuuriin ja ohjelmistoarkkitehtuuriin. Fyysinen arkkitehtuuri koostuu tietokoneista, sensoreista, aineellisista objekteista ja

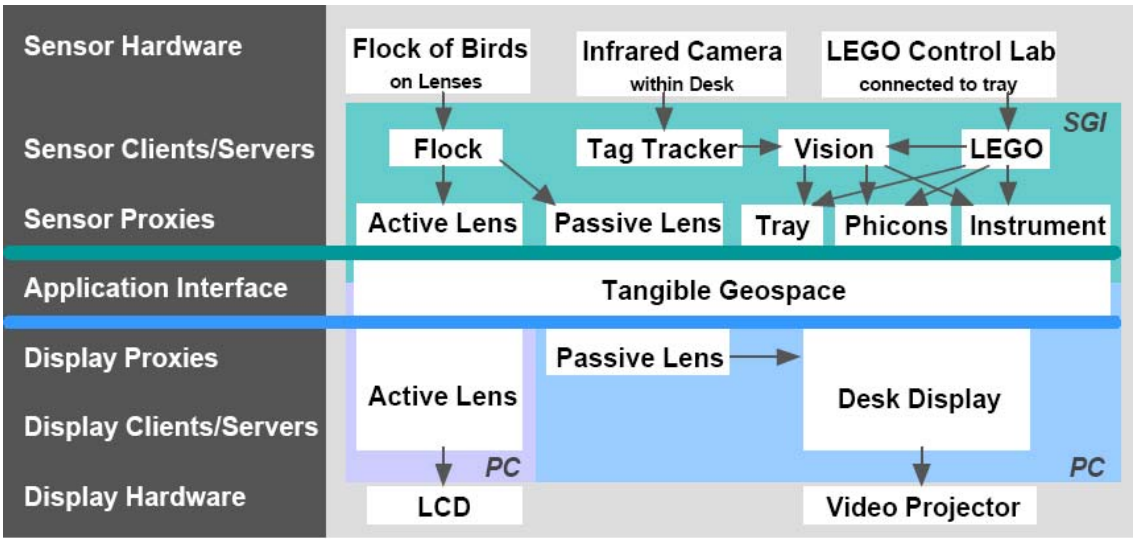
näytöistä. Ohjelmistoarkkitehtuurin kerrokset vastaavat aineellisia laitteita ja objekteja ja se jakautuu keskeltä kahteen osaan riippuen siitä, onko kyseessä olevalla objektilla sensori- vai näyttöominaisuuksia. Alustaan kuuluu lisäksi nimipalvelin, joka abstrahoi ja koordinoi järjestelmän hajautettuja resursseja.

Alustan kerroksittainen rakenne vähentää järjestelmän riippuvuuksia yksityiskohtia abstrahoimalla. Asiakas- ja palvelinkerrokset mahdollistavat verkottuneen kommunikoinnin sensoreille ja näytöille. Sensorien ja näyttöjen edustajakerrokset taas piilottavat objektien edustajilta kykyjen toteuttamiseen käytettävät todelliset teknologiat, jolloin käytettävän teknologian määrää vasta sensoriasiakas. Objektien edustajakerros puolestaan piilottaa sovellukselta kunkin syöte- tai tulostuspalvelun oikean tarjoajan.

Kuva 3. Tangible Geospacen objekti ja aktiivinen linssi. [Ullmer and Ishii, 1997]

Alustan päälle on kehitetty Tangible Geospace -prototyyppisovellus [Ullmer and Ishii, 1997], jossa käyttäjä voi rakennuksia symboloivia objekteja (kuva 3) syötelaitteina käyttämällä liikuttaa pöytätasolle heijastettua karttaa. Samaan aikaan käyttäjä voi katsoa aktiivisen linssin läpi alueen kolmiulotteista näkymää, jonka kuvakulma vaihtuu aktiivisen linssin (kuva 3) asennon ja sijainnin kulloinkin määräämänä. Käyttäjän on mahdollista myös katsoa vaihtoehtoista karttanäkymää tason päällä liikuteltavan passiivisen linssin läpi.

Kuvassa 4 on esitetty Tangible Geospace -sovelluksen arkkitehtuuri, joka perustuu proxdist-arkkitehtuuriin. Kuvasta nähdään kuinka usean objektin edustajat voivat käyttää samaa sensoria ja yksi edustaja voi käyttää useita sensoreita. Kuvasta näkyy myös, kuinka edustajakerrokset abstrahoivat alemmat kerrokset pois ylempien kerroksien näkyvistä yksinkertaistaen rakennetta.



Kuva 4. Tangible Geospace arkkitehtuuri. [Ullmer and Ishii, 1997]

2.3. Papier-Mâché

Papier-Mâché [Klemmer *et al.*, 2004] on kokoelma työkaluja [2], joilla käsinkosketeltavien käyttöliittymien kehittämistä voidaan nopeuttaa ja helpottaa. Se tarjoaa kehittäjien käyttöön useita eri syötelaitteita tukevat kirjastot sekä erilaisia syötteiden kanssa työskentelyä helpottavia työkaluja. Työkalut voivat avustaa esimerkiksi virheiden etsinnässä ja syötteiden simuloinnissa.

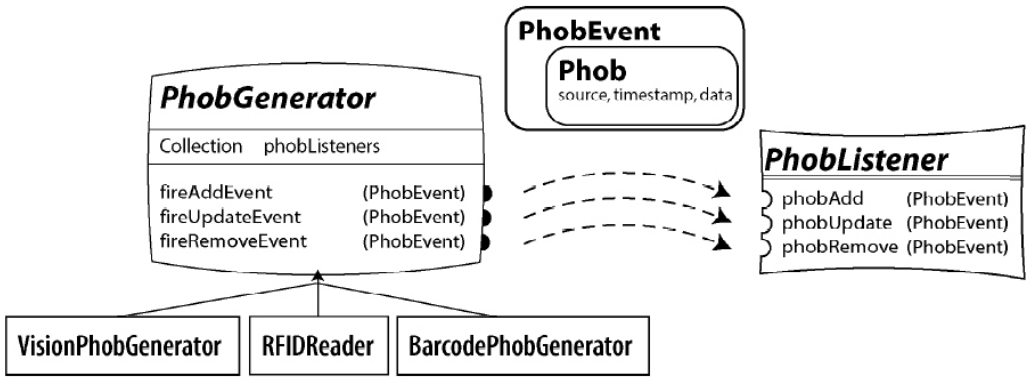
Järjestelmän suunnittelun lähtökohtana on ollut helpottaa käsinkosketeltavien käyttöliittymien kehittäjien työskentelyä erilaisten syötelaitteiden ja niiltä saatujen syötteiden kanssa. Koska suurin osa kehitysajasta kuluu tällä hetkellä vaikeuksiin syötelaitteiden kehittämisessä ja liittämisessä järjestelmään ja ongelmien ratkaisemiseen pystyvät vain alan asiantuntijat, on järjestelmän tarkoitus helpottaa ja nopeuttaa syötelaitteiden käyttöönottoa ja mahdollistaa niiden vaihtaminen mahdollisimman pienillä muutoksilla ohjelmakoodiin [Klemmer, 2003]. Järjestelmässä on toteutettuna kolme syöteteknologiaa: konenäkö, radiotaajuinen tunnistus (radio frequency identification, RFID) ja viivakooditunnistus. Mainittujen teknologioiden kanssa työskentelyä Papier-Mâché tukee abstrahoimalla laitteilta tulevan informaation käyttäjäystävälliseen ja syötelaiteriippumattomaan tapahtumamalliin, minkä

[2] Tietoa työkalujen saatavuudesta on projektin kotisivulla osoitteessa http://guir.berkeley.edu/projects/papier-mache/.

ansiosta kehittäjä voi unohtaa syötelaitekohtaiset yksityiskohdat ja keskittyä korkeamman tason käsitteisiin.

Papier-Mâché helpottaa myös prototyyppien luomista mahdollistamalla konenäön käyttämisen. Sovelluksen prototyyppi voidaan kehittää käyttäen konenäköä, jonka jälkeen lopullinen versio voidaan toteuttaa käyttämään jotakin muuta syötelaiteteknologiaa. Vaatimuksena on vain se, että lopullisen version syöteteknologia pystyy tuottamaan sovelluksessa käytettävät tapahtumat. Papier-Mâchéssa on erillinen käyttöliittymä, joka tarjoaa näkymiä objektien ja tilojen seurantaan. Siinä on myös tälle sovellusalueelle edistyksellisen Wizard of Oz -toiminto (WOz), joka mahdollistaa syötelaitteiden simuloinnin ilman niiden olemassaoloa sekä helpottaa virheiden etsintää ja testausta.

Papier-Mâchéa käyttäen on kehitetty useita käsinkosketeltavia käyttöliittymiä. SiteView:n [Beckmann and Dey, 2003]avulla käyttäjä voi hallita kodin automaatiota. Käyttäjä käsittelee rakennuksen pohjapiirustuksen päällä aineellisia ikoneita, joiden avulla hän luo sääntöjä tietyin ehdoin tapahtuville automaatiotoiminnoille. Järjestelmä käyttää sensoreita ympäristön tilan havainnointiin ja antaa kuvallista palautetta sääntöjen vaikutuksista. Aineellisten ikonien tunnistus tapahtuu konenäön avulla; ikonien koko ja väri ovat tunnistamisessa käytettyjä ominaisuuksia.



Kuva 5. Papier-Mâchén tapahtuma-arkkitehtuuri. [Klemmer, 2003]

Papier-Mâché on kokoelma avoimen lähdekoodin Java-luokkia, joista osa esitetään kuvassa 5. Tapahtumien tuottaja (PhobGenerator) luo tunnistustensa perusteella tapahtumia (PhobEvent), jotka sisältävät tiedon niihin liittyvästä tunnistetusta objektista (Phob). Tunnistusteknologiasta riippuen käytössä on jokin tietty tuottaja, kuten viivakoodinlukijan tapauksessa BarcodePhob-Generator. Yhteystaulut (AssociationMap) liittävät tapahtumat konkreettisiin

toimintoihin, kuten esimerkiksi äänitiedoston soittamiseen. Tapahtuma-kuuntelijat (PhobListener) puolestaan kuuntelevat tuottajien luomia tapahtumia ja pystyvät reagoimaan niihin. Tapahtumasuodattimet (EventFilter) mahdollistavat ennalta määriteltyjen esineiden havaitsemisen ja paikallistamisen konenäön avulla.

Tapahtumat voivat sisältää toisistaan eroavaa informaatiota riippuen syötelaitteen tarjoamista mahdollisuuksista. Konenäkö esimerkiksi välittää kohteen koon, asennon, värin ja sijainnin, kun taas radiotaajuinen tunnistus pelkästään kohteen ja lukijan tunnisteet. Tapahtumat ovat kuitenkin eri laitteiden välillä yhdenmukaisia, joten rajoitetusti käytettynä ne mahdollistavat laiteriippumattomuuden.

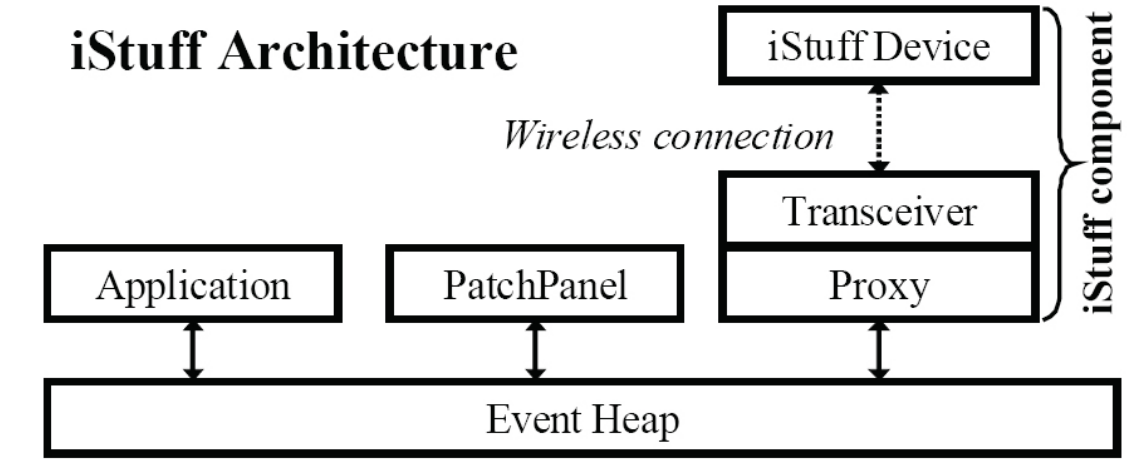
2.4. iStuff

iStuff [3] [Ballagas *et al.*, 2003] on kehitetty helpottamaan uusia käyttöliittymätekniikoita toteuttavien prototyyppien kehittämistä erityisesti jokapaikan tietotekniikan alueella. Sen tarkoituksena on mahdollistaa monien eri syöte- ja tulostuslaitteiden samanaikainen käyttö. iStuffin avulla voidaan yhdistää lukuisia langattomia laitteita sovellukseen siten, että eri laitteet voivat toimia ohjaimina samoille toiminnoille.

iStuff on ollut käytössä neuvottelutilaa muistuttavassa, interaktiivisessa iRoom-huoneessa [Ballagas *et al.*, 2003]. iStuff-laitekomponentteja on kehitetty toimintoihin, joilla pystytään vaikuttamaan huoneessa käytössä oleviin sovelluksiin. Erilaisia kytkimiä ja sensoreita on liitetty esimerkiksi vaikuttamaan ympäristön tilaan ja huoneen seinällä olevan näyttötaulun tapahtumiin.

Kehittäjät asettivat järjestelmälle vaatimuksiksi laajan laitevalikoiman tukemisen, riippumattomuuden alustasta ja langattomasta protokollasta, olemassa oleviin järjestelmiin liittämisen helppouden ja tuen monille samanaikaisille käyttäjille. Tätä varten he kehittivät kuvassa 6 esitetyn iStuff-arkkitehtuurin.

[3] Tietoa kehyksen saatavuudesta on projektin kotisivulla osoitteessa http://media.informatik.rwth-aachen.de/istuff/tiki-index.php.



Kuva 6. iStuff arkkitehtuuri. [Ballagas *et al.*, 2003]

Fyysinen laite ja siihen liittyvä ohjelmistoedustaja muodostavat iStuff-komponentin, johon kuuluvan laitteen monimutkaisuus voi vaihdella melko monimutkaisesta hyvin yksinkertaiseen. Jotta jokin laite voidaan liittää iStuff-ympäristöön, täytyy laitetta varten olla toteutettuna edustaja. Jokaisella laitteella täytyy siis olla edustaja, mutta sama edustaja voi edustaa myös muita laitteita. Riippuen siitä onko laite syöte- vai tulostuslaite, täytyy sen edustajan joko kapseloida dataa tapahtumiin tai osata ottaa sitä sieltä. Komponenttirakenne tekee laitteiden liittämisestä ympäristöön helppoa ja mahdollistaa liittämisen myös olemassa oleville kolmansien osapuolten kehittämille laitteille.

iStuff-komponentit kommunikoivat sovellusten kanssa monikoiden (tuple) avulla. Monikoiden käyttötarkoituksesta johtuen niitä kutsutaan tässä tapahtumiksi. Kullakin tapahtumalla on tyyppi sekä avain-arvo-pareja (key-value pairs), joissa on tapahtuman kannalta oleellista informaatiota. Tuottajat (producer) välittävät tapahtumia tapahtumapinolle (event heap) [Johanson and Fox, 2002], joka taas välittää ne kuluttajille (consumer). Kuluttajat saavat käsiteltävikseen vain ne tapahtumat, joita ne ovat rekisteröityneet kuuntelemaan. Tuottajat ja kuluttajat voivat myös kuulua ryhmiin, ja tällöin tapahtumien lähetys ja vastaanotto voidaan määritellä ryhmien perusteella.

Järjestelmän keskeisin osa on *tapahtumapino*, joka on tapahtumia asiakkailta vastaanottava ja niitä eteenpäin määrätyille vastaanottajille välittävä palvelinprosessi. Kaikki eri laitteiden ja sovellusten tapahtumat menevät tapahtumapinoon, joka välittää ne eteenpäin kaiken tyyppisiä tapahtumia kuuntelevalle muunnostaululle (patch panel). *Muunnostaulun* [Ballagas *et al.*, 2004] avulla komponenttien tapahtumat voidaan kytkeä haluttuihin sovelluksiin,

sillä muunnostaulu määrittää kuinka laitteilta tulevat tapahtumat muunnetaan sovelluksen ymmärtämiksi tapahtumiksi.

Muunnokset tehdään alkuperäiset tapahtumat säilyttäen. Yksi lähdetapahtuma voidaan jakaa useiksi eri tapahtumiksi ja monta tapahtumaa voidaan määrittää yhdessä toteutuessaan laukaisemaan uusi tapahtuma. Muunnoksen yhteydessä voidaan suorittaa muutoksia tai vaihdoksia myös tapahtumien avaimille ja arvoille.

Muunnostaulu voidaan ajonaikaisesti määrätä tekemään uudentyyppisiä tapahtumamuunnoksia ja se mahdollistaa tilakoneiden (state machine) käytön tapahtumien avulla. iStuff-komponentit määräävät tuottamiensa tapahtumien tyypit, mutta kehittäjiä kehotetaan käyttämään kunkin sovelluksen kontekstissa merkityksellisiä tapahtumia, jotta laitteiden vaihtaminen olisi mahdollista ilman muutoksia sovelluksen koodiin.

2.5. Phidgets

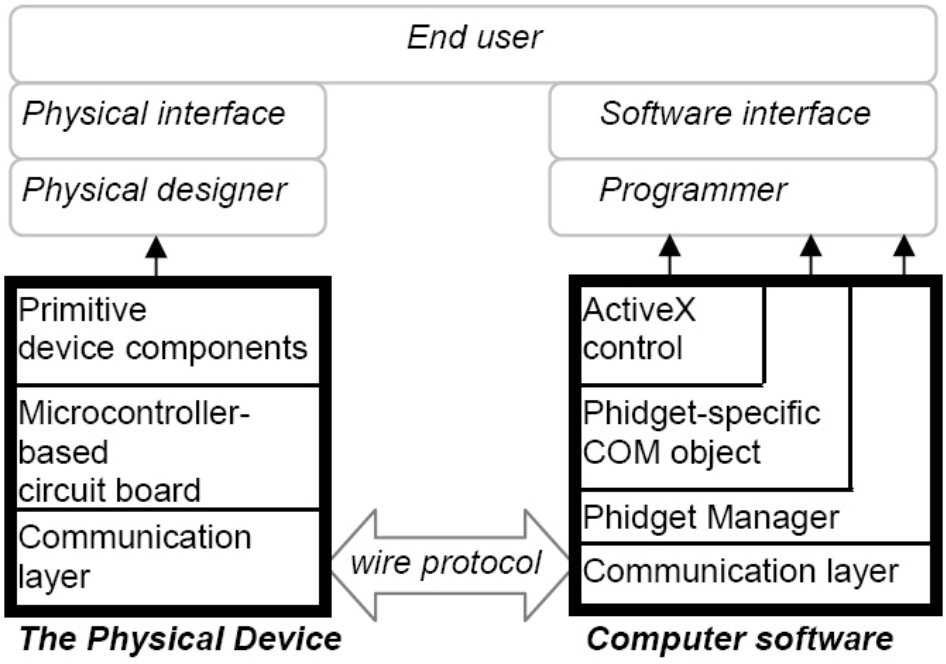
Phidgetsien [4] [Greenberg and Fitchett, 2001] tarkoitus on helpottaa käsinkosketeltavien käyttöliittymien luontia samalla tavalla kuin valmiit käyttöliittymäkomponentit (widget) ovat helpottaneet graafisten käyttöliittymien kehittämistä, eli abstrahoimalla syöte- ja tulostuskokonaisuuksia valmiiksi komponenteiksi. Ratkaistavana ongelmana on siis laitteiden yksinkertainen liittäminen osaksi sovelluksen ohjelmakoodia, sillä nykyisellään yksinkertaistenkin elektronisten laitteiden kokoaminen ja ohjelmointi on hidasta ja vaivalloista. Liitettäviä laitteita voivat olla erilaiset sensorit, moottorit, kamerat tai näytöt.

Olennaisena osana graafisten käyttöliittymäkomponenttien menestystä ovat olleet tarkasti määritellyt rajapinnat, jotka ovat mahdollistaneet komponenttien helpon käytön vaikka itse toiminnallisuus onkin ollut kapseloituna kehittäjän näkymättömiin. Siten kehittäjät ovat pystyneet keskittymään olennaiseen eli vuorovaikutuksen suunnitteluun. Phidgetseillä on kuitenkin graafisista käyttöliittymäkomponenteista eroavia ominaisuuksia, jotka lisäävät niille asetettuja vaatimuksia. Niiden on pystyttävä informoimaan sovellusta kulloinkin saatavilla olevista laitteista ja erottelemaan yksiselitteisesti laitteet toisistaan. Kehittäjillä olisi myös hyvä olla mahdollisuus sovelluksen testaukseen, vaikka jotakin laitetta ei olisikaan saatavilla.

[4] Tietoa komponenttien saatavuudesta osoitteessa http://www.phidgets.com/.

Phidgetsejä käyttäen on kehitetty monia pieniä sovelluksia testaamaan arkkitehtuurin toimivuutta. Alan opiskelijoille annettiin tehtäväksi toteuttaa Phidgetsejä käyttämällä mielikuvituksellinen käyttöliittymä ilman aikaisempaa kokemusta kyseisistä laitteista [Greenberg and Fitchett, 2001]. Tuloksena oli pieniä sovelluksia, joissa oli käytetty hyväksi erilaisia elektronisia sensoreita ja moottoreita. Esimerkkinä voidaan mainita sovellus, joka näyttää kääntyvän viisarin avulla puhelimeen tulleet vastaamattomat puhelut. Toteutuksessaan se käyttää hyväkseen puhelimelle varta vasten suunniteltua jalustaa, joka aistii puhelun tärinää havaitsevilla sensoreilla ja kääntää viisaria ohjelmaan liitetyn moottorin avulla.

Kuvassa 7 on esitetty Phidgetsien arkkitehtuuri, joka voidaan jakaa karkeasti kolmeen osaan: itse fyysiseen laitteeseen, siihen liittyvään ohjelmistokoodiin ja niiden väliseen kommunikointiprotokollaan. Fyysinen laite koostuu elektronisista komponenteista ja on liitettävissä tietokoneen USB-porttiin. Kommunikointiprotokolla välittää informaatiota laitteen ja tietokoneen välillä, mutta ei näy käyttäjälle. Laitteet osaavat välittää tyyppinsä, tunnistenumeronsa ja omaan tilaansa liittyviä viestejä sekä vastaanottaa tietokoneen esittämiä pyyntöjä.



Kuva 7. Phidgets arkkitehtuuri. [Greenberg and Fitchett, 2001]

Ohjelmistopuolella Phidget manager tarjoaa kehittäjälle tavan päästä käsiksi tietokoneeseen sillä hetkellä kiinnitettynä oleviin laitteisiin. Se ilmoittaa sovelluksille uusien laitteiden liittämisestä ja vanhojen poistamisesta. Phidget manager luo kaikista laitteista COM-objektin, jota kehittäjä voi käyttää joko yleisen rajapinnan tai laitteille yksilöllisten rajapintojen kautta. Tietylle laitteelle ominainen rajapinta on laajempi ja sisältää kutsut kaikkiin kyseisen laitteen toteuttamiin toimintoihin. COM-objekteista luodaan myös ActiveX-kontrollit, jotka mahdollistavat graafisen näkymän laitteisiin ja laitteiden simuloinnin ohjelmakoodin avulla.

Phidgetsin kanssa samoihin haasteisiin vastaa Calder-toolkit [Lee *et al.*, 2004], joka keskittyy kenties vielä enemmän prototyyppien toteuttamisen helpottamiseen, ottamalla huomioon suunnittelijoiden työssään käyttämät menetelmät. Myös Calder-toolkit tarjoaa kehittäjille joukon uudelleenkäytettäviä syöte- ja tulostuskomponentteja, jotka voivat Calder-toolkitin tapauksessa olla myös langattomia. Toteutuksen arkkitehtuuri keskittyy suurelta osin elektronisiin laitteisiin. Ohjelmistoarkkitehtuurissa jokaista laitetta vastaa olio, joka esittää laitteen tilan ja kapseloi sen toiminnallisuuden siten, että laitteen ja olion tilat vastaavat aina toisiaan. Myös Calder-toolkit näyttää laitteissa tapahtuvat muutokset ohjelmistopuolen tapahtumina.

3. Arkkitehtuuriratkaisujen vertailua

Tässä luvussa vertailen edellisessä luvussa esittelemiäni arkkitehtuureja kahdesta eri näkökulmasta sekä esitän joitakin arvioita käsinkosketeltavien käyttöliittymien arkkitehtuurien tulevaisuudesta. Ensimmäisessä kohdassa arvioin arkkitehtuureja sen perusteella, kuinka hyvin niissä käytetyt ratkaisut tukevat käyttöliittymänhallintajärjestelmältä (user interface management system, UIMS) vaadittavia piirteitä. Toisessa kohdassa vertailen kuinka hyvin arkkitehtuurit toteuttavat uusien vuorovaikutustapojen asettamia vaatimuksia. Kummassakin kohdassa arviointi perustuu kunkin arkkitehtuurin esittelyn yhteydessä mainittuihin artikkeleihin. Lopuksi esitän mahdollisia kehityssuuntia arkkitehtuureille.

3.1. Käyttöliittymänhallintajärjestelmältä vaadittavat ominaisuudet

Taulukossa 1 kuvataan mitkä edellisessä luvussa esiteltyjen arkkitehtuurien ratkaisut tukevat hyvältä käyttöliittymänhallintajärjestelmältä vaadittavia

ominaisuuksia. Käytän valitsemiani ja muokkaamiani ominaisuuksia Bettsin ja muiden [1987] luokittelusta.

	DWARF	**meta-DESK**	**Papier-Mâché**	**iStuff**	**Phidgets**
Tuki uudelleen-käytölle	Kompo-nentit	Edustajat		Edustajat	
Tuki laajenta-miselle	Kehys	Edustajat	Tapahtuma-malli	Tapahtuma-malli ja edustajat	
Tuki prototyyppien kehittämiselle	Kehys	Alusta	Tapahtuma-malli	Tapahtuma-malli ja laitteet	Laitteet
Tuki testaukselle			Simu-lointi		Simu-lointi
Tuki syöte-laitteiden vaihtamiselle	Kompo-nentit	Edustajat	Valmiiksi toteutetut laitteet	Edustajat	

Taulukko 1. Arkkitehtuureissa käytetyt käyttöliittymänhallintajärjestelmältä vaadittavia ominaisuuksia tukevat ratkaisut.

Uudelleenkäytettävyyden ja laajennettavuuden vertailua hankaloittaa arkkitehtuurien erilaisuus. DWARF-kehykseen toteutettuja komponentteja voidaan käyttää uudestaan muissa kehyksen päälle toteutetuissa sovelluksissa, ja kehys puolestaan on itse laajennettavissa komponenteilla. Edustajien avulla uudelleenkäyttöä ja laajennettavuutta parantavat metaDESK ja iStuff, ja tapahtumamallin avulla Papier-Mâché ja iStuff. Kaikki arkkitehtuurit ovat myös itsessään tarkoitettu uudelleenkäytettäviksi.

Prototyyppien kehittämistä arkkitehtuurit tukevat eri tasoilla. DWARF tarjoaa sovellusten pohjaksi ohjelmistokehyksen ja metaDESK taas fyysisistä laitteista ja ohjelmistosta koostuvan alustan. Papier-Mâché ja iStuff sisältävät valmiin tapahtumamallin, jota sovellukset voivat hyödyntää. iStuff ja Phidgets mahdollistavat lisäksi laitteiden nopean liittämisen.

Syötelaitteiden vaihtaminen on hyvin tuettua. Laiteliittymiä toteuttamalla sen mahdollistavat DWARF-kehys, metaDESK ja iStuff. Papier-Mâchéssa vaihdon voi tehdä valmiiksi toteutettujen syöteteknologioiden välillä. Syötelaitteiden testauksen avuksi Papier-Mâché ja Phidgets tarjoavat simulointityökalun.

3.2. Uusien vuorovaikutustekniikoiden vaatimat ominaisuudet

Taulukossa 2 kuvataan mitä uusille vuorovaikutustavoille tyypilliseksi havaittuja ominaisuuksia [Jacob *et al.*, 1999; Shaer, 2005] edellisessä luvussa esitellyt arkkitehtuurit tukevat.

	DWARF	**meta-DESK**	**Papier-Mâché**	**iStuff**	**Phidgets**
Käyttäjän tarkkailu	X				
Yhdistettävät kanavat	X			X	
Tulkittavat komennot	X	X			
Jatkuvat syötteet	X	X	X	X	
Erilliset syötteet	X	X	X	X	X
Epästandardit laitteet	X	X	X	X	X

Taulukko 2. Arkkitehtuurien toteuttamat uusilta vuorovaikutustekniikoilta vaadittavat ominaisuudet.

DWARF-kehys tukee vaadittuja ominaisuuksia parhaiten, mutta myös metaDESK ja iStuff tukevat niitä hyvin. Muut arkkitehtuurit ovat havaintojen perusteella suunniteltu helpottamaan erilaisten syötelaitteiden liittämistä ja niitä käyttävien sovellusten kehittämistä lähinnä aikaisemmista vuorovaikutustavoista tuttuja tekniikoita käyttäen. Alla vertailtavien piirteiden lisäksi arkkitehtuurin hyödyllisyyteen vaikuttavat paljon myös järjestelmälle asetetut reaaliaikaisuusvaatimukset, joita ei kuitenkaan tässä voida analysoida.

Edellä kuvatun perusteella DWARF-kehys soveltuu käytettäväksi silloin kun täytyy kehittää uusia vuorovaikutustekniikoita, sillä kehys mahdollistaa kehittyneiden ominaisuuksien toteuttamisen ja on laajennettavissa komponenteilla. Se ei kuitenkaan tue käsinkosketeltavien käyttöliittymien ominaispiirteitä niin hyvin kuin varta vasten siihen tarkoitukseen suunnitellut järjestelmät. metaDESK, joka sekin toteuttaa hyvin vaativia ominaisuuksia, on puolestaan melko tiukasti sidottu omaan fyysiseen arkkitehtuuriinsa ja siten ei ole kovinkaan helposti käytettävissä erilaisten käsinkosketeltavien käyttöliittymien prototyyppien kehittämiseen. Valmiit laitetoteutukset sisältävä Papier-Mâché ja erilaisten laitteiden liittämisen mahdollistava iStuff ovat helposti laajennettavissa niiden omien tapahtumamallien päälle. Tapahtumiin perustuva vuorovaikutus ja kehittyneiden ominaisuuksien puuttuminen kuitenkin rajoittavat niiden tarjoamia mahdollisuuksia. Phidgets tarjoaa kehittäjien käyttöön valmiita syötelaitekokonaisuuksia, joita hyväksi käyttäen on mahdollista rakentaa kehittyneitäkin vuorovaikutustekniikoita, mutta jotka eivät kuitenkaan tue käsinkosketeltavien käyttöliittymien toteutusta laiterajapinnasta ylöspäin.

3.3. Arkkitehtuurivaatimukset tulevaisuudessa

Käsinkosketeltavien käyttöliittymien kehittyessä tulee järjestelmiin varmasti sellaisia muutoksia, jotka vaikuttavat niiden arkkitehtuureihin. Vaikka edellä esitetyt arkkitehtuurit tukevat melko hyvin tämän hetken käyttöliittymiä, ilmenee jatkossa uusia vaatimuksia joihin ei ole aikaisemmin varauduttu.

Eräs mielenkiintoinen kehityssuunta on käsinkosketeltavaa palautetta käyttävät käyttöliittymät, joiden toteutustekniikka voi perustua esimerkiksi magneettikenttiin [Pangaro *et al.*, 2002] tai ilmalla täyttyviin soluihin [Mazzone *et al.*, 2004]. Vaikka aineellisen palautteen merkitys järjestelmän käytön ja fyysisen toteutuksen kannalta onkin suuri, ei sen lisääminen välttämättä aiheuta suuria muutoksia ohjelmistoarkkitehtuureihin.

Enemmän muutoksia arkkitehtuureihin aiheutuu jos käsinkosketeltavissa käyttöliittymissä aletaan hyödyntää 3D-tulostinten[5] suomia mahdollisuuksia. Näiden avulla käyttöliittymiin kuuluvia uusia aineellisia objekteja voidaan

[5] 3D-tulostimia valmistavat esimerkiksi Stratasys (http://stratasys.com/), Z Corporation (http://www.zcorp.com/) ja 3D Systems (http://www.3dsystems.com/).

tuottaa ohjelmallisesti. Tällöin tulee ongelmaksi reagointi uusiin objekteihin, joita voidaan luoda parhaimmillaan jopa ajonaikaisesti.

Osa esitellyistä arkkitehtuureista on hajautettu usealle tietokoneelle, mutta kaikki sovellukset on suunniteltu toimimaan itsenäisesti omia objektejaan käyttäen. Jos eri objektien pitäisi toimia useissa eri järjestelmissä, esimerkiksi eri konteksteihin vietynä, tulee toteutukseen lisähaasteita. Tällöin vaaditaan nykyistäkin hajautetumpia arkkitehtuureja.

4. Yhteenveto

Arkkitehtuuriratkaisuja tulee käyttää käsinkosketeltavien käyttöliittymien toteutuksessa hyödyksi, sillä käsinkosketeltavuus asettaa järjestelmille vaatimuksia, joita aikaisemmat arkkitehtuurit eivät täytä. Olen esitellyt viisi erilaista käsinkosketeltavien käyttöliittymien arkkitehtuuria sekä niissä käytettyjä ratkaisuja. Tärkeimmät ratkaisumallit joilla vaatimuksiin on esitellyissä arkkitehtuureissa vastattu, ovat komponenttipohjaisuus, edustaja-suunnittelumalli ja tapahtumapohjaisuus. Ratkaisut tukevat hyvin sovellusten prototyyppien luomista ja erilaisten syöte- ja tulostuslaitteiden liittämistä, mutta puutteellisesti uudenlaisien vuorovaikutustekniikoiden toteuttamista.

Mitä uusia vuorovaikutustapoja arkkitehtuurien sitten tulisi tukea? Näyttäisi siltä, että valikko- ja komentopohjaisista käyttöliittymistä periytyvä tapahtumamalli jossain määrin rajoittaa uusien vuorovaikutusten kehitystä tai ainakin ohjaa sitä tiettyyn suuntaan. Uusia arkkitehtuureja varten tulisi kehittää vuorovaikutukseen sellainen malli, joka mahdollistaa rinnakkaiset ja toisiinsa vaikuttavat komennot sekä tukee usean käyttäjän yhtäaikaista työskentelyä. Samalla mallin tulisi mahdollistaa käyttäjän tarkkailemisen perusteella toimiminen sekä vaikeasti tulkittavien komentojen päätteleminen.

Viiteluettelo

[Ballagas *et al.*, 2003] Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers, iStuff: a physical user interface toolkit for ubiquitous computing environments. In: *Proceedings of CHI 2003, Conference on Human Factors in Computing Systems* (2003), ACM Press, 537-544.

[Ballagas *et al.*, 2004] Rafael Ballagas, Andy Szybalski, and Armando Fox, Patch panel: enabling control-flow interoperability in ubicomp environments. In:

Proceedings of Second IEEE International Conference on Pervasive Computing and Communications (2004), IEEE Computer Society, 241-252.

[Beckmann and Dey, 2003] Chris Beckmann and Anind Dey, SiteView: tangibly programming active environments with predictive visualization. *Intel Research Tech Report IRB-TR-03-019* (2003).

[Betts *et al.*, 1987] Bill Betts, David Burlingame, Gerhard Fischer, Jim Foley, Mark Green, David Kasik, Stephen T. Kerr, Dan Olsen, and James Thomas, Goals and objectives for user interface software. *ACM Computer Graphics* **21,** 2 (1987), 73-78.

[Buschmann *et al.*, 1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[Gamma *et al.*, 1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.

[Greenberg and Fitchett, 2001] Saul Greenberg and Chester Fitchett, Phidgets: easy development of physical interfaces through physical widgets. In: *Proceedings of the ACM Symposium on User Interface Software and Technology* (2001), ACM Press, 209-218.

[Ishii and Ullmer, 1997] Hiroshi Ishii and Brygg Ullmer, Tangible Bits: towards seamless interfaces between people, bits and atoms. In: *Proceedings of CHI 1997, Conference on Human Factors in Computing Systems* (1997), ACM Press, 234-241.

[Jacob *et al.*, 1999] Robert J. K. Jacob, Leonidas Deligiannidis and Stephen Morrison, A software model and specification language for non-WIMP user interfaces. *Transactions on Computer-Human Interaction* **6**, 1 (1999), 1-46.

[Johanson and Fox, 2002] Brad Johanson and Armando Fox, The event heap: a coordination infrastructure for interactive workspaces. In: *Proceedings of 4th IEEE Workshop on Mobile Computing Systems and Applications* (2002), IEEE Computer Society, 83-93.

[Klemmer, 2003] Scott Klemmer, Papier-Mâché: Toolkit support for tangible interaction. In: *Proceedings of ACM Symposium on User Interface Software and Technology: Doctoral Consortium* (2003).

[Klemmer *et al.*, 2004] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay, Papier-Mâché: toolkit support for tangible input. In: *Proceedings of*

CHI 2004, Conference on Human Factors in Computing Systems (2004), ACM Press, 399-406.

[Lee *et al.*, 2004] Johnny C. Lee, Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz, and Darren Leigh, The Calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In: *Proceedings of DIS 2004, Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, (2004), ACM Press, 167-175.

[Mazzone *et al.*, 2004] Andrea Mazzone, Christian P. Spagno and Andreas M. Kunz, The HoverMesh: a deformable structure based on vacuum cells: new advances in the research of tangible user interfaces. In: *Proceedings of ACE 2004, International Conference on Advances in Computer Entertainment Technology* (2004), ACM Press, 187-193.

[Pangaro *et al.*, 2002] Gian Pangaro, Dan Maynes-Aminzade and Hiroshi Ishii, The actuated workbench: computer-controlled actuation in tabletop tangible interfaces. In: *Proceedings of ACM Symposium on User Interface Software and Technology*, (2002), ACM Press, 181-190.

[Sandor and Klinker, 2005] Christian Sandor and Gudrun Klinker, A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. *Personal and Ubiquitous Computing* **9**, 3 (2005), 169-185.

[Shaer, 2005] Orit Shaer, A framework for building reality-based interfaces for wireless-grid applications. In: *Proceedings of CHI 2005, Conference on Human Factors in Computing Systems, Extended Abstracts* (2005), ACM Press, 1128-1129.

[Ullmer, 1997] Brygg Ullmer, 3wish: distributed [incr Tcl] extensions for physical-world interfaces. In: *Proceedings of Fifth Annual Tcl\Tk Workshop* (1997), USENIX, 169-170.

[Ullmer and Ishii, 1997] Brygg Ullmer and Hiroshi Ishii, The metaDESK: models and prototypes for tangible user interfaces. In: *Proceedings of ACM Symposium on User Interface Software and Technology* (1997), ACM Press, 223-232.

Sovelluslogiikan eriyttäminen käyttöliittymästä

Juuso Näsi

Tiivistelmä

Sovellusten koon yleinen kasvaminen on johtanut siihen, että niiden suunnittelu, toteutus ja ylläpito on vaikeutunut jatkuvasti. Ohjelmakoodin huonon uudelleenkäytettävyyden takia sovelluskehittäjät joutuvat tekemään jo kertaalleen tehtyä työtä uudelleen. Näihin ongelmiin on haettu ratkaisua hajauttamalla ohjelman sovelluslogiikka erilleen käyttöliittymästä. Tässä artikkelissa käsitellään aiheen historiallinen tausta ja suurimpia syitä siihen miksi nykyisissä arkkitehtuureissa ja suunnittelumalleissa on usein päädytty eriyttämään sovelluslogiikka käyttöliittymästä. Artikkelissa esitellään historiallisesti ja käytännön ohjelmistokehityksen kannalta keskeisiä arkkitehtuurimalleja. Tämän jälkeen käsitellään tarkemmin MVC-mallia (Model-View-Controller), jonka perusajatuksena on erottaa käyttöliittymä sovelluksen ydintoiminnallisuudesta ja minimoida käyttöliittymän muutosten vaikutukset sovelluksen toimintaan. Erityisesti tutkitaan MVC-mallin soveltuvuutta moderneihin käyttöliittymiin ja etsitään MVC-mallin sovellusalueen rajoja. Lopuksi pohditaan yleiseltä kannalta käyttöliittymäarkkitehtuurien tulevaisuutta ja uusien arkkitehtuurien kehittämismahdollisuuksia.

Aivansanat ja -sanonnat: UIMS, MVC, Seeheim, Arch, PAC, PAC-Amodeus, PAC*, käyttöliittymäarkkitehtuuri, multimodaalinen, ryhmäohjelma, openMVC
CR-luokat: D.2.11, D.2.13, H.5.2, I.3.6.

1. Johdanto

Uudet vuorovaikutustekniikat ja sovellusten multimodaalistuminen asettavat jatkuvasti uusia haasteita ohjelmistokehitykselle ja erityisesti käyttöliittymien suunnittelulle. Tulevaisuuden näkymät viittaavat siihen, että esimerkiksi puhe- ja kosketuskäyttöliittymät tulevat yleistymään perinteisten käyttöliittymien rinnalla. Multimodaalisten sovellusten kehitys on pitkälti riippuvaista tarjolla olevista arkkitehtuureista ja niiden hyödyntämisestä oikealla tavalla.

Perinteiset WIMP-käyttöliittymät (Window, Icon, Mouse, Pointer) sisältävät yleensä vuorottelevan ja jaksottaisen syöte- sekä tulostevirran ja ovat näin ollen yksinkertaisempia kuin multimodaaliset käyttöliittymät. Tästä syystä myöskään käyttöliittymäarkkitehtuurilta ei vaadita niin paljon perinteisiä käyttöliittymiä suunniteltaessa. Nykyaikaisia monen aistikanavan käyttöliittymiä suunniteltaessa vanhat ja perinteiset arkkitehtuurit ovat kuitenkin osoittautuneet hankaliksi. Useimmiten ongelmana on ollut sovelluksen eri osien suuri riippuvuus toisistaan ja yleinen ohjelmakoodin sidonnaisuus. Uusia arkkitehtuureita on jatkuvasti yritetty kehittää helpottamaan kompleksisten sovellusten suunnittelua. Nykyajan ohjelmistokehitys on yleensä luonteeltaan inkrementaalista ja iteratiivista. Tästä syystä useimmissa arkkitehtuureissa keskeisimpänä lähtökohtana on ollut jakaa sovellus mahdollisimman itsenäisiin moduuleihin, joiden ominaisuudet ovat myöhemmin muokattavissa ilman koko sovelluksen uusimista.

Tässä artikkelissa pyritään luomaan yleiskuva käyttöliittymä-arkkitehtuureista, jotka mahdollistavat sovelluslogiikan ja käyttöliittymän eriyttämisen. Tarkemmin käsitellään perinteisissä graafisissa WIMP-käyttöliittymissä laajalti käytettyä MVC-mallia ja pyritään arvioimaan sen soveltuvuutta erityyppisiin sovelluksiin ja erityisesti nykyaikaisiin käyttöliittymiin. Osaltaan pyritään myös arvioimaan MVC-mallin pohjalta kehitettyjen muiden agenttiarkkitehtuurien hyviä ja huonoja puolia MVC-malliin nähden. Esimerkkitapauksissa pyritään arvioimaan MVC-mallin mahdollisia ongelmia tietynlaisten sovellusten yhteydessä ja visioimaan ongelmia korjaavia muutoksia. Artikkeli esittelee lyhyesti myös web-sovelluksia varten kehitetyn OpenMVC arkkitehtuurin, joka pyrkii vähentämään sovellusten sidonnaisuutta kaupallisiin teknologioihin. Artikkeli arvioi myös millaisia uusia tarpeita tulevaisuuden käyttöliittymät saattavat luoda käyttöliittymäarkkitehtuureille.

2. Käyttöliittymien arkkitehtuureista

2.1. Yleistä

Käyttöliittymäarkkitehtuurilla tarkoitetaan tapaa määritellä sovelluksen rakenne yleisellä tasolla. Arkkitehtuuritasolla ei vielä puututa toteutuksen yksityiskohtiin kuten ohjelmointikehykseen tai ohjelmointikieleen, mutta sen sijaan pyritään määrittelemään sovelluksen rakenne ja esimerkiksi osien välinen kommunikaatio yleisellä tasolla. Arkkitehtuurisuunnittelussa sovelluksen eri tehtävät jaetaan

yleensä kerroksittain tai moduuleittain tietylle osalle sovellusta. Suunnittelun tavoitteina on usein tehdä sovelluksen osista mahdollisimman riippumattomia, loogisia ja helposti uudelleenkäytettäviä. Luonnollisesti on tärkeää myös luoda edellytykset nopealle ohjelmiston kehittämiselle ja pyrkiä vähentämään tulevan ylläpidon vaatimaa aikaa mahdollisuuksien mukaan.

2.2. Käyttöliittymien hallintajärjestelmät

Käyttöliittymän hallintajärjestelmällä (UIMS, User Interface Management System) on terminä monia eri tulkintoja, mutta yleisesti sitä pidetään sovelluksesta erillisenä käyttöliittymästä huolehtivana osana. Jotkut tulkinnat puolestaan pitävät käyttöliittymän hallintajärjestelmää vastakkaisena terminä tietokannanhallintajärjestelmälle (DBMS, Database Management System), jonka tehtävä on huolehtia matalan tason datatoiminnoista. [Dobos, 2002]. Kaikille tulkinnoille on kuitenkin yhteistä, että käyttöliittymän hallintajärjestelmän pääasiallinen tarkoitus on abstraktoida käyttöliittymä erilliseksi osaksi jossakin suuremmassa kokonaisuudessa. Sovelluslogiikan ja käyttöliittymän eriyttämiseen pyrittäessä on useimmiten kuitenkin yritetty tehdä sovelluslogiikasta uudelleenkäytettävä komponentti sen keskimäärin suuremman monimutkaisuusasteen vuoksi.

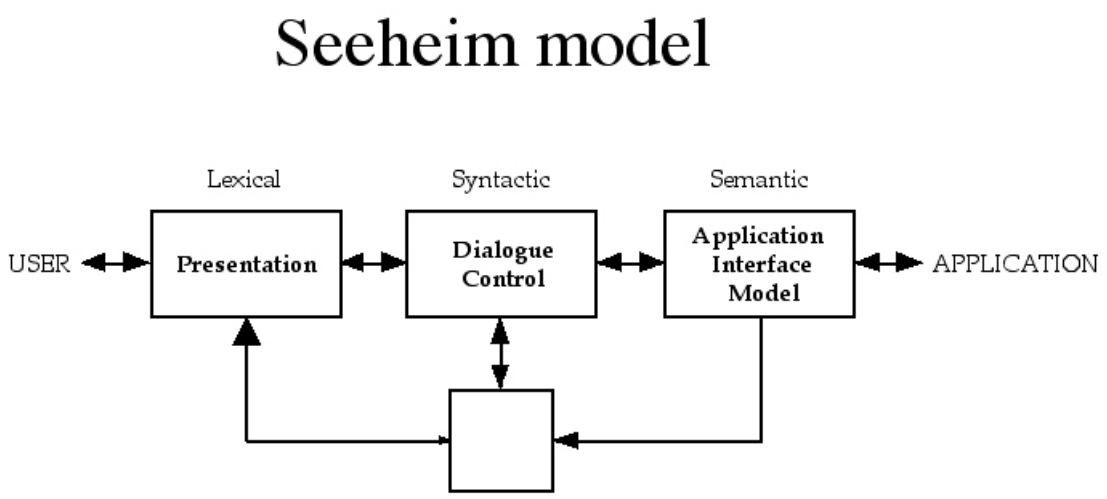
2.3. Eri arkkitehtuureja

Seuraavaksi esitellään tunnetuimpia käyttöliittymän ja sovelluslogiikan eriyttämiseen pyrkiviä arkkitehtuureja. Lisäksi verrataan kolmannen luvun esimerkeissä tarkemmin esiteltävää MVC-mallia muihin artikkelissa esiteltäviin arkkitehtuureihin. Tässä artikkelissa käsitellyt arkkitehtuurit muodostavat vain peruskatsauksen käyttöliittymäarkkitehtuureihin. Todellisuudessa on olemassa muitakin merkittäviä arkkitehtuureita ja jotkut niistä saattavat olla pidemmälle kehittyneitä kuin tässä artikkelisssa esiteltävät arkkitehtuurit. Valintaperusteena on käytetty kuitenkin arkkitehtuurien yleisyyttä eikä tässä tapauksessa ole tutkittu harvinaisempia tai pidemmälle johonkin tarkoitukseen räätälöityjä arkkitehtuureja.

2.3.1 Seeheim

Seeheim-mallin historia juontaa juurensa vuonna 1983 Seeheimissä pidettyyn workshop-tapaamiseen, jossa esiteltiin abstrakti malli käyttöliittymän jaosta erillisiin komponentteihin [Edmonds, 1992]. Seeheim-malli perustuu

monoliittisiin kerroksiin, joista kullakin on oma rajattu tehtävänsä. Käyttäjä kommunikoi staattisen esitystapa-komponentin kanssa, josta käyttäjän syötteet ohjautuvat dialogi-komponenttiin. Dynaamisesti toimivan dialogi-komponentin pääasiallinen tehtävä on huolehtia tapahtumien käsittelystä ja päättää niiden perusteella malli-komponentilta pyydettävistä palveluista. Malli-komponentti hallitsee esitystapa-komponenttia, joka puolestaa toimii tulostekanavana käyttäjän suuntaan [Kurlander and Ling, 1995].



Kuva 1: Seeheim-malli [Dix, 2000].

Seeheim-mallin kehittäminen toi mukanaan uusia etuja sovelluskehittäjille. Näistä merkittävimpinä voidaan pitää erillisten kerrosten helpompaa ylläpidettävyyttä, ohjelmakoodin uudelleenkäytettävyyttä ja ohjelmointitehtävien jakamisen helpottumista useiden ohjelmoijien kesken [Dalmatian].

Seeheim-malli oli ensimmäinen merkittävä käyttöliittymän ja sovelluslogiikan eriyttämiseen pyrkivä käyttöliittymäarkkitehtuuri. Seeheim-malli ei ole vieläkään vanhentunut vaan toimii edelleen pohjana monille uudemmille arkkitehtuureille ja myös monet nykyaikaiset agenttipohjaiset arkkitehtuurit perustuvat osittain Seeheim-malliin [Calvary *et al.*, 1997]. Seeheim-malli jakaa sovelluksen vain yhdeksi sovellusmalliksi ja yhdeksi käyttöliittymäksi, joten yksi Seeheimin huonoista puolista on tuki monelle samanaikaiselle käyttäjälle. Myöhemmin tässä artikkelissa esiteltävät

agenttipohjaiset mallit tuovat paremman ratkaisun monen käyttäjän ohjelmille. Toisaalta ryhmäohjelmien tukeminen ja samanaikaisuuden mahdollistaminen ei ole Seeheim-mallin tarkoituskaan vaan Seeheim on kehitetty alun perin yhden käyttäjän sovelluksille [Phillips and Graham, 2000].

2.3.2 Arch

Arch-malli kehitettiin minimoimaan tulevaisuuden teknologiakehityksen vaikutukset [Calvary *et al.*, 1997] ja tukemaan entistä paremmin ohjelmakoodin uudelleenkäytettävyyttä. Arch-malli on Seeheim-mallin laajennus [Phillips and Graham, 2000], jossa eri komponenttien tehtäviä on jaettu uusiksi ja niiden välillä kulkeva tieto on tarkemmin määritelty. Arch-malli koostuu viidestä erillisestä komponentista. Phillipsin ja Grahamin [2000] mukaan sovelluskomponentti (Domain-Specific Component) muodostaa Arch-mallin kivijalan ja huolehtii sovelluksen datasta sekä siihen liittyvistä metodeista. Dialogikomponentin pääasiallinen tehtävä on tapahtumien käsittely ja vuorovaikutuskomponentti huolehtii syötteistä ja tulosteista. Arch malli sisältää myös kaksi ns. adapterikomponenttia, jotka pyrkivät yksinkertaistamaan pääasiallisia kerroksia ja mahdollistamaan niiden eristämisen toisistaan riippumattomiksi. Sovellusadapterikomponentti huolehtii niiden tehtävien toteuttamisesta, joita ei ole määritelty sovelluskomponentissa. Esitysadapterkomponentti sen sijaan tarjoaa dialogikomponentin käyttöön riippumattomia objekteja, joita voidaan käyttää tiedon esittämiseen.

Multimodaalisten käyttöliittymien tapauksessa Arch-malli ei ole paras mahdollinen ratkaisu, koska Arch-malli ei huolehdi tehtävien järjestämisestä dialogikomponentissaan [Saarinen, 2006]. Arch-malli on Seeheim-mallin tavoin kehitetty yhden käyttäjän sovelluksille, mutta Arch-mallia on toisaalta helppo laajentaa tukemaan myös ryhmäohjelmia ja multimodaalisia käyttöliittymiä [Phillips and Graham, 2000].

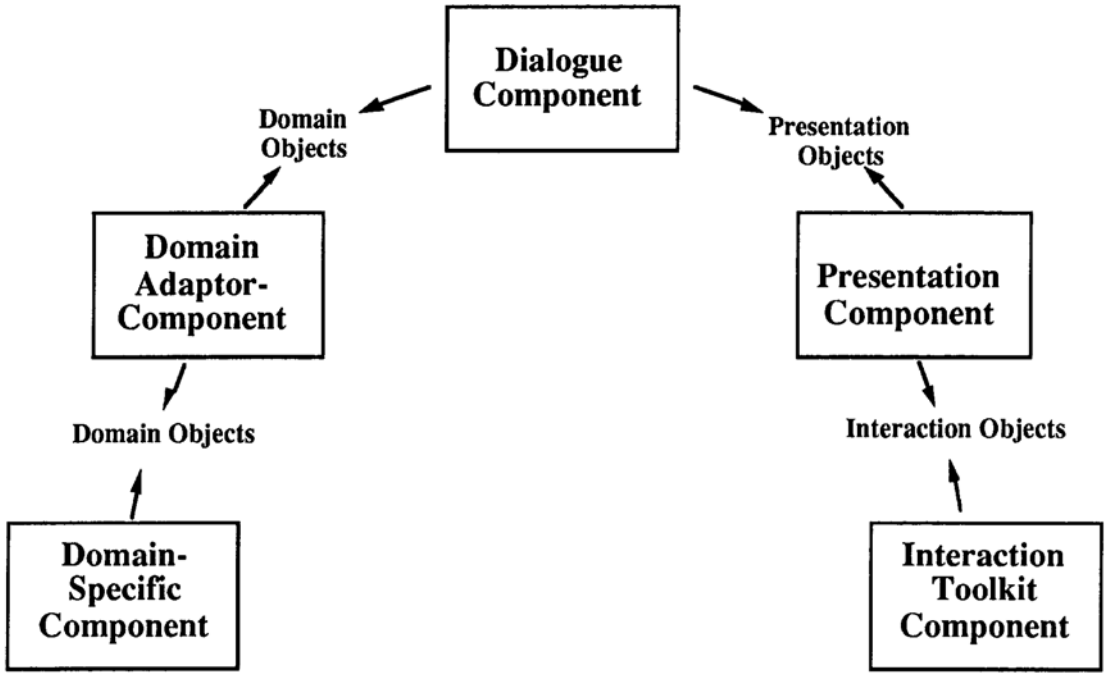


Figure 1. The interfaces between the components

Kuva 2: Arch-malli [Bass *et al.*, 2000].

2.3.3 MVC

Model-View-Controller (MVC) on yksi tämän hetken suosituimmista arkkitehtuuritason suunnittelumalleista. MVC-mallin esittelivät Smalltalk-ohjelmointikielen kehittäjät vuonna 1987 [Krasner and Pope, 1988]. MVC-malli on alun perin kehitetty graafisille WIMP-käyttöliittymille ja sen perusajatuksena on erottaa käyttöliittymä sovelluksen ydintoiminnallisuudesta sekä minimoida käyttöliittymän muutosten vaikutukset sovelluksen toimintaan [Tao, 2002]. Vaikka MVC-malli on alun perin suunniteltu perinteisten graafisten käyttöliittymien toteuttamiseen, sen sovellusalue on sittemmin laajentunut.

- Malli (Model) muodostaa sovelluksen ytimen, joka sisältää sovelluksen datan ja tärkeimmän toiminnallisuuden. Samaan malliin

on mahdollista yhdistää useita eri näkymä-ohjain-pareja. Mallin ei tarvitse tietää mitään siihen liitettävistä näkymä-ohjain-pareista.

- Näkymä (View) muodostaa varsinaisen käyttöliittymän, josta sovelluksen käyttäjä saa tulosteet. Näkymän tehtävä on esittää mallin sen hetkinen tila käyttäjän ymmärtämässä muodossa.
- Ohjain (Controller) tulkitsee käyttäjän antamat syötteet ja välittää tiedot halutuista muutoksista malliin. Ohjain on aina sidoksissa johonkin näkymään.

MVC-mallista on olemassa useita variaatioita eikä monikaan ohjelmointikehys noudata mallia täydellisesti. MVC-mallissa näkymistä ja ohjaimista puhutaan usein pareina, koska jokainen näkymä tarvitsee aina ohjaimen ja päinvastoin. Näkymä-ohjain-parien täytyy aina tietää täydellisesti mallin rakenne, jotta mallia voisi päivittää ja mallin lähettämiä ilmoituksia (callback) tulkita. Joissakin ohjelmointikehyksissä näkymä-ohjain-parit toteutetaan yhdistettyinä objekteina [Phillips and Graham, 2000].

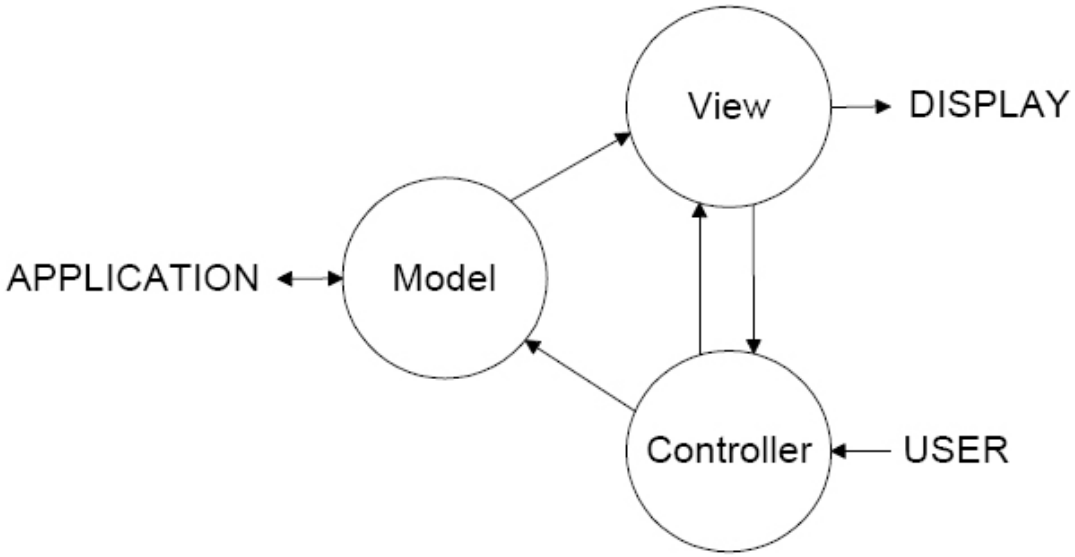


Figure 6: MVC model for interactive systems [5].

Kuva 3: MVC-malli [Dobos, 2002].

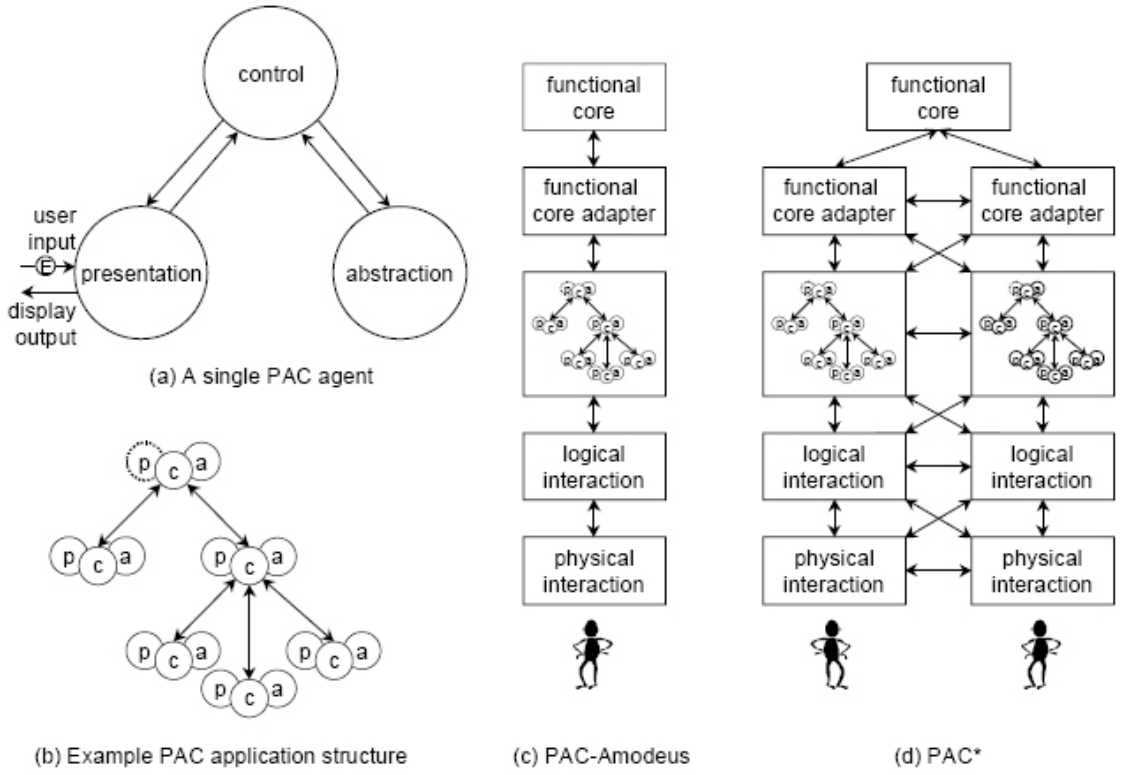
Suurimpia etuja on mallin sisältämän ydintoiminnallisuuden hyvä uudelleenkäytettävyys ja siirrettävyys toiselle alustalle. Lisäksi samasta mallista on mahdollista luoda useita näkymiä, minkä ansiosta yhden mallin ympärille voidaan rakentaa teoriassa rajaton määrä näkymä-ohjain-pareja. Mallejakin voi olla useita, mutta niiden välistä yhteyttä MVC-malli ei mitenkään määrittele [Phillips and Graham, 2000].

MVC-mallin yhtenä ongelmana pidetään sovellusten tehokkuutta, joka yleensä laskee modularisoinnin kasvaessa. Toinen puute on, että MVC tukee ainoastaan sovelluksen mallin (model) uudelleenhyödyntämistä [Tracz, 1995]. Muiden osien suunnittelu on aloitettava aivan tyhjästä jos siirretään malliosan sisältämä ydin esimerkiksi perinteisestä työpöytäsovelluksesta web-sovellukseen. Lisäksi mainitsemisen arvoinen ongelma on, että erityisesti Web-sovellusten alueella lukuisat ohjelmointikehykset toteuttavat MVC-mallin, mutta ne ovat yleisten standardien puuttuessa lähes poikkeuksetta jonkin yrityksen teknologiaan sidottuja [Barrett and Delany, 1995].

2.3.4 PAC

Presentation-Abstraction-Control (PAC) on monessa mielessä hyvin MVC-mallin kaltainen agenttiarkkitehtuuri. PAC on moniagenttimalli, jossa modularisointi on viety pidemmälle kuin MVC-mallissa. PAC-mallin agenteille on tyypillistä jokin tila ja jokin tehtävä jonka suorittamisesta ne vastaavat agenttihierarkiassa.

Yksi keskeinen ero MVC-malliin nähden on hierarkisen agenttirakenteen lisäksi syöte- ja tulostekäsittelyssä. PAC-mallissa molemmat tehtävät ovat esitystapa-moduulin tehtäviä, mutta MVC-mallissa ne on jaettu tiukasti erilleen ohjain- ja näkymä-moduulien välillä [Calvary *et al.*, 1997].



The PAC family of architectural styles.

Kuva 4: PAC-malli ja sen variaatiot [Phillips and Graham, 2000].

PAC-mallista on sittemmin kehitetty myös hybridiversio PAC-Amodeus, joka käyttää pohjanaan Arch-mallin komponentteja, mutta sijoittaa PAC-mallin agenttirakenteen Arch-mallin dialogikomponenttiin. Etuina on tuki samanaikaisuudelle ja näin ollen PAC-Amodeus on erinomainen arkkitehtuuri multimodaalisten käyttöliittymien kannalta [Nigay *et al.*, 1995]. Suunnittelijan vastuulle jää kuitenkin agenttien välisestä vuorovaikutuksesta huolehtiminen. PAC-Amodeus malli esitellään perusteellisemmin Saarisen [2006] artikkelissa.

PAC-Amodeuksestakin on kehitetty uusia variaatioita, joista tunnetuin lienee ryhmäohjelmia varten ideoitu PAC* [Calvary *et al.*, 1997], jossa jokaisen agentin sisällä eritellään kolme toimintoa. PAC*-agentin jakaminen tuotanto, kommunikaatio ja koordinaatiotoimintoihin yksinkertaistaa eri käyttäjien välisen vuorovaikutuksen toteuttamista ryhmäohjelmissa.

3. MVC-malli eri tyyppisissä käyttöliittymissä

3.1. MVC ja multimodaaliset käyttöliittymät

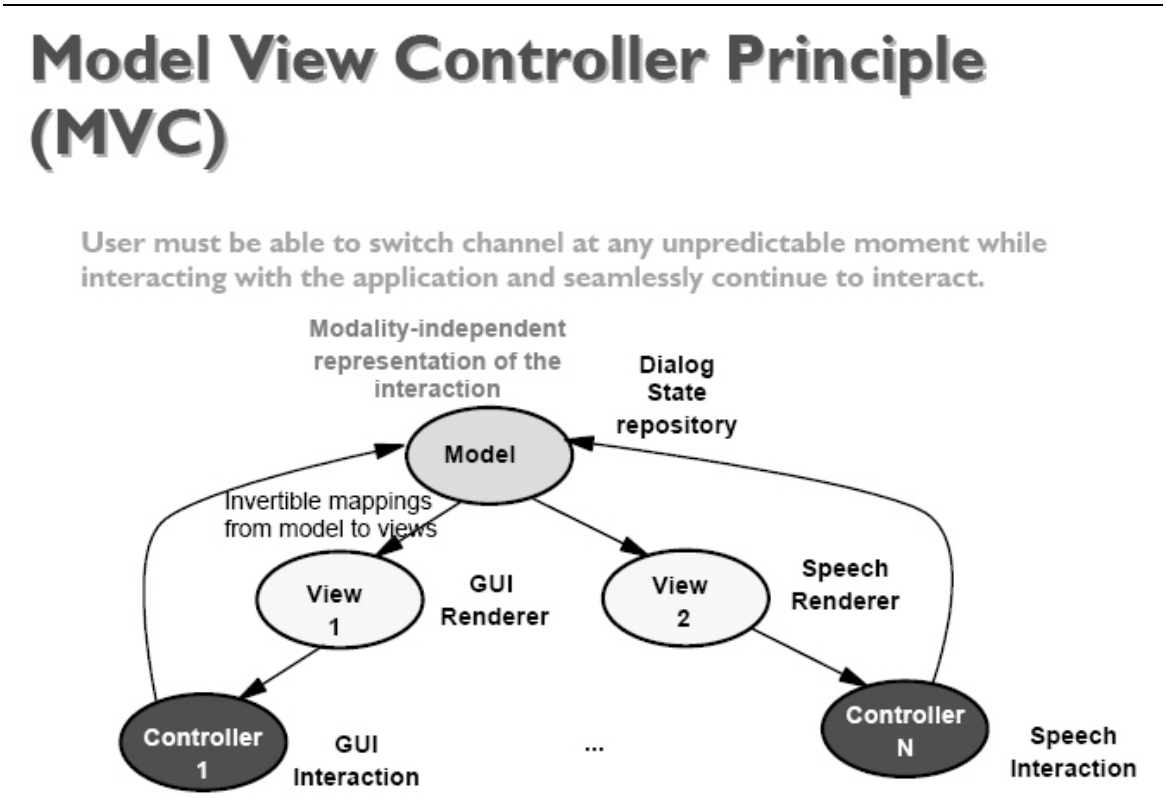
MVC-malli sopii tietyin rajoituksin myös multimodaalisten käyttöliittymien toteuttamiseen, koska arkkitehtuuri tukee useiden näkymien samanaikaista olemassaoloa. Käytännössä se tarkoittaa, että samaa ydintoiminnot ja datan sisältävää mallia voidaan hallita käyttäjän valinnan mukaan eri aistikanaviin perustuvilla käyttöliittymillä. Vaikka MVC-malli on suunniteltu alun perin graafisille WIMP-käyttöliittymille, sillä voidaan saada hyviä tuloksia aikaan myös moderneja käyttöliittymiä suunniteltaessa. Barnett *et al.* [2005] mukaan MVC-mallin vahvat puolet tulevat erityisen hyvin esille multimodaalisia käyttöliittymiä sisältävissä sovelluksissa. Toisaalta negatiivisena puolena voidaan pitää sitä, että MVC-mallia käytettäessä ei voida saavuttaa täydellistä samanaikaisuutta, koska se ei tue useita samanaikaisesti aktiivisia ohjaimia [Krasner and Pope, 1988]. Ongelmallinen tilanne voi syntyä, jos esimerkiksi useampi näkymä-ohjain-alijärjestelmä pyrkii kirjoittamaan samaa ydintoimintomallin dataa yhtä aikaa.

Ydintoiminnot sisältävän mallin täydellisen riippumattomuuden saavuttamiseksi kaikkien eri modaliteettien (näkymä-ohjain-alijärjestelmien) pitäisi kommunikoida ydinmallin kanssa yhteisellä kielellä, jotta malli ei ole riippuvainen käytetystä modaliteetista ja mallin toteuttama ohjelmointirajapinta mahdollistaisi uusien modaliteettien käyttöönoton ilman mallin muokkaamista. MVC-malli ei rajoita lainkaan sovellukseen kuuluvien näkymien ja ohjainten määrää, joten teoriassa MVC-mallin mukaisella sovelluksella voi olla eri syöte- ja tulostemuotoja rajoittamaton määrä.

PAC-Amodeus-malli on kiistatta MVC-mallia parempi vaihtoehto monimutkaisten, jatkuvia ja samanaikaisia syötevirtoja sisältävien multimodaalisten käyttöliittymien suunnittelussa, mutta yksinkertaisemmissa multimodaalisissa sovelluksissa MVC-malli saattaa olla helpompi ratkaisu. Saarinen [2005] kirjoittaa, että PAC-Amodeus kannattanee ottaa käyttöön vasta isoissa järjestelmissä, koska sen toteuttaminen on aikaavievää ja vaativaa.

MVC-malli soveltunee parhaiten sellaisiin multimodaalisiin sovelluksiin, joissa syöte- ja tulostevirtoja ei käytetä yleensä samaan aikaan. Esimerkkinä on matkavarauspalvelu [Maes and Ferial, 2002], jota voi käyttää perinteisellä tietokoneella, wap-laitteella tai puhekäyttöliittymällä. Kukin modaliteetti muodostaa oman MVC-mallin mukaisen näkymä-ohjain-parin, ja kaikki

käyttävät samaa datan ja ydintoiminnot sisältävää mallia. Seuraava kaavio esittää MVC-malliin perustuvan kaksi syöte- ja tulostekanavaa sisältävän sovelluksen rakenteen.



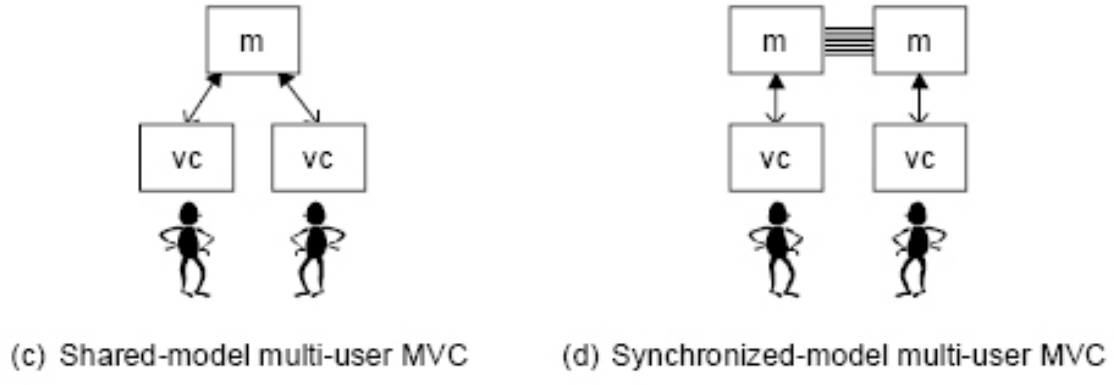
Kuva 5: Multimodaalisuutta tukeva MVC-malli [Maes and Ferial, 2002].

MVC-malli tukee eri modaliteetteja periaatteessa yhtä hyvin kuin kehittyneemmät agenttiarkkitehtuurit, mutta heikkoudet tulevat esille kun useita modaliteetteja käytetään samanaikaisesti. MVC-mallin sovellusalue multimodaalisissa käyttöliittymissä on siis rajallinen, koska ideaalisen multimodaalisen käyttöliittymän voima perustuu siihen, että monia modaliteetteja voisi käyttää yhtä aikaa eikä yhden käyttäminen lukitsisi muita modaliteetteja. MVC-mallin suurimpia vahvuuksia puolestaan on, että se mahdollistaa PAC-Amodeus mallia vähemmällä työmäärällä omien räätälöityjen versioiden tekemisen samasta käyttöliittymästä eri ihmisten tarpeisiin. Samasta sovelluksesta voidaan tehdä esimerkiksi visuaalinen versio kuulovammaisille ja auditiivinen versio näkövammaisille.

3.2. MVC ja ryhmäohjelmat

Ryhmäohjelmat (Groupware) ovat sovelluksia, joilla voi olla useita samanaikaisia käyttäjiä. Ryhmäohjelmat ovat toteutukseltaan hyvin lähellä multimodaalisen käyttöliittymän sisältäviä yhden käyttäjän sovelluksia. Suurimpana erona on se, että ryhmäohjelmissa useimmiten sovelluksen näkymät ovat fyysisesti kaukana toisistaan ja tulosteiden sisältöön saattaa liittyä erilaisia käyttäjäoikeuksia. Lisäksi ryhmäohjelmilta vaaditaan syötteiden käsittelyssä enemmän, koska käyttäjät eivät osaa lukea toistensa ajatuksia. Multimodaalisissa yhden käyttäjän sovelluksissa käyttäjä harvoin antaa tahattomasti täysin ristiriitaisia syötteitä eri modaliteettikanavia pitkin. MVC-mallin pohjalta on mahdollista toteuttaa ryhmäohjelmia yhdistämällä useita ohjain-näkymä-pareja jokaiseen ydintoiminnot sisältävään malliin, joka pitäisi jakaa usealle käyttäjälle [Phillips and Graham, 2000].

Vaikka tiettyjä ongelmia sisältyy esimerkiksi samanaikaisuuden toteuttamiseen, MVC-malli tarjoaa ohjelmistosuunnittelijalle yksinkertaisen lähestymistavan ryhmäohjelmien kehittämiseen. Kompastuskivenä on usein myös verkon kapasiteetti, mutta sama ongelma tulee yleensä vastaan enemmin tai myöhemmin myös muita suunnittelumalleja käytettäessä. Graham *et al.* [1996] totesi artikkelissaan, että erilaisilla optimointialgoritmeilla on mahdollista parantaa tehokkuutta riittävälle tasolle ja tehdä MVC-mallin mukaisten ryhmäohjelmien suunnittelusta mielekästä. Verkkoliikenteen nopeuden kehittymisen myötä kyseiset esteet ovat varmasti vähentyneet artikkelin ilmestymisen jälkeen, mutta toisaalta myös sovellusten monipuolisuus ja verkkokaistavaativuus ovat jatkuvasti kasvaneet. Ryhmäohjelmien kohdalla tietoverkkojen nopeusongelmasta tuskin voidaan päästä kokonaan eroon ainakaan lähituleivaisuudessa.



Kuva 6: MVC-malli ryhmäohjelmissa [Phillips and Graham, 2000].

MVC-mallin avulla voi toteuttaa ryhmäohjelmia kahdella eri lähestymistavalla. Kuvassa 6 on ensimmäisenä vaihtoehtona esitetty keskitettyyn malliin perustuva ryhmäohjelmatoteutus. Mallin keskittäminen lienee helpoin tapa toteuttaa sovelluksia. Lisäksi kuvassa 6 on esitetty synkronoitu malli, josta jokaisella käyttäjällä on identtinen kopio. Synkronoidun mallin tapauksessa identtisen kopioiden ylläpitoon tarvitaan jotain protokollaa [Patterson, 1995] ja tämä tekee toteutuksesta lähes väistämättä hankalamman.

Hyvä esimerkki ryhmäohjelma-arkkitehtuurista on RendezVous, joka on kehitetty synkronoitujen ryhmäohjelmien toteuttamiseen ja sisältää tuen monen käyttäjän sessiolle sekä syöte- ja tulostetoiminnoille. RendezVous voidaan toteuttaa MVC-mallin mukaisesti. RendezVous-arkkitehtuuri on alun perin suunniteltu mallintamaan sovelluksia, jotka sallivat useamman käyttäjän manipuloivan samaa tietoa samanaikaisesti. Ryhmäohjelmissa sovelluksen osien jakaminen usealle eri käyttäjälle ei itsessään ole monimutkaista, mutta tällaisissa sovelluksissa kontrollille asetetaan tiettyjä vaatimuksia. Patterson *et al.* [1990] ovat sitä mieltä, että ryhmäohjelma on kuin mikä tahansa ohjelma, mutta sisältää enemmän hallittavia syöte- ja tulostekanavia.

Esimerkkinä MVC-malliin perustuvasta RendezVous-arkkitehtuurista on tässä tapauksessa usean käyttäjän korttipeli. Sovelluksessa on vain yksi malli, joka sisältää kaiken datan, mutta jokaisella pelaajalla on omat näkymänsä tästä mallista. Kaikki pelaajat voivat nähdä julkiset pöydässä olevat kortit ja omat korttinsa, mutta näkymä ei anna pelaajan nähdä toisten pelaajien kortteja. Tämän

kaltaisissa ryhmäohjelmasovelluksissa keskeisin ero yhden käyttäjän multimodaalisiin sovelluksiin on toiminta- ja näkymäoikeuksista huolehtiminen [Patterson *et al.*, 1990].

RendezVous-arkkitehtuurin yleisin toteutustapa lienee Abstraktio-Linkki-Näkymä-malli (Abstraction, Link, View), mutta RendezVous on mahdollista toteuttaa myös MVC-malliin perustuen [Hill, 1992]. Abstraktio-Linkki-Näkymä mallissa abstraktioon on tallennettu tieto sovelluksen tilasta, näkymän tehtävä on esittää tieto ja linkin tehtävä on tiedon välitys ohjelman ja käyttöliittymän välillä [Rantala, 2006].

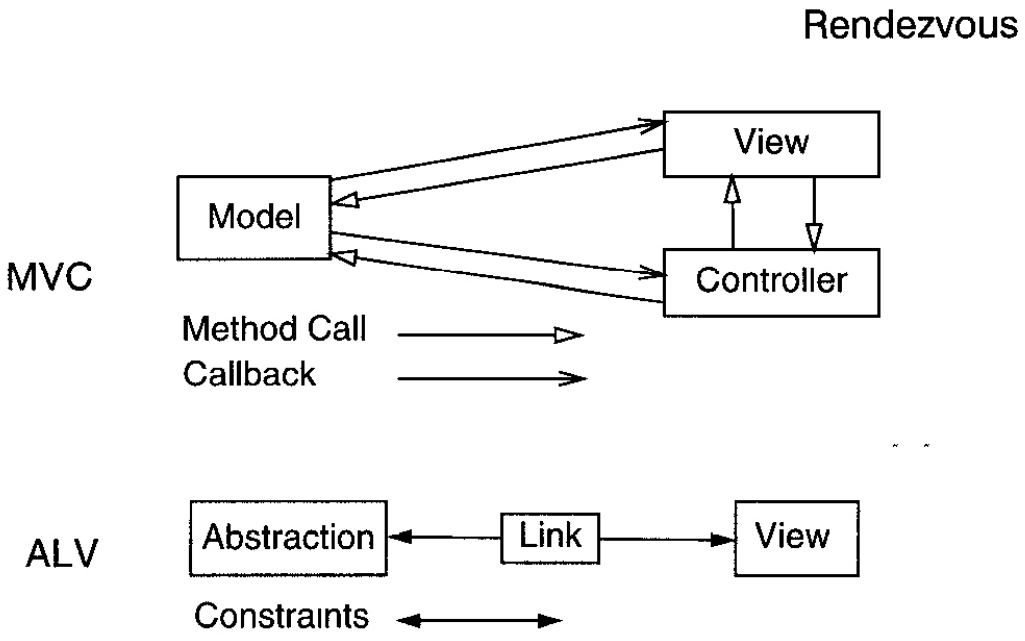


Fig. 10. MVC vs. ALV architecture.

Kuva 7: Rendezvous-arkkitehtuurin toteutusmallien vertailu [Hill *et al.*, 1994].

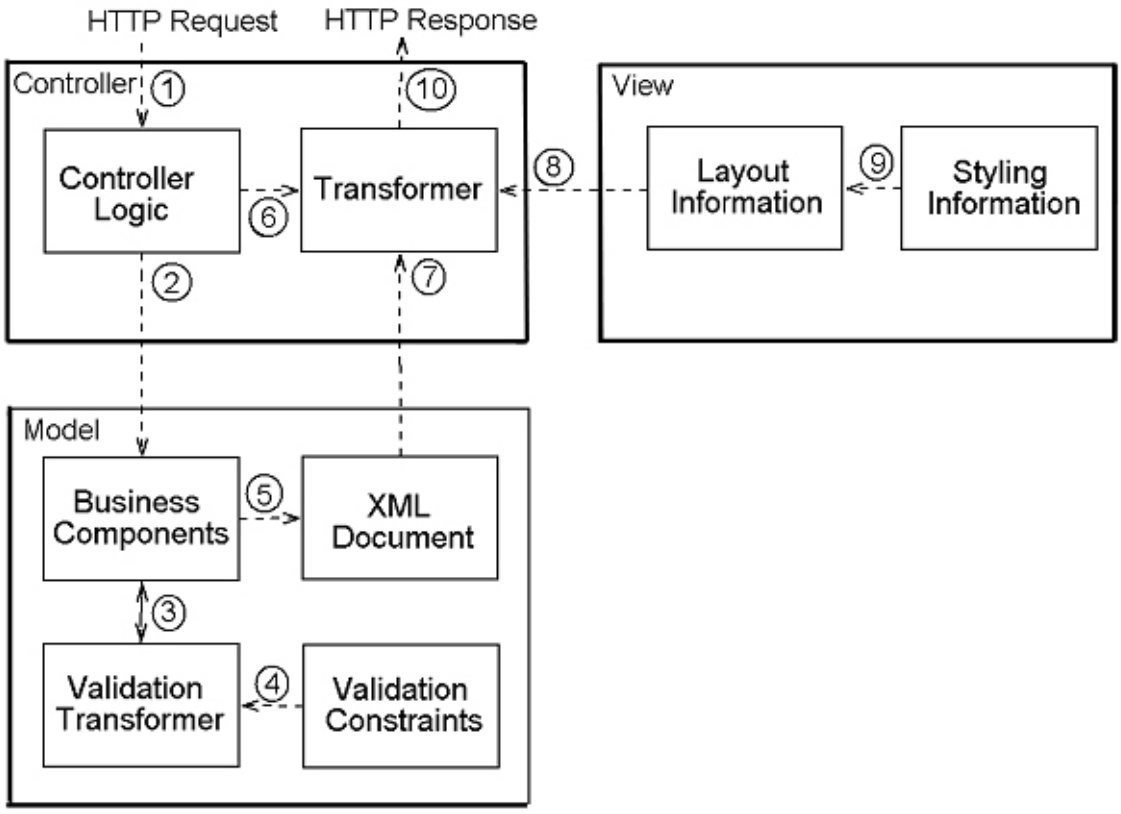
MVC-mallin haittapuolena näkymän ja ohjaimen täytyy tietää mallin sisäinen rakenne, jotta muutoksiin voitaisiin reagoida. Lisäksi mallin lähettämien ilmoitusten (Callbacks) tulee olla kiinteästi ohjelmoitu kaikkiin kommunikaatiokomponentteihin. MVC-malli mahdollistaa RendezVous-arkkitehtuurin toteutuksen, mutta Abstraktio-Linkki-Näkymä on yleisesti

mielletty kokonaisuudeltaan helpommaksi tavaksi toteuttaa monimutkaisia ryhmäohjelmia [Hill *et al.*, 1994].

3.3. OpenMVC

MVC on tähän asti ollut yleinen arkkitehtuuritason suunnittelumalli, jota lukuisat eri ohjelmointikehykset ovat toteuttaneet parhaaksi katsomallaan tavalla. Tällä hetkellä moni web-käyttöön tarkoitettu MVC-mallin mukainen toteutus on erittäin riippuvainen jostakin kaupallisesta teknologiasta ja standardien puute vaikeuttaa pitkäjänteistä suunnittelua.

Barrettin ja Delanyn [2004] ideoima OpenMVC on hahmotelma web-sovellusten kehittämiseen tarkoitetusta MVC-mallin evoluutioversiosta, jossa pyritään käyttämään avoimia W3C:n määrittelemiä standardeja. OpenMVC on viisikerroksinen arkkitehtuuri, joka koostuu asiakaskerroksesta, näkymäkerroksesta, sovelluslogiikkakerroksesta, data-abstraktiokerroksesta ja tietokantakerroksesta. OpenMVC modularisoi MVC-mallia entistä pienemmiksi osiksi siten, että esimerkiksi tyylit ja käyttöliittymän asettelu ovat erillisiä komponenttejaan näkymä-alijärjestelmän sisällä. Ohjain-alijärjestelmässä on normaaliin MVC-malliin verrattuna lisäkomponentti, joka huolehtii malli-alijärjestelmästä tulevien XML-dokumenttien muokkaamisesta XHTML-muotoon WWW-selainta varten käyttäen näkymä-alijärjestelmän tyylejä ja asettelua apunaan. Lisäksi ydintoiminnot sisältävän mallin rakenne on erilainen kuin perinteisessä MVC-mallissa, koska se sisältää perinteisen sovelluslogiikkakomponentin lisäksi validaatiosääntökomponentin, joka huolehtii huolehtii datan oikeellisuudesta ja yhtenäisyydestä asetettujen sääntöjen mukaisesti. Toinen normaaliin MVC-malliin nähden ylimääräinen lisäkomponentti malli-alijärjestelmässä on validaatiomuuntajakomponentti, joka muokkaa datan validaatiosääntökomponentin edellyttämään muotoon.



Kuva 8: OpenMVC-malli [Barrett and Delany, 2004].

Barrett ja Delany [2004] arvioivat OpenMVC:n suurimmiksi eduiksi tyyli-, asettelu- ja validointisääntöjen (validation constraints) ylläpidon ilman ohjelmakoodin muokkausta tai uudelleen kääntämistä. Tämä helpottaa huomattavasti esimerkiksi käyttäjäkohtaista räätälöintiä jos samaa sovellusta halutaan tarjota eri käyttäjille, joilla on kullakin omat toiveensa sovelluksen ulkonäön suhteen. OpenMVC:n käyttämät avoimet teknologiat parantavat lisäksi teoriassa sovelluksen siirrettävyyttä ohjelmointikehykseltä toiselle, mutta nekään eivät kuitenkaan ratkaise kaikkia ongelmia. Sovelluksen siirtäminen kahden alustan välillä tarkoittaisi kuitenkin sovelluslogiikan uudelleenkirjoitusta jollain toisella ohjelmointikielellä. OpenMVC:n mukanaan tuoma suurempi modulaarisuusaste on varmasti eduksi monimutkaisissa sovelluksissa, mutta pienempiä sovelluksia tehtäessä OpenMVC saattaa tehdä toteutuksesta jopa hitaampaa. Suuren määrän komponentteja sisältävät arkkitehtuurit vaativat yleensä suhteellisen paljon opettelua ennen kuin niiden käyttö nopeuttaa ohjelmointityötä.

OpenMVC on mielestäni kuitenkin oikeilla jäljillä tulevaisuuden arkkitehtuureja kehitettäessä. Vaikka kyseessä on enemmänkin hahmotelma standardista kuin laajalti käytetty suunnittelumalli, OpenMVC on arkkitehtuurina hyvin perusteltu ja sopisi varmasti myös käytännön ohjelmistokehitykseen. Toisaalta OpenMVC ei tuo käytännön ohjelmistokehitykseen mitä uutta, koska moni ohjelmointikehys on jo pitkään hyödyntänyt esimerkiksi XML-dataa ja riippumattomia tyylitiedostoja.

4. Yhteenveto ja katsaus tulevaisuudesta

Ohjelmistokehityksessä modulaarisuus on jatkuvasti lisääntynyt ja sovelluksia on pilkottu enemmän ja enemmän loogisiin osiin. Tarjolla on tällä hetkellä useita kilpailukykyisiä arkkitehtuurivaihtoehtoja, mutta jokaisella niistä on yleensä myös huonoja puolia. Agenttiarkkitehtuurit tulevat varmasti vielä pitkään olemaan käyttöliittymäsuunnittelussa keskeisessä asemassa ja niiden syrjäyttäjästä ei ole vielä havaintoja.

Tässä artikkelissa tarkimmin käsitellystä MVC-mallista voisi yhteenvetona sanoa, että se on toimivaksi testattu ja luotettava suunnittelumalli, mutta se on oikea ratkaisu vain osaan sovelluksista. MVC-mallin aika ei ole missään nimessä ohi, koska siihen perustuvia sovelluksia voidaan toteuttaa, ylläpitää ja uudelleenhyödyntää verraten yksinkertaisesti. Toisaalta MVC-mallin sovellusaluetta ei tulisi väkisin yrittää laajentaa liian monimutkaisiin sovelluksiin. Esimerkiksi PAC-malli variaatioineen mahdollistaa käyttöliittymien kannalta vaativampien sovellusten suunnittelun, mutta toisaalta sovellukset ovat yleensä myös huomattavasti vaikeampia toteuttaa. MVC-mallin vahvuudet tulevat parhaiten esille yhdelle käyttäjälle tarkoitetuissa, versiopohjaisissa ja jatkuvasti kehittyvissä sovelluksissa. Uudentyyppisten käyttöliittymien osalta MVC saattaa kuitenkin jäädä kehittyneempien agenttipohjaisten hajautusarkkitehtuurien varjoon.

Tulevaisuudessa arkkitehtuurien määrä tulee varmasti kasvamaan huomattavasti ja erityisesti multimodaalisten käyttöliittymien osalta tällä hetkellä ollaan vasta kehityksen alussa. Luultavasti tulevaisuudessakaan mikään arkkitehtuuri ei voi kuitenkaan olla kaikissa käyttötapauksissa paras vaihtoehto eikä täyttää kaikki odotuksia. Ennemminkin voisi odottaa, että jatkossa tullaan näkemään lukuisia tiettyyn tarkoitukseen räätälöityjä arkkitehtuureita. Arkkitehtuurit eivät silti yksistään takaa kehitystä vaan myös ohjelmointikielten

ja tietoverkkoteknologian on kehityttävä, jotta voitaisiin saada käytännön tuloksia aikaan.

Viiteluettelo.

[Barnett et al., 2005] Jim Barnett, Jonny Axelsson, Michael Bodell, Brad Porter, Dave Raggett, Andrew Wahbe, Multimodal Architecture and Interfaces. World Wide Web Consortium, http://www.w3.org/TR/mmi-arch/, viitattu 16.11.2005.

[Barrett and Delany, 2004] Ronan Barrett and Sarah Jane Delany, OpenMVC: a non-proprietry component-based framework for web applications. Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, ACM Press, 2004, 464–465.

[Bass et al., 1992] Len Bass, Ross Faneuf, Reed Little, Niels Mayer, Bob Pellegrino, Scott Reed, Robert Seacord, Sylvia Sheppard, Martha R. Szczur, A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop, ACM SIGCHI Bulletin, ACM Press, 24, 1, Jan. 1992, 32-37.

[Calvary et al., 1997] Gaëlle Calvary, Joëlle Coutaz, Laurence Nigay, From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. ACM Press, 1997, 242–249.

[Dalmatian] TeleUSE technical brief, The Dalmatian Group, http://www.dalmatian.com/teleuse.htm, viitattu 21.11.2005.

[Dix, 2000] Alan Dix, Implementation and architecture support. Lancaster University, Computing Department, 2000.

[Dobos, 2002] Hanna Dobos, Separable User Interfaces Interaction Controls. University of Jyväskylä, Department of Mathematical Information Technology, Nov. 2002.

[Edmonds, 1992] E. Edmonds, The Emergence of the Separable User Interface. The Separable User Interface, Academic Press, Nov. 1992, 5-18.

[Graham et al., 1996] Nicholas Graham, Tore Urnes, Roy Nejabi, Efficient distributed implementation of semi-replicated synchronous groupware. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'96), ACM Press, Nov. 1996, 1-10.

[Hill, 1992] R. D. Hill, The Abstraction-Link-View paradigm: using constraints to connect user inretfaces to applications. Proceedings of the SIGCHI

conference on Human factors in computing systems, ACM Press, 1992, 335-342.

[Hill et al., 1994], Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson, Wayne Wilner, The Rendezvous architecture and language for constructing multiuser applications. ACM Transactions on Computer-Human Interaction (TOCHI), ACM Press, 1, 2, Jun. 1994, 81–125.

[Krasner and Pope, 1988] G. Krasner, S. Pope, A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1, 3, Sep 1988, 26-49.

[Kurlander and Ling, 1995] David Kurlander, Daniel T. Ling, Planning-Based Control of Interface Animation. CHI' 95 proceedings, 1995.

[Maes and Ferial, 2002] Stephane H. Maes, Chummun Ferial, Multi-Modal Browser Architecture – Overview on the support of multi-modal browsers in 3GPP, World Wide Web Consortium, http://www.w3.org/2002/mmi/2002/MM-Arch-Maes-20010820.pdf, viitattu 16.11.2005.

[Nigay et al., 1995] Laurence Nigay, Daniel Salber, Simon Buckingham, Shum & Joelle Coutaz, Teaching Trainee and Professional Designers to use the PAC-AMODEUS Software Architecture Modelling Technique. The Amodeus Project, Jun. 1995.

[Patterson et al., 1990] John F. Patterson, Ralph D. Hill, Steven L. Rohall, Scott W. Meeks, Rendezvous: An architecture for synchronous multi-user applications. Proc. Conf. Computer-Supported Cooperative Work (CSCW'90,), Oct. 1990, ACM Press, 317-328.

[Patterson, 1995] John Patterson, A taxonomy of architectures for synchronous groupware applications. ACM SIGOIS Bulletin, ACM Press, 15, 3, Apr. 1995, 27-29.

[Phillips and Graham, 2000] Greg Phillips, Nicholas Graham, Software Architectures for Multiuser Interactive Systems. Univesity of Calgary, 2000.

[Rantala, 2006] Jussi Rantala, Jaettujen editorien arkkitehtuurit, Ilmestyy raportissa Roope Raisamo (toim.), Käyttöliittymien ohjelmistoarkkitehtuurit, Tampereen yliopisto, Tietojenkäsittelytieteiden laitos, 2006.

[Saarinen, 2006] Rami Saarinen, "Multimodaalisuus, PAC-Amodeus ja moniagenttijärjestelmät". Ilmestyy raportissa Roope Raisamo (toim.),

Käyttöliittymien ohjelmistoarkkitehtuurit, Tampereen yliopisto, Tietojenkäsittelytieteiden laitos, 2006.

[Tao, 2002] Yonglei Tao, Component- Vs. Application-level MVC Architecture, 32nd Frontiers in Education Conference, Nov. 2002.

[Tracz, 1995] Will Tracz, Third International Conference on Software Reuse Summary. ACM SIGSOFT Software Engineering Notes, 20, 2, Apr. 1995, 21-22.

Rakenteelliset kielet käyttöliittymän määrittämisessä

Ville Parviainen

Tiivistelmä.

Tässä paperissa esitellään tällä hetkellä suuren kiinnostuksen alla olevaa tutkimus aluetta, jonka pohjimmaisena tarkoituksena on kehittää käyttöliittymää yleisellä korkealla tasolla kuvaava kieli. Tällaisista kielistä käytetään lyhennettä UIDL (User Interface Description Language). Paperissa esitellään yleisellä tasolla (menemättä kovinkaan tarkkoihin yksityiskohtiin) neljä UIDL- perusteista kieltä. Näiden kielten ominaisuuksia myös verrataan keskenään ja yritetään löytää kielille sekä yhteneväisyyksiä, että eroavaisuuksia.

Avainsanat ja -sanonnat: Rakenteelliset kielet, XML, UIDL, käyttöliittymä

1. Johdanto

Käyttöliittymän määritteleminen rakenteellisella korkeamman abstraktisuus tason omaavalla kielellä, tuo käyttöliittymä kehitykseen paljon uusia ulottuvuuksia. Käyttöliittymä kehittäjän ei enää tarvitse välttämättä tuntea tai hallita ohjelmointikieliä, vaan hän voi keskittyä käyttöliittymän suunnitteluun. Lisäksi käyttöliittymä saadaan helposti alustariippumattomaksi. Samaa käyttöliittymäkuvausta voidaan siis ajaa usealla eri alustalla, ilman muutoksia varsinaiseen ns. lähdekoodiin.

Tällaisten käyttöliittymää kuvaavien kielten kehittelyyn on suunnattu huomiota jo ihmisen ja tietokoneen vuorovaikutustutkimuksen (HCI) alkuajoista lähtien. Viime aikoina kielten kehitteleminen ja yleisen standardin löytäminen on kuitenkin ollut varsinaista kilpajuoksua (jossa kilpailu on kuitenkin vielä kesken).

Puhuttaessa käyttöliittymää kuvaavista kielistä on yleisesti otettu käyttöön lyhenne UIDL-kielet (User Interface Description Language). Tällaisen kielen kehittämisen perus lähtökohtina voisivat olla seuraavanlaiset päämäärät [USIXML, 2004].

- Saavutetaan siirrettävä käyttöliittymä, joka toimii useissa erityyppisissä alustoissa, ilman suuria muutoksia itse määrityksiin.
- Pystytään kuvaamaan käyttöliittymä sellaisella abstraktilla tasolla, jolla taataan käyttöliittymän säilyminen vakaana vaikka muu maailma ympärillä muuttuisikin.
- Parannetaan uudelleen käytettävyyttä. Samantyyppisiä käyttöliittymiä ei tarvitse suunnitella aina uudestaan vaan voidaan muokata valmista käyttöliittymää uusien tarpeiden mukaiseksi.
- Voidaan tehdä vain yksi käyttöliittymäkuvaus(rakenne), jota hyödynnetään usealla eri alustalla tai laitteella (PC, matkapuhelimet, PDA, jne.).
- Tuetaan käyttöliittymän mukautumista erilaisiin käyttötarkoituksiin.
- Korkeantason kuvauksesta voidaan automaattisesti generoida varsinainen käyttöliittymä koodi. Tämä johtaa myös ns. tulkkien käyttöön, jolloin korkean tason koodia voidaan ajaa suoraan tulkkien (interpreter) avulla ilman varsinaista näkyvää käännöstä konekielelle.

Kilpajuoksu tällaisen yleisen käyttöliittymää kuvaavan kielen kehittämiseksi vietti vuosia varsin hiljaista eloa. Kuitenkin erityyppisten rakenteellisten merkintäkielien suuri saatavuus viimevuosina, sekä XML-metamallin [XML, 2005] tuoma mahdollisuus esitellä täysin uusia kieliä, on herättänyt kilpajuoksijat jälleen radoilleen.

Tässä paperissa esittelen neljä rakenteellista XML-perustaista [XML, 2005] UIDL-kieltä. Käytännössä UIDL-kieliä on kehitelty useita kymmeniä, sekä uusia kehitetään kaikenaikaa. Tähän paperiin olen esiteltävät kielet valinnut niistä löytämieni viitteiden perusteella. Hakiessani tietoa tähän paperiin esitellyt neljä kieltä osoittautuivat useimmin viitatuiksi, joten olen tehnyt jokseenkin epätieteellisen johtopäätöksen niiden suosiosta. Toinen paperissa esiteltyjen kielten valintaan vaikuttanut peruste on peräisin Souchonin ja Vanderdonckt tutkimuksesta, jossa on vertailtu useita UIDL-kieliä ja niiden ominaisuuksia [Souchin et al., 2003].

Luvussa kaksi on esitelty hyvin yleisellä tasolla, yllämainituin perustein valitut neljä kieltä. Luvussa kolme olen pyrkinyt löytämään yhteisiä ja eroavia tekijöitä esiteltyjen kielten suhteen. Viimeisessä luvussa (luvussa 4) pyritään summaamaan yhteen missä kohtaa kilpajuoksussa nykyisellään ollaan menossa.

2. Rakenteellisia käyttöliittymän kuvaus kieliä

Tässä kappaleessa esittelen lyhyesti neljä rakenteellista (XML-pohjaista) kieltä, joiden avulla voidaan määritellä käyttöliittymiä. Kolme ensimmäistä kielimallia pyrkivät suurimmilta osiltaan toteuttamaan UIDL-kielille johdannossa osoitettuja vaatimuksia. Viimeinen (kohdassa 2.4) esitelty kieli XUL, ei täysin ole UIDL-vaatimusten mukainen, mutta se on otettu mukaan tähän paperiin tuomaan vertailupohjaa hieman alemman abstraktisuustason omaavien kielten ja UIDL-kielten välillä.

2.1. UIML

User Interface Markup Language (UIML) on XML-pohjainen kieli, jonka avulla voidaan määritellä käyttöliittymiä eri laitteille ja sovellusalustoille. UIML ei varsinaisesti ole alustariippumaton kieli vaikka sen avulla voidaankin kuvata käyttöliittymiä erityyppisille laitteille ja sovellusalustoille. Käytännössä kuitenkaan yhdellä määrityksellä ei voida luoda käyttöliittymää usealle alustalle, vaan jokaiselle alustalle joudutaan käyttöliittymä määrittelemään alustakohtaisesti.

Käyttöliittymän määritelmä UIML-kielen mukaan, koostuu joukosta käyttöliittymä elementtejä. Jokaiselle elementille voidaan määritellä kolme ominaisuutta. Nämä ovat lueteltu seuraavassa.

- **Ulkoasun**, miltä elementti näyttää (värit, fontit, jne.).
- **Sisältö,** varsinainen ohjelman esittämä sisältö(teksti, kuva, jne.).
- **Vuorovaikutus toiminnot,** mahdolliset käyttäjän ja elementin väliseen vuorovaikutukseen liittyvät tapahtumat ja toiminnot(input tapahtumat ja output actionit).

Kieli on suunniteltu varsin yksinkertaiseksi käyttää. Se sisältää vain pari kymmentä eri tagia, tällöin kielen hallinta helpottuu. Lisäksi kieli on suoranaisesti täysin sitoutumaton mihinkään erityiseen sovellusalusta tyyppiin. UIML sallii siis käännöksen mille tahansa kohdekielelle (esim. Java, C, HTML, VoiceXML), joten sen avulla voidaan määritellä käyttöliittymiä perinteisestä WIMP käyttöliittymästä aina puhekäyttöliittymiin. Kuten jo aikaisemmin todettiin, ei UIML:lä kuitenkaan voida luoda erityyppisiä käyttöliittymiä saman määritelmän pohjalta. Toisin sanoen esimerkiksi puhekäyttöliittymälle ja WIMP käyttöliittymälle joudutaan tekemään omat UIML-pohjaiset määritykset vaikka takana olisikin sama sovelluslogiikka. Tämä tietenkin rajoittaa uudelleen

käytettävyyttä. Seuraavassa kuvassa on esitelty yksinkertainen HelloWorld käyttöliittymä käyttäen UIML-kieltä.

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 2.0a Draft//EN"
    "http://uiml.org/dtds/UIML2_0a.dtd">

<uiml>
  <interface>
    <structure>
      <part name="TopHello">
        <part name="hello" class="helloC"/>
      </part>
    </structure>
    <style>
      <property
        part-name="TopHello" name="rendering">Container</property>
      <property
        part-name="TopHello" name="content">Hello</property>
      <property
        part-class="helloC"  name="rendering">String</property>
      <property
        part-name="hello"    name="content">Hello World!</property>
    </style>
  </interface>
  <peers> ... </peers>
</uiml>
```

Kuva 1. HelloWorl käyttöliittymän kuvaus UIML-kielellä [UIML,2005].

Ylläesitetystä kuvasta nähdään helposti kuinka UIML-dokumentti koostuu kahdesta osasta. Dokumentin alussa sijaitsevasta prologista [UIML, 2005], joka määrittelee XML- ja UIML-versiot. Tämän jälkeen alkaa dokumentin toinen osa, jossa suoritetaan varsinainen käyttöliittymän määrittäminen. Tässä dokumentissa en esittele varsinaisia elementtejä tämän tarkemmin (ks. [UIML, 2005]).

2.2. AUIML

Abstract User Interface Markup Language (AUIML) [Nunes, 2001] on IBM:n kehittämä XML-pohjainen käyttöliittymien määrittely kieli. Kielen tarkoituksena on pyrkiä määrittelemään käyttöliittymä, sen tarkoituksen perusteella, ei niinkään kiinnittämällä huomiota sen esitykseen pohjautuviin kysymyksiin. AUIML:ssä voidaan tarkempien esitystapaan liittyvien kysymysten ratkaisut jättää kuvausta tulkkaavalle kerrokselle ns. tulkille (renderer) [Nunes, 2001]. Tällaisia tulkkeja voidaan määritellä useille eri alustoille, jolloin AUIML:llä

voidaan luoda käyttöliittymän kuvaus, jota voidaan suorittaa eri tulkeilla usealla eri alustalla. Kuitenkin itse käyttöliittymän määritelmä pysyy aina samana.

AUIML koostuu kahdesta pääelementtiluokasta. Tietomallista (Data Model [Nunes, 2001]), jonka perustehtävä on määrittää käyttöliittymän ja käyttäjän välinen vuorovaikutus. Tietomalli ei ole AUIML:ssä ota kantaa modaliteetteihin, vaan kuvaa vuorovaikutusta abstraktilla tasolla. Tällöin samalla kielellä voidaan määritellä käyttöliittymiä erilaisia modaliteettejä omaaville alustoille.

Toinen AUIML:n pääelementeistä on esitysmalli (Presentation Model) [Nunes, 2001]. Esitysmalli vastaa käyttöliittymäkomponenttien (pienemmät osaset kuten napit, valikot, jne.) ns. ”look and feel” [Nunes, 2001] ominaisuuksista. Toisin sanoen esitysmallissa määrätään miltä käyttöliittymän koostavien komponenttien tulisi näyttää ja tuntua. Esitysmalli kuvaa käyttöliittymää varsin korkealla tasolla, yrittäen mallintaa kunkin komponentin tarkoituksen ja tehtävän käyttöliittymässä, ei niinkään yksityiskohtaista toteutusta.

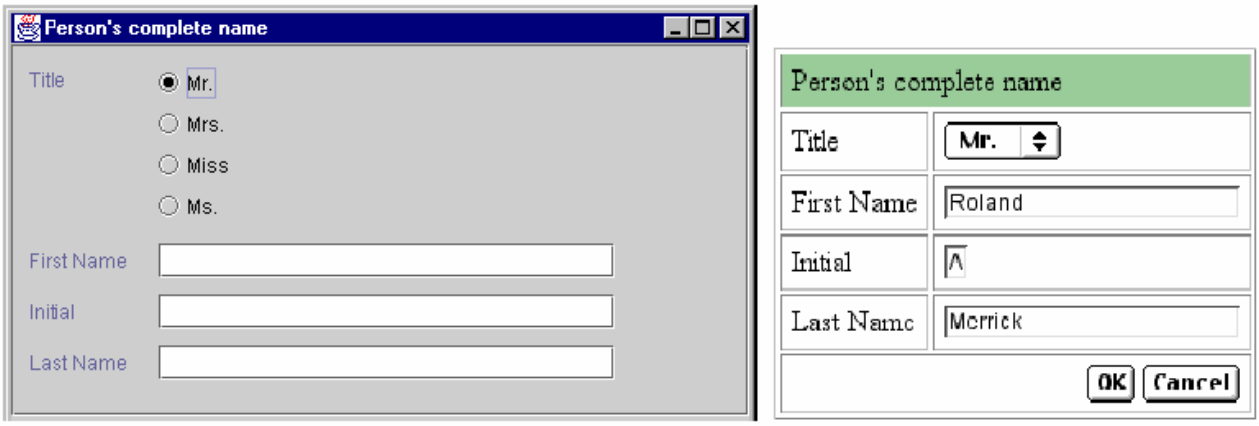
Näiden kahden pääelementin lisäksi AUIML:ssä voidaan määritellä toimintoja (Actions) [Souchon et al., 2003], joiden avulla voidaan luoda tapahtuman käsittelyä käyttöliittymän ja varsinaisen tiedon välille.

Seuraavassa kuvassa on esitelty yksinkertainen käyttöliittymä kuvaus, joka kysyy käyttäjältä tämän kokonimeä. Kuvauksesta nähdään kuinka esimerkiksi valintaryhmä sijoitetaan CHOISE-elementin sisään. Tässä kuvauksessa ei ole suoranaisesti otettu siis kantaa millä tavoin valintaryhmä esitettään. Vain valintaryhmään kuuluvat elementit on merkitty.

```
<GROUP NAME="PersonName">
      <CAPTION>
            <META-TEXT>Person's complete name</META-TEXT>
      </CAPTION>
      <CHOICE NAME="PersonTitles" SELECTION-POLICY="SINGLE">
      <CAPTION><META-TEXT>Title</META-TEXT></CAPTION>
      <HINT>
            <META-TEXT>This is a set of valid titles you may choose
from.</META-TEXT>
      </HINT>
      <STRING NAME="MR">
            <CAPTION><META-TEXT>Mr.</META-TEXT></CAPTION>
      </STRING>
      <STRING NAME="MRS">
            <CAPTION><META-TEXT>Mrs.</META-TEXT></CAPTION>
      </STRING>
      <STRING NAME="MISS">
            <CAPTION><META-TEXT>Miss</META-TEXT></CAPTION>
      </STRING>
      <STRING NAME="MS">
            <CAPTION><META-TEXT>Ms.</META-TEXT></CAPTION>
      </STRING>
      </CHOICE>
      <STRING NAME="FirstName">
            <CAPTION><META-TEXT>First Name</META-TEXT></CAPTION>
      </STRING>
      <STRING NAME="Initial">
            <CAPTION><META-TEXT>Initial</META-TEXT></CAPTION>
      </STRING>
      <STRING NAME="LastName">
            <CAPTION><META-TEXT>Last Name</META-TEXT></CAPTION>
      </STRING>
</GROUP>
```

Kuva 1. Yksinkertainen käyttöliittymä kuvaus AUIML kielellä, joka kysyy käyttäjän koko nimen [Nunes, 2001; Azevedo et al., 2000].

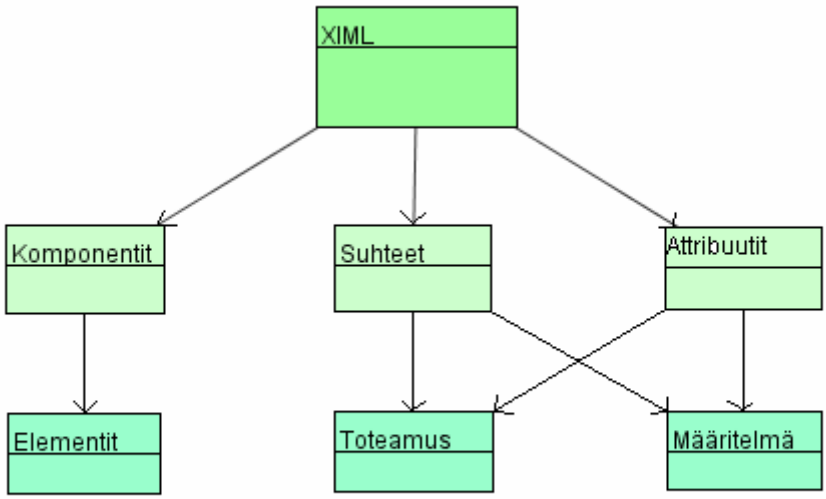
Seuraavassa kuvassa on esitelty ylläesitellyn käyttöliittymäkuvauksen kaksi tulkattua versiota. Ensimmäinen on tulkattu JavaSwing[Java, 2005] käyttöliittymä kirjastoa käyttäen. Jälkimmäinen taas on tulkattu DHTML-tulkilla, jolloin käyttöliittymä on selaimessa toimiva HTML-sivu. Kuvasta näemme kuinka itse kuvaus ei ole ottanut kantaa esimerkiksi valintaryhmän esittämiseen. Javalla valintaryhmä esitetään radiopainikkeina, kun taas DHTML:llä samainen asia on esitetty alasvetovalikolla.



Kuva 2. AUIML käyttöliittymä kuvaus tulkattu JavaSwing kirjaston avulla (vasemmalla) ja DHTML tulkin avulla [Nunes, 2001; Azevedo et al., 2000].

2.3. XIML

Extensive Interface Markup Language (XIML) tarjoaa kielen, jonka avulla voidaan määritellä käyttöliittymä ilman tarvetta ottaa kantaa varsinaiseen toteutukseen. XIML on siis toisin sanoen täysin alustariippumaton, jolloin sen avulla voidaan määritellä käyttöliittymiä useille eri laitteille. Seuraavassa kuvassa on esitelty kielen perusrakenne.



Kuva 3. Perusrakenne XIML-käyttöliittymä kuvaukselle. [XIML, 2005]

Yksinkertaisimmillaan XIML koostuu joukosta käyttöliittymä elementtejä, jotka voidaan luokitella yhteen tai useampaan käyttöliittymä komponentti luokkaan. Teoriassa kieli ei rajoita komponentti luokkien määrää. Toisin sanoen komponentti luokkia voidaan määritellä tarvittaessa aina uusia. Myöskään ei rajoiteta komponenttiluokan alle määriteltyjen elementtien määrää. Käytännössä kieli sisältää kuitenkin viisi peruskomponenttia, jotka on esitelty seuraavassa [XIML, 2005; Souchon et al., 2003].

- **Tehtäväkomponentti** (task component), kuvaa bisnes prosessit tai käyttäjälle näkyvät toiminnot, joita käyttöliittymä tukee, abstraktilla tasolla.
- **Toimialakomponentti** (domain component), kokoelma kaikista olioista ja olioluokista joita käyttöliittymä käyttää. Toimiala komponentin alle kuuluvat vain sellaiset oliot, joita käyttäjä voi hallinnoida (esim. päivämäärä).
- **Käyttäjäkomponentti** (user component), käytetään kuvaamaan sovellusta käyttävien käyttäjien ominaisuuksia.
- **Esitystapakomponentti** (presentation component), kuvaa varsinaiset elementit, joita käytetään käyttäjän ja sovelluksen väliseen vuorovaikutukseen. Esimerkiksi painikkeet, ikkunat, jne.
- **Dialogikomponentti** (dialog component), samankaltainen kuin tehtävä komponentti, mutta kuvaa käyttöliittymän tukemat toiminnot tarkemmalla tasolla. Voidaan esimerkiksi määritellä tapahtumien välistä tiedonsiirtoa.

Pelkästään yksittäisten elementtien määritteleminen ei riitä toimivan käyttöliittymän kuvaamiseen. Siksi XIML:ssä kuvataan näiden lisäksi suhteet kyseisten elementtien välille. Suhde pitää sisällään toteamuksen kahden tai useamman elementin välisestä linkistä (suhteesta). Suhteessa olevien elementtien ei tarvitse välttämättä olla saman komponentin alla, joten suhde voidaan määritellä mille tahansa elementeille. Suhde voi sisältää lisäksi määritelmän suhteelle.

Elementeille voidaan myös määrätä ominaisuuksia sekä niitä määrittelevää tietoa. Näitä kutsutaan XIML:ssä attribuuteiksi. Attribuutti voi olla, joko perus tietotyyppi tai se voi olla myös jokin toisen elementin instanssi. Toisin sanoen attribuutti on olio joka kuvaa jonkin elementin ominaisuutta.

2.4. XUL

Viimeiseksi esittelen kielen joka on tullut tutuksi Mozillan vallatessa alaa selain käyttäjien keskuudessa. Extensible User Interface Language (XUL) ei varsinaisesti ole, tai edes tavoittele olevansa täydellinen UIDL-kieli. XUL:in tarkoituksena on yksinkertaistaa sovelluskehitystä, mutta vain sovelluksille, joita käytetään Mozilla-tulkin kautta [Mozilla, 2005].

XUL erottelee selkeästi sovelluslogiikan ja varsinaisen käyttöliittymän. XUL sovelluksia voidaan lokalisoida helposti, koska arkkitehtuuri tukee erilaisten esitystapojen käyttöä varsinaisen käyttöliittymäkuvauksen esittämisessä. Uusi esitystapa voidaan määritellä esimerkiksi CSS-tyylien avulla. Lisäksi sovelluksille määritellään ns. kielitaulut, joita vaihtamalla voidaan sovelluksen kieli vaihtaa ilman suurempia muutoksia itse käyttöliittymään.

Käyttöliittymä määritellään joukolla ennalta määrättyjä käyttöliittymä elementtejä (painikkeet, ikkunat, jne.). Jokaisella elementeillä on myös ennalta määrätty joukko attribuutteja joiden arvoja muuttamalla saadaan elementit näyttämään tai toimimaan erilailla. Käyttäjän ja sovelluksen välistä vuorovaikutusta ohjataan scriptien avulla. Itse XUL ei siis ota kantaa tähän.

XUL on keskittynyt WIMP-käyttöliittymien kuvaamiseen, ja tarvitsee siis alustan jolla Mozilla toimii. Tämä rajoittaa kielen käytettävyyttä muihin UIDL-kieliin verrattuna, koska esimerkiksi pieniin mobiililaitteisiin ei menetelmällä ainakaan toistaiseksi voida sovelluksia kehittää.

3. Esiteltyjen kielten yhteneväisyyksiä ja eroavaisuuksia

Yllä esitellyt neljä kieltä ovat siis kaikki tarkoitettu helpottamaan käyttöliittymien toteuttamista. Kaikissa peruslähtökohtana on ainakin jollaintasolla saada aikaan yksinkertaisempia määrityksiä useille erialustoilla toimiville käyttöliittymälle. Käytännössä kuitenkin UIDL-mallin määritelmässä mainittuun ns. siirrettävyyteen ei esitellyistä kielistä yllä kuin AUIML ja XIML. UIML ja XUL eivät varsinaisesti määrittele käyttöliittymää sellaisella abstraktisuus tasolla, ettei määritelmässä jouduttaisi ottamaan kantaa varsinaiseen suoritusalustaan. XUL lisäksi erottaa varsinaisen käyttöliittymän täysin sovelluslogiikasta, vielä siinä määrin, ettei se tarjoa itse minkäänlaisia

keinoja määrittää esimerkiksi käyttäjän ja käyttöliittymän vuorovaikutusta. Tämä hoidetaan scriptein joita XUL-määritelmään voidaan liittää.

Kielistä vain XIML omaa mahdollisuuden esitellä uusia tag-elementtejä rakenteeseensa. Tästä syystä se on myös joustavin esitellyistä kielistä. Kolmessa muussa kielessä tagien määrä on rajoitettu. Tästä syystä myös kielen kuvaavuus kärsii. XIML on kielistä siis kaikkein kuvaavin. Kielen ollessa ns. avoin [Souchon et al., 2003], voidaan siihen esitellä uusi elementti, jos ja kun sellaista tarvitaan.

Kielten käyttöön ottaminen ja saatavuus on varsin helppoa. Ainoa kielistä, josta on ylipäätänsä hankala saada edes tietoa, on AUIML. AUIML on kehitetty IBM:n sisäisiin hankkeisiin, joten suuriosa sen dokumentaatiosta on salaista. Kieli on kuitenkin varsin kuvaava ja vaikuttaa käytettävältä, joten voidaan olettaa, että jonkinlaisia ainakin kaupallisia versioita on myös tämän kielen tiimoilta odotettavissa. Muiden kielten osalta sovellutuksia löytyy varsin runsaasti, niitä tässä tarkemmin esittelemättä (ks. [UIML,2005 ;XIML,2005; XUL,2005]).

4. Yhteenveto

Tässä paperissa esitellyt neljä UIDL-kieltä eivät toki ole ainoat kehitteillä olevat käyttöliittymien korkeantason määrittämiseen tähtäävät kielet. Tätä paperia varten tehdyn tiedonhaun perusteella voin olettaa että tällä hetkellä varteenotettavia UIDL-ideologiaan tähtääviä jollaintasolla valmiita kieliä on toista kymmentä. Lisää kehitellään kaiken aikaa. Tällä hetkellä sopivan kielen löytäminen riippuukin varsin suuresti käyttötarkoituksesta. Monissa kielissä on hyvät ja huonot puolensa, mutta täysin UIDL-ideologiaa noudattavaa kieltä ei varmastikaan vielä löydy.

Jokainen kehitteillä oleva UIDL-kieli pyrkii määrittelemään yleisen standardin, jonka mukaan voitaisiin kehitellä laajempi yleiseen käyttöön tarkoitettu kieli. Nykyisellään ollaan kuitenkin vielä jokseenkin kaukana yleisen standardin määrittelemisestä. Tällä hetkellä markkinoilla on liikkeellä useita samaan tähtääviä, mutta jokseenkin erityyppisin metodein toteutettuja kieliä. Tämä johtaa myös siihen, että loppukäyttäjä joutuu tekemään valinnan mitä hän kieleltä tarvitsee, koska kaikilla kielillä ei onnistu kaikki, mitä UIDL-idea pitää sisällään.

Jos esimerkiksi ollaan kehittämässä käyttöliittymää monelle erityyppiselle laitteelle, löytyy varmasti kieli tähän tarkoitukseen. Samalla kielellä ei

välttämättä kuitenkaan pystytä kehittämään käyttöliittymiä yksinomaan selainpohjaisille sovelluksille. Tai vaikka pystyttäisiinkin, saattaa se olla vaikeampaa, tai kalliimpaa, kuin jokin yksinomaan tälle alustalle suunnattu sovelluskehitys vaihtoehto. Vaikka löydettäisiinkin varsin kuvaava kieli, ei välttämättä löydykään tulkkeja tai kääntäjiä, joiden avulla korkeantason kuvauksesta saataisiin alustalla toimiva käyttöliittymä.

UIDL-kieliä kehitellään siis kaikenaikaa, ja kilpajuoksu kattavan standardin saamiseksi on käynnissä. Tällä hetkellä valinnan varaa kuitenkin löytyy, sekä kaupallisella sektorilla, että avoimellakin puolella. Tulevaisuudessa UIDL-kielten kehittelyä eteenpäin vievä voima on todennäköisesti mobiili-puolen sovelluksissa. Johan Plomp tutkimusryhmineen esittelee esimerkiksi paperissaan *Comparing Transcoding Tools for Use with a Generic User Interface Format* [Plomp, 2002], kuinka voidaan helposti siirtää yhden tyyppinen käyttöliittymä kuvaus erityyppiselle alustalle, käytännössä mahdollisimman vähin muutoksin. Mobiili maailmassa tällaisten tekniikoiden avulla saavutetaan helppokäyttöisyyttä, sekä säästöjä, koska pystytään samalla tekniikalla kehittämään käyttöliittymä monelle erityyppiselle alustalle.

Toinen hyväpuoli UIDL-kielten kehittelemiseen on erilaisten käyttöliittymä editoreiden kehittäminen. Kun käyttöliittymä pystytään määrittelemään yleisellä korkeantason kielellä, voidaan tälle kielelle kehitellä helppokäyttöisiä graafisia editoreita, jolla voidaan piirtää käyttöliittymä. Tällaisella graafisella editorilla käyttöliittymän toteuttajan ei tarvitse varsinaisesti tuntea minkäänlaista ohjelmointikieltä, vaan käyttöliittymän voi toteuttaa sen varsinainen suunnittelija, joka yleisesti omaa käytettävyys taustaa. Tällöin ei välttämättä myöskään tarvita kovin tarkkoja kuvauksia käyttöliittymästä koska sen suunnittelija tekee varsinaisen toteutuksen. Kun yksi vaihe jää tuotanto vaiheesta pois säästetään aikaa sekä rahaa. Lisäksi vaikka käytetäänkin graafista sovellusta, voidaan kehitettyä käyttöliittymää käyttää uudelleen, koska määrittely tallentuu UIDL-kielellä tehtyyn varsinaiseen kuvaukseen.

UIDL-kielillä katson hekilökohtaisesti olevan tulevaisuudessa paljonkin käyttöä, varsinkin nykyisellään paljon huomiota saaneella mobiili puolella, mutta myös web, sekä työasema sovelluspuolella.

Viiteluettelo

[Azevedo et al., 2000] Azevedo, P., Merrick, R. and Roberts, D., *OVID to AUIML – User-Oriented Interface Modelling,* 2000. In Proceedings of Towards a UML Profile for Interactive System development (TUPIS'00) Workshop, http://math.uma.pt/tupis00, York –UK.

[Java, 2005] Java Technology, Sun Microsystems, 2005. http://java.sun.com

[Mozilla, 2005] Mozilla.org, 2005, http://www.mozilla.org

[Nunes, 2001] Duarte Nuno Jardim Nunes, *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach,* 2001. Sivut 203-212. http://xml.coverpages.org/NunoWisdomThesis.pdf

[Plomp, 2002] Plomp Johan, Schaefer Robbie, Mueller Wolfgang, *Compairing Transcoding Tools for Use with a Generic User Interface Format*, 2002. http://www.idealliance.org/papers/extreme03/xslfo-df/2002/Schaefer01/EML2002Schaefer01.pdf

[Souchon et al., 2003] Natalie Souchon and Jean Vanderdonckt, *A Review of XML.Compliant User Interface Description Languages,* 2003. http://www.isys.ucl.ac.be/bchi/publications/2003/Souchon-DSVIS2003.pdf

[UIML, 2005] *User Interface Markup Language(UIML), 2005*. Harmonia, Inc, 2005. http://www.uiml.org

[USIXML, 2004] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins and Daniela Trevisan, *USIXML: A User Interface Description Language for Context-Sensitive User Interfaces,* 2004. http://www.isys.ucl.ac.be/bchi/publications/2004/Limbourg-UIXML2004.pdf

[XIML, 2005] Angel puerta and Jacob Eisenstein, *XIML:A Universal Language for User Interface,* XIML Forum 2004. http://www.ximl.org

[XML, 2005] *Extensible Markup Language,* World Wide Web Consortium W3C 2005. http://www.w3.org/XML/

[XUL, 2005] The XML User Interface Language, Mozilla Foundation, 2005. http://www.xulplanet.com/

Jaettujen editorien arkkitehtuurit

Jussi Rantala

Tiivistelmä.
Jaetut editorit ovat keino tukea yhteistoiminnallisuutta ohjelmistosovelluksissa. Käyttäjät pääsevät samanaikaisesti käsiksi jaettuun tietoon, jota voidaan muokata reaaliaikaisesti yhteistyönä. Jaettujen editorien ominaisuudet, kuten samanaikaisuus, tiedon välittäminen ja sen näkyvyyden kontrollointi, asettavat omat vaatimuksensa ohjelman arkkitehtuurin suunnittelulle ja toteutukselle. Eri arkkitehtuurivaihtoehtoja on useita, joista kaikilla on omat hyvät ja huonot puolensa. Tässä tutkielmassa käydään läpi jaettujen editorien tyypillisiä piirteitä, sekä niiden vaikutuksia ohjelman toteuttamiseen.

Avainsanat ja -sanonnat: Jaettu editori, yhteistoiminnallisuus, ohjelmistoarkkitehtuuri

1. Johdanto

Useimmat käyttämistämme ohjelmistoista ovat suunniteltu yhtä käyttäjää varten. Kirjoitamme dokumenttia tai työstämme graafista esitystä itsenäisesti, jonka jälkeen työn tulokset tallennetaan. Tämän jälkeen on mahdollista lähettää tai siirtää työ muiden käyttäjien saataville. Monissa tilanteissa tämä perinteinen työskentelymuoto on tehokas, ellei jopa paras vaihtoehto. On kuitenkin myös lukuisia tilanteita, joissa työskentelystä saadaan tehokkaampaa käyttäjien läheisemmän yhteistyön avulla. Yhteistoiminnallisuutta tukevia ohjelmia käytettäessä työstettävää informaatiota ei tarvitse erikseen lähettää muille oman osuuden valmistuttua, vaan tieto on kaikkien käyttäjien saatavissa ja muokattavissa reaaliaikaisesti.

Jaetut editorit ovat tarkoitettu tilanteisiin, joissa käyttäjät työskentelevät samanaikaisesti yhteisen kohteen parissa esimerkiksi tekstiä tai grafiikkaa käsitellen. Nämä editorit voidaan lukea CSCW-tyyppisiin (Computer Supported Cooperative Work) ohjelmiin, joiden tarkoituksena on tukea ihmisten välistä yhteistoiminnallisuutta tietokoneympäristössä. Jaettujen editorien idea ei ole aivan uusi, sillä niiden parissa on tehty tutkimusta jo noin parikymmentä vuotta.

Vaikka tekniikka on vuosien saatossa kehittynyt, samat lainalaisuudet ovat yhä voimassa. Yhdellä näytöllä työskentelystä on siirrytty verkottumisen myötä lähinnä tietoliikenneyhteyden yli tapahtuvan usean päätteen käyttämiseen.

Hyvän jaetun editorin tulee olla helppokäyttöinen, jaettuja objekteja pitää päästä muokkaamaan kun siihen on tarve ja muiden käyttäjien toiminnan näkyvyyttä pitää pystyä tarvittaessa säätelemään [Newman-Wolfe *et al.,* 1992]. Samanaikainen työskentely ei saa vaikuttaa käyttäjän työskentelyyn negatiivisesti; yhteistoiminnallisuuden pitää tuoda lisäarvoa käyttäjälle. Tiedon jakamisen tulee olla käyttäjälle mahdollisimman läpinäkyvää, mutta toisaalta tarvittaessa jakamisesta pitäisi saada lisätietoa.

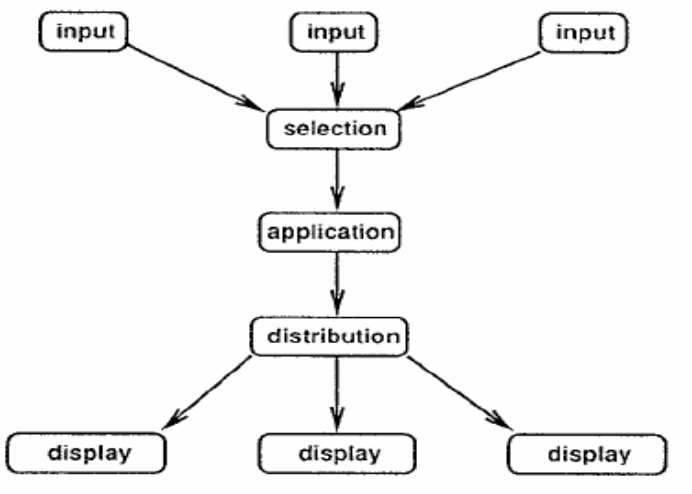
Yhteistoimintamahdollisuus tuo mukanaan omat vaatimuksensa editorien arkkitehtuuritoteutuksille. Onnistunut toteutus vaatii monien asioiden huomioimista. Perinteisiin ohjelmiin verrattuna esimerkiksi tiedon välittäminen käyttäjien välillä, samanaikaisuuden hallinta, syötteiden käsittely sekä toisten käyttäjien toimien näkyminen ovat kriittisiä seikkoja. Käytettävyydeltään hyvän ja toiminnaltaan tehokkaan jaetun editorin luomisessa on punnittava monia asioita ja välillä tehtävä kompromisseja eri vaihtoehtojen välillä.

Henkilökohtaisilla päätteillä ajettavien editorisovellusten lisäksi on olemassa myös suurille tauluille tarkoitettuja laajempia sovelluksia, kuten Xerox PARC:issa kehitetty Liveboard [Elrod *et al.,* 1992]. Nämä järjestelmät mahdollistavat esimerkiksi syötteiden antamisen koskettamalla kynällä taulua. Suora vuorovaikutustekniikka sekä taulun fyysinen koko ovat ominaisuuksista, jotka asettavat suunnittelulle ja toteutukselle uusia haasteita. Tämän tutkielman puitteissa käsitelläänkin ainoastaan yksittäisille koneille suunnattuja editorisovelluksia.

Tutkielman tarkoituksena on antaa yleiskuva jaettujen editorien ominaisuuksista sekä niiden arkkitehtuuritoteutuksille asettamista vaatimuksista. Aluksi esitellään perustavat arkkitehtuurivalinnat, jotka toimivat pohjana editorin muussa toteutuksessa. Seuraavaksi tarkastellaan, kuinka muiden käyttäjien toimet näkyvät omalla näytöllä sekä sitä, kuinka tätä yhteistoiminnallisuuden astetta voidaan muuttaa. Lisäksi tarkastellaan käyttäjien syötteiden käsittelyä ja samanaikaisuuden hallintaa. Lopuksi esitellään kaksi yleisten toteutusmallien mukaan rakennettua jaettua editoria.

2. Runkotason arkkitehtuurivalinnat

Jaettujen editorisovellusten toteuttamisessa tulee huomioida niiden asettamat vaatimukset käyttäjien toimien välittämisestä muille osallistujille. Ei ole yhdentekevää, minkä periaatteen mukaisesti järjestelmä on rakennettu. Toinen ratkaisu saattaa tarjota nopeamman reagoinnin editorilla tehtyihin muutoksiin, kun taas toinen voi soveltua paremmin tiedonsiirtokapasiteetin säästämiseen. Seuraavissa alikohdissa esittelen kolme jaettujen editorisovellusten yhteydessä käytettyä arkkitehtuurimallia [Newman-Wolfe *et al.*, 1992].

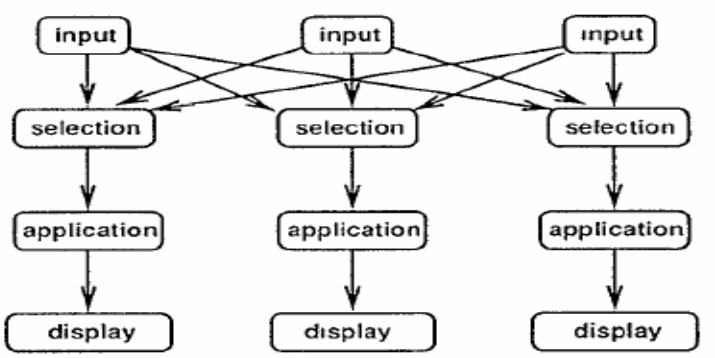


Kuva 1. Keskitetty arkkitehtuuri. [Newman-Wolfe *et al.*, 1992]

Keskitetyn arkkitehtuurin etuna on kaikkien käyttäjien näkymien yhtenäinen tila. Koska syötteet käsitellään samassa paikassa ja samat näytönpäivityskäskyt lähetetään kaikille editorin käyttäjille, ei ohjelman tilassa ole eroavaisuuksia eri käyttäjien välillä. Keskitetyn arkkitehtuurin haittapuolena voidaan pitää keskusprosessoinnin asettamia aika- ja verkkovaatimuksia. Käyttäjän tekemän muutoksen lähetyshetken ja sen näytölle ilmestymisen välissä kuluu aikaa. Keskusprosessista huolehtiva kone voi olla fyysisesti kaukana, jolloin siirrettävän tiedon määrä ja verkkoyhteyden kapasiteetti määräävät käyttöön vaikuttavan viiveajan [Ahuja *et al.,* 1990]. Reaaliaikaisuuden puute ja näyttöpäivitysten venyminen ovat mahdollisia ilmiöitä keskitettyä arkkitehtuuria käytettäessä [Crowley *et al.,* 1990].

2.1. Monistettu arkkitehtuuri

Monistettua arkkitehtuuria hyödyntävissä editoriohjelmissa jokaisen käyttäjän omalla koneella käsitellään syötteiden ja tulosteiden lisäksi myös muutokset ohjelman sisäiseen tilaan. Käyttäjän luomat syötteet ohjataan raakamuodossa käsiteltäviksi jokaiselle editoria käyttävälle. Nyt jokaisella käyttäjällä on siis oma paikallinen versio ohjelmasta syöte- ja näkymäprosessien lisäksi (kuva 2). Ohjelmaprosessit käsittelevät saamansa tiedot itsenäisesti, mikä mahdollistaa personoidut muutokset editoriohjelmaan.



Kuva 2. Monistettu arkkitehtuuri. [Newman-Wolfe *et al.*, 1992]

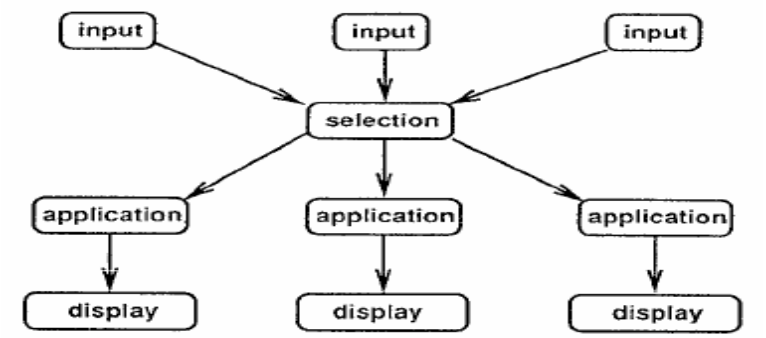
Monistetun arkkitehtuurin etuna on paikallisen käsittelyn tuoma hyöty tiedonsiirrossa. Nyt käyttäjien kesken välitetään lähinnä vain syötetietoa. Keskitettyyn arkkitehtuuriin verrattuna pelkän syötetiedon siirtäminen sekä syöte- että näyttötietoihin verrattuna säästää tiedonsiirtokapasiteettia. Usein syötelaitteilta tulevan tiedon määrä on jo sinällään pienempi kuin muutosten jälkeisen näyttötilainformaation kuvaamisen vaatima tietomäärä. Lisäksi monistetun arkkitehtuurin yhteydessä syötteen lähetyshetken ja näytön päivittymisen välillä kuluva aika on huomattavasti lyhyempi, sillä paikallinen kopio prosessoi saadut syötteet välittömästi. Monistetun arkkitehtuurin avulla myös työskentely erilaisten laitteistostandardien kesken on helpommin toteutettavissa [Ahuja *et al.*, 1988]. Editorisovelluksen näyttämistä monitorilla voidaan mukauttaa esimerkiksi eri resoluutioiden vaatimusten perusteella, kun informaatio käsitellään paikallisten kopioiden avulla.

Monistetun arkkitehtuurin suurin haittapuoli on vaatimus toimien synkronoinnista. Lukuisat käyttäjät suorittavat suuren määrän komentoja omaan tahtiinsa, jolloin siirrettävien syötetietojen määrä kasvaa suureksi. Periaatteessa jos kaikki käyttäjät vastaanottavat samat tiedot samassa järjestyksessä, näkymät

pysyvät konsistentteina kaikissa paikallisissa kopioissa, mutta käytännössä näin ei kuitenkaan aina ole. Synkronointiongelmia ilmenee esimerkiksi pitkien fyysisten etäisyyksien aiheuttamien tiedonsiirtovaatimusten takia.

2.2. Ensemblen hybridiarkkitehtuuri

Selkeästi keskitettyjen ja monistettujen arkkitehtuuriratkaisuiden lisäksi voidaan käyttää näitä kahta periaatetta yhdistäviä järjestelmätoteutuksia. Ensemblessä [Newman-Wolfe *et al.*, 1992] pyritään hyödyntämään edellä esiteltyjen ratkaisuiden parhaita puolia. Ensemble käyttää keskusprosessia, joka ylläpitää editorin sisäistä tilaa ja ohjaa käyttäjiltä saatavia syötteitä eteenpäin (kuva 3). Lisäksi jokaisella käyttäjällä on omat syöte-, näyttö- ja ohjelmaprosessit monistetun arkkitehtuurin kaltaisesti. Eroavaisuutena aikaisempaan on se, että kaikkia käyttäjien syötteitä ei ohjata keskusprosessin kautta. Ennen kuin käyttäjän syöte lähetetään prosessoitavaksi eteenpäin, tarkastetaan sen luonteen perusteella, voiko se aiheuttaa ristiriitoja ohjelman sisäisessä esityksessä. Jos syöte on yksiselitteinen, eikä se voi aiheuttaa konflikteja ohjelman tilassa, ohjataan se suoraan käyttäjän oman kopion käsiteltäväksi. Mikäli syöte on altis aiheuttamaan ristiriitoja muiden käyttäjien toimien kanssa, täytyy se ensin hyväksyä keskusprosessin toimesta ennen sen varsinaista käsittelyä.



Kuva 3. Ensemblen arkkitehtuuri. [Newman-Wolfe *et al.*, 1992]

Tämän arkkitehtuurivalinnan avulla vasteaika syötteen antamisen ja ruudunpäivityksen välillä pysyy hyvänä, sillä suurin osa tiedosta on ristiriidatonta ja se käsitellään paikallisesti. Konfliktitilanteiden ja synkronointiongelmien määrää saadaan vähennettyä perinteiseen monistettuun arkkitehtuuriin verrattuna tarkistamalla ongelmaherkät syötteet keskusprosessissa. Paikallisesta prosessoinnista johtuen myös verkon kuormitus on vähäisempää. Lisäksi paikallista sovellusta voidaan muuttaa esimerkiksi

käyttäjän tarpeiden mukaisesti, sillä se toimii oman kopionsa perusteella. Näin voidaan tapauskohtaisesti tehdä päätös vaikkapa siitä, mitä syötteitä otetaan vastaan muilta käyttäjiltä.

3. Näkymien mukauttaminen

Jaettujen editorien vahvuutena on usean käyttäjän työpanoksen yhdistäminen. Lukuisten erilaisten editoriohjelmistojen ja työskentelytapojen seurauksena samanlaiset näkymävalinnat eivät ole parhaita kaikissa tilanteissa. Välillä käyttäjä tarvitsee enemmän yksityisyyttä, kun toisessa tilanteessa taas läheinen yhteistyö muiden kanssa on tehokkain vaihtoehto. Seuraavissa aliluvuissa esitellään erilaisia tapoja muokata muiden käyttäjien toimien näkymistä kontrolloimalla siirrettävää tietoa [Berlage ja Genau, 1993].

3.1. Tiukka yhteen liittäminen

Tiukasti yhteen liitetyissä (tightly coupled) editoreissa kaikki omalla näytöllä näkyvä informaatio lähetetään myös muille käyttäjille. Tämä tarkoittaa esimerkiksi hiiren kursorin liikkeitä sekä valikkojen käyttöä. Editorin toiminta vastaa hyvin pitkälle jaetun ikkunan periaatetta, missä useampi käyttäjä toimii samalla työskentelyalalla. Tiukasti yhteen liitettyjä näkymiä käytettäessä toiminta on WYSIWIS-periaatteen (What You See Is What I See) mukaista.

Tiukasti yhteen liitettyjen näkymien hyvänä puolena on toiminnan läpinäkyvyys, jolloin käyttäjät pystyvät seuraamaan muiden työskentelyn edistymistä. Toisaalta näytöllä esitettävän informaation määrä voi kasvaa liian suureksi, mikä vaikuttaa negatiivisesti omaan työskentelyyn. Esimerkiksi toisen käyttäjän kursorin näkyminen saattaa häiritä omaa hiirenkäyttöä. Kaikkien tapahtumien lähettäminen ja erityisesti niiden looginen näyttäminen voi olla haastavaa. Useampi henkilö voi pyrkiä käyttämään valikoita samanaikaisesti, jolloin ohjelmiston tulee olla teknisesti valmis näyttämään useita samanaikaisesti auki olevia valikoita. Edellä esitellyistä arkkitehtuuriratkaisuista keskitetty arkkitehtuuri on luonteeltaan looginen valinta tiukasti yhteen liitettyjen editorien perustaksi. Syötteet ja näkymille vietävät tulostetiedot kulkevat yhden keskusprosessin kautta, jolloin ne ovat helppo välittää samassa muodossa kaikille.

3.2. Vapaa yhteen liittäminen

Vapaasti yhteen liitettyjen (loosely coupled) editorien kohdalla periaatteena on se, että ainoastaan ohjelman tilan muutokset lähetetään toisille käyttäjille. Nyt esimerkiksi osoitinlaitteen liikkeitä ei välitetä muiden nähtäväksi. Vasta jos graafisen esityksen jotain komponenttia on muokattu, täytyy tämä muutostieto lähettää muille käyttäjille. Ohjelman komennot voidaan jakaa kahteen ryhmään sen perusteella, tarvitseeko niiden suorittamisesta välittää informaatiota eteenpäin muille käyttäjille. Tämä lajittelu dokumentin tilaa muuttavien ja ainoastaan käyttäjän omaan näkymään vaikuttavien komentojen välillä on olennainen vapaammin yhteen liitettyjen näkymien kohdalla. WYSIWIMS-periaate (What You See Is What I May See) on lähellä vapaasti yhteen liitettyjen näkymien toimintaa.

Käyttäjien on mahdollista mukauttaa näkymäänsä omien mieltymystensä mukaan. Heillä ei esimerkiksi tarvitse olla objektin samoja osia näkyvillä, tai he voivat katsella eri versiota siitä. Tällainen mukautettavuus saavutetaan suodattamalla syötteistä ne, jotka käyttäjä haluaa nähdä. Käytännössä yksi tapa saavuttaa tämä on käyttää edellä esitellyn monistetun arkkitehtuurin mukaisia paikallisia kopioita, jotka vastaavat suodatuksesta. Käyttäjä voi omasta pääteohjelmastaan valikoida objektit tai ikkunat, jotka haluaa nähdä. Nyt muiden käyttäjien lähettäessä komentosyötteitään, pystytään valikoimaan ainoastaan ne, joista ollaan kiinnostuneita.

3.3. Erilliset näkymät

Erillisiä näkymiä (separate mode) käytettäessä jokainen käyttäjä työskentelee itsenäisesti, eikä muiden toiminta näy millään tavalla omalla näytöllä. Tämä vastaa perinteisen yhdelle käyttäjälle suunnitellun editoriohjelmiston käyttöä. Käyttäjät voivat jopa siirtää dokumentin erilliselle laitteelle, jolloin muutosten välittäminen muille editoria käyttäville on mahdotonta. Yhteistoiminta tulee mukaan vasta silloin, kun käyttäjien itsenäisesti työstämät tiedot liitetään yhdeksi kokonaisuudeksi. Erillisesti suoritettujen komentojen perusteella kasataan yksilöllinen komentohistoria, josta voidaan jäljittää tehdyt muutokset yksitellen. Eri komentohistoriat voidaan yhdistää jälkikäteen. Käytännössä tämä ei aina suju ongelmitta. Toisistaan poikkeavien komentojen perusteella yhteistä ohjelman tilaa luotaessa tulee eteen ongelmatilanteita, jolloin käyttäjät ovat vastuussa kompromissien tekemisestä. Editorisovellus voi tarjota vaihtoehtoisia tapoja muutosten integroimiseksi, jolloin käyttäjät ovat vastuussa

päätöksenteosta. Tätä eri komentohistorioiden yhteen liittämistä tarkastellaan tarkemmin seuraavassa kappaleessa sekä sivutaan kappaleessa 5.3.

3.4. Näkymien vaihtaminen

Jotta erilaisiin yhteistoiminnallisiin tilanteisiin kyettäisiin sopeutumaan joustavasti, voi editorisovellus tarjota mahdollisuuden vaihtaa työskentelynäkymää kesken sovelluksen suorituksen. Yleisesti tiukemmin yhteen liitetyistä näkymistä vapaampiin siirtyminen ei tuota ongelmia. Yhteisestä tilasta siirtymisen yhteydessä ei tarvitse koota aikaisempia komentoja yhteen, sillä ne ovat olleet samat kaikille käyttäjille.

Vapaasta työskentelymuodosta tiukemmin yhteen liitettyihin näkymiin siirtyminen on haastavampaa. Vaikka komentohistorioiden avulla palattaisiin tilaan, jossa siirtyminen vapaampaan työskentelyyn tapahtui, ei suoraa siirtymistä takaisin voida suorittaa välissä mahdollisesti tehtyjen yksilöllisten editointien vuoksi. Siirtyminen on suoritettava askel kerrallaan. Käyttäjien erillisen työskentelyn aikana suorittamat komennot käydään läpi yksitellen, ja niiden perusteella muokataan yhteinen ohjelman tila. Vapaan työskentelyn aikana tehdyt komennot mallinnetaan erillisiksi haaroiksi, jotka pyritään yhdistämään. Yksi haara tulee valita nykyiseksi tilaksi, minkä jälkeen muista haaroista siirretään komentoja tämän haaran jatkoksi. Vapaammasta työskentelytilasta tiukempaan siirtymistä määrittää se, kuinka itsenäisesti käyttäjät työskentelivät aikaisemmin. Jos käyttäjien yhteinen työskentely oli verrattain tiivistä, esimerkiksi ainoastaan käyttäjäkohtaisten tietojen muutokset jätettiin käsittelemättä muiden käyttäjien koneilla, ohjelman näkymä on kaikilla käyttäjillä sama. Sen sijaan jos käyttäjät ovat työskennelleet täysin itsenäisesti, on jokainen luonut oman yksilöllisen historiansa ja muutoksensa. Nyt erillisiä haaroja käsiteltäessä on merkitystä järjestyksellä, missä komentohistoriat yhdistetään. Viimeisestä erillisestä haarasta yhteiseen haaraan tuodut komennot jäävät aina päällimmäisinä voimaan. Työskentelyn tulosten yhdistäminen vaatiikin osallistujien panosta yhteistä versiota luotaessa, sillä kaikkia päätöksiä historioiden yhdistämisestä ei ole hyvä tehdä automaattisesti.

4. Syötteiden ja tulosteiden käsittely

Yhteistoiminnallisuutta tukevien ohjelmien yhteydessä erilaisten syöttölaitteiden kautta saatavien käskyjen määrä voi kohota suureksi. Jos käyttäjien vuorovaikutuksen hallinta ei ole riittävän hyvin suunniteltua, saattaa

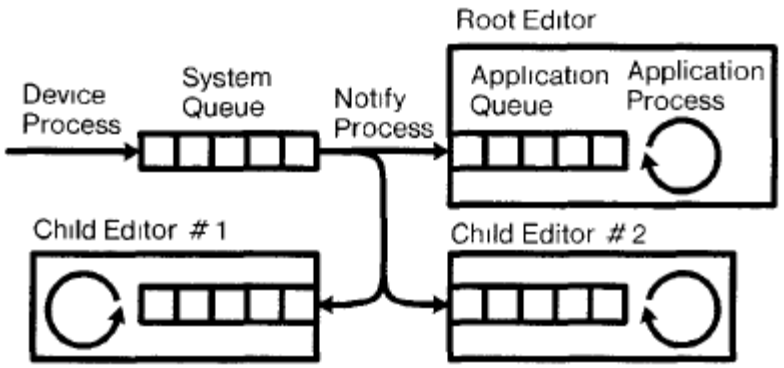
samanaikaisesta työskentelystä koitua enemmän haittaa kuin hyötyä. Tiedon joustavan käsittelyn takaamiseksi ja käytettävyydeltään hyvien editoriohjelmien luomiseksi täytyy suunnitteluvaiheessa huomioida tiettyjä informaation käsittelyyn liittyviä asioita. Seuraavissa alikohdissa esitellään tapoja syötteiden ja tulosteiden sujuvaan hallintaan.

4.1. Syötteiden käsittely

Jaettujen editoriohjelmien yhteydessä syötteiden hallinnan tulisi olla viiveetöntä ja luotettavaa. Jos käyttäjän haluama kohde ei reagoi tehtyyn muutokseen tarpeeksi nopeasti, voi syötteiden käsittely olla puutteellista. Pahimmassa tapauksessa haluttu editointi voi jäädä kokonaan käsittelemättä, jolloin muutosta ei myöskään nähdä näytöllä. Edellä mainittujen tapausten välttämiseksi syötteiden hallinnan on oltava joustavaa ja virheetöntä.

Esimerkkinä syötteiden hallinnasta käytetään MMM-järjestelmää [Bier ja Freeman, 1991]. MMM (Multi-Device Multi-User Multi-Editor) on Xerox PARC:issa kehitetty jaettu editorisovellus tekstin ja yksinkertaisten kuvien käsittelyyn. MMM on tarkoitettu yhdellä näytöllä tapahtuvaan editointiin, jolloin kaikki syötteet käsitellään siis samalla koneella. Samalla yhden pääohjelman periaatteella toimivat keskitetyn arkkitehtuurin mukaiset järjestelmät, jolloin jokaisella käyttäjällä voi olla oma päätteensä.

Ohjelma kirjaa syötteen ylös, kun käyttäjä painaa esimerkiksi näppäimistöltä jotain merkkiä tai liikuttaa hiirtä. Syöte tallennetaan tapahtumana, josta koordinaattien lisäksi otetaan ylös tapahtuma-aika. Tapahtuma asetetaan tämän jälkeen jonoon, jota hallinnoi keskitetyn arkkitehtuuriperiaatteen mukaisesti pääprosessi. Yksittäisten käyttäjien prosessit eivät pääse muokkaamaan tätä tapahtumajonoa. MMM tukee sisäkkäisiä editoreita, jotka voidaan ymmärtää ohjelman pääikkunan sisällä olevina pienempinä ikkunoina. Pääohjelma ylläpitää tietoa näistä sisäkkäisistä editoreista puumallin avulla. Puun juurena toimii pääeditori, ja sen sisällä olevat editorit taas edustavat juuresta lähteviä haaroja (kuva 4). Kun pääprosessin jonosta (System Queue) otetaan syöte käsittelyyn, aloitetaan puun juurena olevasta editorista (Root Editor) ja tutkitaan, kuuluuko syöte tälle editorille. Lapsieditoreja (Child Editor) käydään läpi kunnes oikea editori löytyy.



Kuva 4. MMM-editorin syötteidenkäsittelymekanismi. [Bier ja Freeman, 1991]

Pääprosessi seuraa jokaisen käyttäjän työskentelyä kirjaamalla ylös käyttäjän tekemiä muutoksia. Esimerkiksi piirto-ohjelman yhteydessä voidaan ylläpitää tietoa käyttäjien valitsemista työkaluista ja väreistä. Kun tapahtumajonon tapahtumalle löydetään puurakenteesta oikea editori, täydennetään tapahtuma näillä käyttäjän tiedoilla.

Kun syötetapahtuma on käsitelty pääprosessissa, voidaan se välittää muille käyttäjille näytönpäivitystä varten. MMM-järjestelmän kohdalla jokaisella käyttäjällä on käytössä henkilökohtainen jonorakenne, johon vastaanotetut tapahtumat sijoitetaan odottamaan päivitysrutiinien käsittelyä. Koska kaikki tapahtumat sijoitetaan samaan jonoon, tarvitaan jokin periaate syötteiden sujuvan käsittelyn takaamiseksi. Tapahtumat voidaan suorittaa niiden tapahtumahetkellä kirjatun kellonajan perusteella. Vanhimmat tapahtumat näytetään ensimmäiseksi. Tämän lisäksi tapahtumia voidaan kategorisoida tärkeyden mukaan. Pakolliset tapahtumat näytetään ennen vähemmän kiireellisiä tapahtumia. Tämä on yksi tapa pyrkiä syötteiden joustavaan käsittelyyn.

4.2. Näytönpäivitys

Näytönpäivityksen toteutus usean käyttäjän järjestelmissä ei ole triviaali tehtävä. Ruudulla näkyvän tiedon ja käyttäjien toimien tulee olla konsistentteja keskenään. Nykyisten tehokkaiden järjestelmien aikana riittävän nopea päivittäminen reaaliaikaisuuden takaamiseksi ei muodostu laitteiston puolesta ongelmaksi. Käyttäjien päällekkäiset toimet voivat kuitenkin aiheuttaa hankaluuksia päivitysoperaatioiden kohdalla.

Samaan tietoon kohdistuvat luku- ja kirjoitusoperaatiot saattavat aiheuttaa ongelmia jaettujen editorien yhteydessä. Onkin huolehdittava siitä, etteivät eri prosessit pääse samaan aikaan lukemaan ja kirjoittamaan samaa tietoa [Bier ja Freeman, 1991]. Tämä on tilanne varsinkin silloin, kun editorisovellus noudattaa keskitettyä arkkitehtuuria. Kaikki sovelluksen tilaa kuvaava tieto on yhdessä paikassa, josta sitä käytetään monen päätteen toimesta. Jotta ristiriitatilanteilta saman tiedon luku- ja kirjoitusoperaatioiden kohdalla vältyttäisiin, voidaan pääsy päivitettävään tietoon estää hetkeksi. Kun tiedon kirjoitus tai luku on suoritettu loppuun, pääsevät jälleen muiden käyttäjien prosessit käsittelemään samaa tietoa. Näin voidaan estää konfliktitilanteita fyysisen tiedonkäsittelyn tasolla.

5. Samanaikaisuuden vaatimukset

Jaettujen editorien etu perinteisiin ohjelmiin verrattuna on siinä, että niiden avulla useampi henkilö voi työskennellä samanaikaisesti esimerkiksi saman dokumentin parissa. Tästä seuraa se, että mitä huomaamattomampaa ja tarkempaa saman dokumentin työstäminen on käyttäjille, sitä vaativampaa sen toteutus taas on suunnittelijoille. On pidettävä huoli siitä, etteivät käyttäjien tekemät muutokset aiheuta ristiriitoja ohjelman tilassa. Käyttäjä voi esimerkiksi poistaa tekstinkäsittelyohjelmalla lauseen samanaikaisesti kun toinen taas muokkaa sitä. Konfliktitilanteiden välttämiseksi yhteistoiminnallisuutta sisältävissä ohjelmissa tulisi olla keino samanaikaisten tapahtumien hallinnointiin. Seuraavaksi käydään läpi jaetuille editoreille soveltuvia ratkaisuita samanaikaisuuden hallintaan.

5.1. Tiedon lukitseminen

Yksinkertainen tapa ongelmatilanteiden välttämiseksi on tiedon lukitseminen. Tällöin ainoastaan tiedon lukinnut käyttäjä pystyy muokkaamaan sitä. Samanaikaisen käsittelyn aiheuttamilta konfliktitilanteilta vältytään. Ensemble [Newman-Wolfe *et al.,* 1992] käyttää lukituksia, jotka asetetaan haluttuun kohteeseen sen valinnan yhteydessä. Jos toinen käyttäjä yrittää valita saman kohteen, operaatio epäonnistuu. Periaatteessa siis ääretön määrä käyttäjiä voi käsitellä tiedostoa samanaikaisesti, kun se on jaettu pienempiin lukittaviin kokonaisuuksiin. Lukitukset poistetaan, kun käyttäjä vapauttaa objektin valitsemalla esimerkiksi jonkin muun kohteen. Tehdyt muutokset tulevat nyt

näkyviin kaikille järjestelmän käyttäjille. Lukituksen ollessa päällä he siis näkevät ainoastaan muokkaamattoman kohteen.

Tiedon lukitsemisen haittapuolena on kuitenkin työskentelyn keskeytyminen [Berlage ja Genau, 1993]. Haluttu kohde voi olla jo valittu, jolloin käyttäjän tulee muuttaa suunnitelmiaan. Tiedon lukitseminen on tavallaan ristiriidassa jaettujen editorien periaatteiden kanssa, sillä niiden tulisi mahdollistaa saman kohteen muokkaaminen useamman kuin yhden käyttäjän toimesta. Kokonaisen ikkunan tai työpöytäalan varaaminen vaikuttaa suoraan muiden käyttäjien toimintamahdollisuuksiin [Stefik *et al.,* 1987]. Lukitusten aiheuttamia haittoja voidaan pienentää määrittelemällä yksityiskohtaisemmin kohteet, joita ainoastaan yksi käyttäjä voi muokata samanaikaisesti. Esimerkiksi graafiikkaohjelmiston yhteydessä jokainen objekti voidaan valita erikseen, jolloin rinnakkainen työskentely on mahdollista. Valittujen kohteiden turhaa uudelleenvalintaa voidaan ehkäistä visuaalisin keinoin näyttämällä kohteen yhteydessä sen lukitusta kuvaava indikaattori. Tämä vähentää turhien lukituspyyntöjen määrää [Stefik *et al.,* 1987]. Toinen tapa tehdä lukitsemisesta joustavampaa on tutkia, onko resurssin valinnut käyttäjä aktiivinen [Greif *et al.,* 1986]. Jos käyttäjä ei ole tietyn ajan sisällä suorittanut operaatioita, tai on esimerkiksi ilmoittanut itsensä poissaolevaksi, voidaan lukitus purkaa. Tällöin kohde on jälleen vapaasti valittavissa.

5.2. Konfliktitilanteiden tunnistaminen

Samanaikaisilta tai toistensa kanssa ristiriidassa olevilta syötteiltä ei voida välttyä. Edellä käsitellyn lukituksiin perustuvan ennaltaehkäisevän toiminnan vaihtoehtona on reaaliaikainen ongelmatilanteiden selvittely. Ohjelma reagoi vasta silloin, kun konfliktitilanne havaitaan. Näin ei turhaan estetä saman työskentelykohteen mahdollisesti valideja samanaikaisia muutoksia antamalla se ainoastaan yhden käyttäjän hallittavaksi.

Ellis ja Gibbs [1989] ovat määritelleet muutosmatriisin (transformation matrix), jolla pyritään saavuttamaan tiedon lukitsemattomuus sekä päällekkäisten syötteiden tasapuolinen käsittely. Kahden ristiriitaisen syötteen yhteydessä tehdään ikään kuin kompromissina muutos, jonka jälkeen ohjelman tila on sama kummankin käyttäjän päätteellä. Muutosmatriisin toimintaa voidaan kuvata lyhyesti näin; kun käyttäjä suorittaa operaation, sille luodaan prioriteetti ja operaatio lähetetään muille editoria käyttäville. Tämän operaation saapuessa toisille käyttäjille, tutkitaan, onko lähettävä käyttäjä jo ehtinyt

suorittamaan muita operaatioita. Jos on, aikaisemmin vastaanotettu operaatio laitetaan jonoon odottamaan, että välissä suoritettu toinen operaatio saapuu perille. Muilta käyttäjiltä saadut operaatiot suoritetaan siis aikajärjestyksessä. Jos lähettäjä ei ole tehnyt välissä uusia operaatioita, vastaanottanut käyttäjä suorittaa normaalisti saamansa operaation omassa päässään. Mikäli taas vastaanottava taho on ehtinyt suorittamaan itse tekemänsä operaation ennen toiselta käyttäjältä vastaanotettua operaatiota, tarkistetaan vastaanotetun operaation prioriteetti ja muokataan tämän perusteella operaatiota ennen sen paikallista suorittamista. Muutos perustuu tarkemmin määriteltyihin algoritmeihin, joiden mukaan kahden päällekkäisen operaation perusteella luodaan yksi, molempia osapuolia tyydyttävä operaatio. Tämän ratkaisumallin haittapuolena on se, että käyttäjien suorittamia komentoja joudutaan muuttamaan, eikä niitä voida suorittaa sellaisenaan. Lukituksiin verrattuna tämä tarjoaa kuitenkin joustavamman tavan käyttää jaettua tietoa samanaikaisesti.

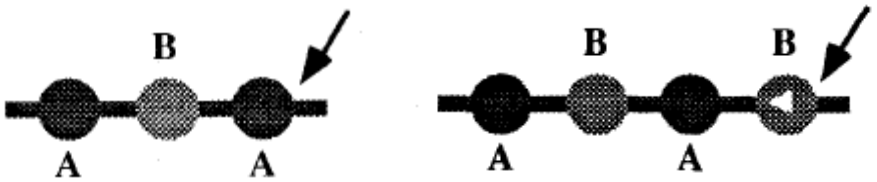
5.3. Komentojen kumoaminen ja uudelleensuorittaminen

Perinteisissä ohjelmissa käyttäjän suorittamat komennot tallennetaan esimerkiksi pinorakenteeseen, josta niitä voidaan tarpeen tullen ottaa käsiteltäviksi komennon kumoamiseksi (undo) tai uudelleensuorittamiseksi (redo). Tämä on suoraviivaista ja yksiselitteistä. Siirryttäessä jaettuihin editorisovelluksiin, käyttäjiä on useita. He suorittavat komentoja omilla päätteillään omaan tahtiinsa. Nyt tilanne monimutkaistuu, sillä kuten jo kappaleessa kolme todettiin, erillisiä komentohistorioita voi olla käyttäjien lukumäärästä riippuen lukuisia.

Mietitään tilannetta, jossa käyttäjä A raahaa objektia näytöllä [Brinck ja Hill, 1993]. Tämän toiminnon kumoamiseksi ohjelman tulisi vain siirtää objekti takaisin alkuperäiselle paikalleen. Käyttäjä B kuitenkin tuhoaa objektin. Nyt jos käyttäjä A yrittää kumota objektin raahauksen, objekti tulisi palauttaa uudelleen näkyviin ja viedä alkuperäiselle paikalleen, tai se tulisi vaihtoehtoisesti vain jättää poistetuksi. Tämä on esimerkkitilanne, missä ohjelman tulee pystyä käsittelemään komennon kumoaminen. Käyttäjä B on voinut suorittaa äärettömän määrän komentoja ennen käyttäjän A kumoamispyyntöä, jolloin kaikkien jo suoritettujen komentojen validius täytyy tarkistaa. Tämä tarkoittaa sitä, että suoritettua komentoa ei aina voida kumota jälkeenpäin.

Berlage ja Genau [1993] ovat hyödyntäneet GINA-järjestelmää [Spenke ja Beilken, 1990], joka sisältää tavan komentojen kumoamiseen. Käyttäjä B on suorittanut komennon kahden käyttäjän A komennon välissä (kuva 5, vasen

puoli). Oletetaan, että B haluaa kumota komentonsa. Normaalia kumoamismenettelyä, jossa käyttäjä B vain poistaisi oman komentonsa kahden A:n komennon välistä, ei voida nyt käyttää. Tämä sekoittaisi samalla käyttäjän A komentohistorian, sillä se on riippuvainen käyttäjän B tekemästä komennosta. Tilanne voidaan ratkaista luomalla B:n aloitteesta uusi komento, joka on B:n alkuperäisen komennon vastakohta (kuva 5, oikea puoli). Tämä uusi komento lisätään komentohistoriaan. Näin B:n aikaisempi komento on kumottu, ja A:n komentohistoria on yhä ristiriidaton.



Kuva 5. Käyttäjän B komennon kumoaminen. [Berlage ja Genau, 1993]

6. Yleisten arkkitehtuurimallien käyttö

Edeltävissä kappaleissa on käsitelty jaetuille editorisovelluksille tyypillisiä seikkoja, jotka tulevat poikkeuksetta esille yhteistoiminnallisten editorien parissa työskennellessä. Yksittäisten ominaispiirteiden huomioimisella ja huolellisella suunnittelulla saadaan luotua moitteettomia yhteistoiminnallista toimintaa tukevia editoreita. Toteutuksen apuna voidaan kuitenkin myös käyttää muista yhteyksistä tunnettuja suunnittelumalleja, jotka ovat yleisesti hyväksyttyjä. Näin editoreita koskevan sovellusaluetuntemuksen lisäksi saadaan tukea hyviksi todetuista toimintamalleista. Seuraavissa kahdessa alikohdassa käsitellään kaksi erilaista tapaa hyödyntää yleisiä suunnittelumalleja jaettujen editorien luomisessa.

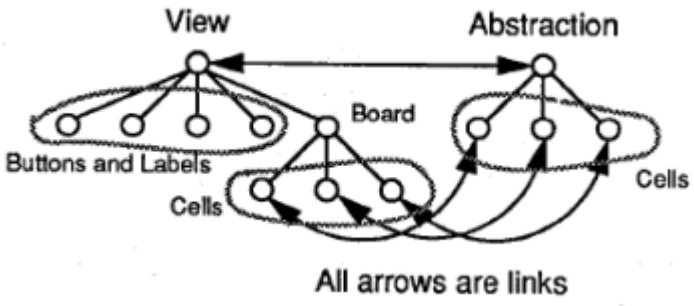
6.1. Abstraktio-Linkki-Näkymä -malli

Abstraktio-Linkki-Näkymä -malli [Hill, 1992] on soveltuva jaettujen editorien suunnitteluun ja toteuttamiseen. Vastaavuudet mallin ja todellisen ohjelman osien kanssa ovat melko intuitiivisia. Abstraktio sisältää tiedot ohjelman sisäisestä tilasta, esimerkiksi ikkunoiden määrästä tai koosta. Näkymä vastaa näiden tietojen esittämisestä näyttölaitteella. Linkit ovat objekteja, jotka välittävät tietoa ohjelman ja käyttöliittymän kesken. Jaettujen editorisovellusten toteutuksessa Abstraktio-Linkki-Näkymä -malli on käyttökelpoinen, sillä se

erottaa loogisesti ohjelman eri osat. Linkkien avulla on mahdollista yhdistää useampi käyttäjä samaan abstraktioon.

RENDEZVOUS-projektin [Hill *et al.,* 1993] puitteissa ALV:tä on käytetty luomaan editorisovelluksia, joissa käyttöliittymä- ja näyttöosuudet ovat erotettu selkeästi muusta ohjelmasta. Abstraktio säilyttää käyttäjiltä näkymättömissä olevaa tietoa ohjelman tilasta. Vaikka samaa abstraktiota hyödyntäisi useampi käyttäjä, on tallennettu tieto kaikille yhteistä. Personoituja ominaisuuksia ei siis hallinnoida abstraktioiden kautta. Näkymät esittävät tietoa, sekä mahdollistavat tiedon, tai sen esitystavan, muokkaamisen. Käyttäjä ei ole sidottu vain yhteen näkymää, vaan esimerkiksi rinnakkaisissa ikkunoissa voidaan esittää tietoa useista näkymistä. Linkit taas siirtävät tietoa abstraktion ja näkymän välillä. Linkkien tulee tarkkailla, milloin uutta tietoa on siirrettävissä, eli milloin käyttäjä on tehnyt muutoksia. Käytännössä tässä käytetään rajoitteita (constraints), jotka ylläpitävät yhtenäisyyttä abstraktion ja näkymän toisiaan vastaavien arvojen välillä. Lisäksi linkkien tulee muuntaa välitettävä tieto joko abstraktion tai näkymän tarpeiden mukaiseen muotoon.

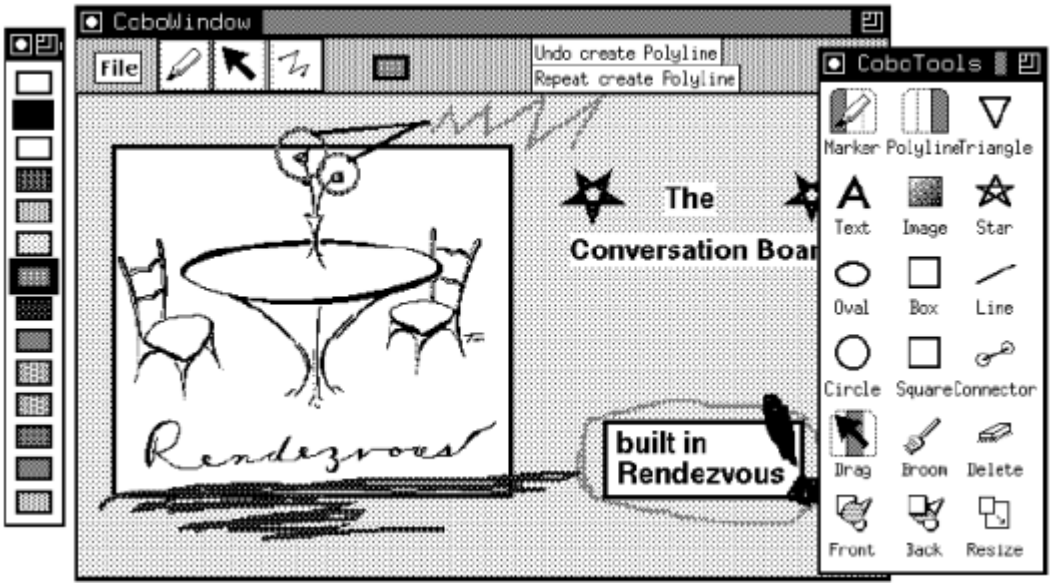
RENDEZVOUS-arkkitehtuuri perustuu hierarkkisiin puumalleihin. Abstraktio ja näkymä ovat mallinnettu sisäkkäisten puurakenteiden avulla, missä yksityiskohtaisimmat objektit ovat alimpina lehtinä (kuva 6).



Kuva 6. Hierarkkinen malli näkymän ja abstraktion välillä. [Hill *et al.,* 1993]

Toisiaan vastaavat abstraktion ja näkymän objektit ovat yhdistetty omilla linkeillään. Puut eivät ole välttämättä täysin toistensa kopioita, sillä näkymässä voi olla käyttöliittymän yksityiskohtia koskevaa tietoa. Kaikkien näkymän objektien ei siis tarvitse olla linkitettyinä abstraktioon. Puut muuttuvat käytön yhteydessä, jolloin linkkienkin tulee mukautua uusiin tai poistettuihin objekteihin.

RENDEZVOUS-arkkitehtuuria on hyödynnetty jaetun graafisen editorin luomisessa [Brinck ja Hill, 1993]. Conversation Board on ohjelma, joka sisältää yksinkertaisia työkaluja tekstin ja grafiikan luomiseen (kuva 7). Conversation Board mahdollistaa samanaikaisen editoinnin usean käyttäjän toimesta. Käytännössä kahden käyttäjän välinen yhteys käynnistetään luomalla linkki käyttäjien näkymien välille. Samat abstraktion ja näkymän välille ensisijaisesti tarkoitetut linkit toimivat myös suoraan kahden eri näkymän välillä. Toisen käyttäjän tehdessä muutoksen omaan näkymäänsä, siirtyy sama muutos välittömästi myös toisen käyttäjän näkymään. Useamman kuin kahden käyttäjän kohdalla periaatteessa riittää, kun uusi käyttäjä linkittää näkymänsä toiseen jo keskenään yhdistettyyn näkymään. Yhden käyttäjän näkymästä voidaan myös tehdä keskusnäkymä, johon kaikki muut käyttäjät ottavat yhteyden.



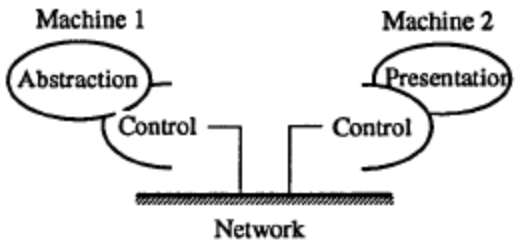
Kuva 7. Conversation Board -editori. [Brinck ja Hill, 1993]

Abstraktio-Linkki-Näkymä -mallilla on useita hyviä puolia toteutettaessa jaettuja editoreita. Kaikki kommunikointi tapahtuu linkkien avulla. Mitään tiedonsiirtoon liittyvää ei ole sisällytetty abstraktioon tai näkymään. Tästä johtuen ne pysyvät erillisinä osina, jotka ovat toisistaan riippumattomia. Uudelleenkäytettävyys ja hallittavuus ovat hyvällä tasolla. Useiden käyttäjien järjestelmissä on lisäksi paljon samanaikaisuutta, mikä on väistämätöntä. Sen järkevän hallinnan toteuttaminen on usein haastavaa. Rajoitteiden ja linkkien avulla tapahtumien hallinta on suoraviivaista ja helposti omaksuttavaa.

6.2. PAC-agenttitoteutus

Agenttiperustaisten toteutusten avulla pystytään luomaan CSCW-ohjelmia, kuten jaettuja editoreita, jotka täyttävät informaation vaihdon sekä näkymien jakamisen vaatimukset. Esittelen yhden jaetun editorisovelluksen, joka pohjautuu agentteihin. COOP-ympäristö [Legault, 1993] on tarkoitettu samanaikaisten groupware-ohjelmien toteutukseen. Sen pohjana on moniagenttiperustainen arkkitehtuurimalli PAC (Presentation-Abstraction-Control) [Bass ja Coutaz, 1993]. COOP-ympäristön pohjalle on rakennettu jaettu graafinen COOPDRAW-editori [Ramstein, 1993], jonka avulla myös useamman kuin yhden käyttäjän on mahdollista työskennellä saman kohteen parissa.

COOP-ympäristön käyttämän PAC-mallin mukaan rakennetut järjestelmät koostuvat agenttien hierarkioista. Jokaisella PAC-agentilla on kolme osaa; yksi syötteiden ja tulosteiden käsittelyyn (Presentation), yksi toiminnallisten arvojen määrittelyyn (Abstraction) ja yksi kahden muun osan välisen kommunikoinnin järjestämiseen (Control). COOP-ympäristössä on neljän tyyppisiä agentteja, jotka pohjautuvat PAC-mallin mukaisiin agentteihin. Perusagentit käsittävät tekstin ja grafiikan, kuten viivat, suorakulmiot, ellipsit ja polygonit. Komposiittiagentit hallitsevat ja organisoivat muista agenteista koostuvaa ryhmää. Jaettu agentti taas on tarkoitettu tilanteisiin, joissa yhdellä agentilla on useampi kuin yksi vanhempi. Tällöin agentti huolehtii yhteyksistä, joiden välillä pyynnöt kulkevat vanhemmalta lapselle ja vastaavasti lapselta vanhemmalle. Käytännössä jaetut agentit yhdistävät eri näkymiä yhteen, jolloin käyttäjä näkee näkymät yhtenä. Neljäs ja viimeinen agentti on hajautettu agentti. Kolme edellä esiteltyä agenttia toimivat samalla koneella, jolloin ne eivät voi kommunikoida kahden eri käyttäjän välillä. Hajautetun agentin PAC-mallin mukaiset abstraktio- ja presentaatio-osat ovat hajautettu kahdelle eri koneelle. Nyt kontrolliosan tehtävänä on huolehtia kommunikoinnista eri koneilla sijaitsevien osien välillä (kuva 8). Editorisovelluksen yhteistoiminnallisuuden toteuttamisessa hajautetut agentit ovat olennaisia.



Kuva 8. Hajautettu COOP-agentti. [Ramstein, 1993]

COOPDRAW-editori on jaettu eri abstraktiotasoihin. Toimintojen analysoinnin perusteella on tehty jako neljään eri tasoon. Alimman tason abstraktioon kuuluvat perusagentit, kuten nelikulmio ja ellipsi. Dokumenttitaso on vastuussa attribuuteista, joita ovat esimerkiksi sivunumerot ja marginaalit. Muokkaustaso käsittää grafiikan ja tekstin luomisen, siirtämisen ja muokkaamisen. Muokkaustaso vastaa myös valittujen objektien hallinnoimisesta. Ylimpänä oleva editoritaso huolehtii toiminnoista kuten leikkaamisesta, kopioimisesta ja liimaamisesta. Nämä abstraktiotasot ovat keskeisiä näkymien jakamisessa käyttäjien välillä. Mitä alemmalle tasolle abstraktioiden jakaminen yltää, sitä tiukemmin käyttäjät toimivat yhdessä. Jos taas ainoastaan ylin taso, editoritaso, on jaettu, ovat jaetun editorin käyttäjät hyvin vapaassa työskentelymuodossa. Tällöin he eivät näe muiden käyttäjien alemman tason editointeja. COOPDRAW käyttää keskitettyä arkkitehtuurimallia, missä ohjelman ydin on omalla koneellaan. Muilla koneilla ovat ainoastaan graafiseen näyttämiseen ja syötteiden käsittelyyn tarkoitetut rutiinit.

Agenttien avulla voidaan mallintaa koko editorisovellus hierarkkisesti järjestettyjen agenttien avulla. Agentit toimivat eri tasoilla riippuen niiden sisältämän toiminnan abstraktiotasosta. Jaettujen ja hajautettujen agenttien avulla yhteistoiminnallisen editorin luominen on mahdollista. Pienten agenttikomponenttien käyttäminen tekee ohjelmasta uudelleenkäytettävän, koska yksittäisiä osia voidaan hyödyntää helposti. Lisäksi hierarkkinen malli auttaa hahmottamaan ohjelman toimintaa ja se myös mahdollistaa eri näkymätasojen jakamisen usean käyttäjän välillä.

7. Yhteenveto

Jaettuja editoreita hyödyntämällä saadaan tietokoneiden avulla tapahtuvaan työskentelyyn lisää käyttäjien välistä vuorovaikutusta. Ihmisten luonnolliseen ongelmanratkaisuun kuuluu kommunikointi toisten kanssa, sekä muiden ihmisten toimien havainnointi. Tämän toimintamallin sisällyttäminen ohjelmistoihin mahdollistaa tehokkaamman ja joustavamman tavan käsitellä yhteistä tietoa. Perinteistä työskentelyä rajoittavien asioiden, kuten osoittamisen ja reaaliaikaisuuden puute voidaan korjata jaettuja editoreja käyttämällä.

Tiedostamalla jaettujen editorien luonteeseen kuuluvat piirteet ja huomioimalla niiden vaikutukset ohjelman toteuttamiseen, voidaan välttyä toimivuutta haittaavilta ongelmakohdilta. Eri arkkitehtuuriratkaisuiden avulla jaettujen editorien toiminta voidaan määritellä halutun kaltaiseksi. Käyttötilanteista riippuen tehdään valinta käyttäjien välisen vuorovaikutuksen läheisyydestä tai tiedon monistamisesta jokaisen käyttäjän päätteelle. Yhteistoiminnallisten editorien suunnittelussa ja toteutuksessa on myös menestyksekkäästi käytetty yleisiä arkkitehtuurimalleja, kuten abstraktioiden, linkkien ja näkymien käsitteitä, sekä agentteja. Jaettujen editorien toteutusmalleja ja -tapoja on useita, joista parhaiten soveltuvan valitseminen vaatii paitsi teknisten vaihtoehtojen, myös ihmisten välisen vuorovaikutuksen ymmärtämistä.

Erityyppisistä jaetuista editoreista tekstin ja grafiikan käsittelyyn tarkoitetut sovellukset ovat perinteisimpiä. Tietojenkäsittelykapasiteetin kasvaessa tulemme näkemään yhä enemmän raskaammille sovellusalueille, kuten hypermediaan ja liikkuvan kuvan käsittelyyn, tarkoitettuja editoritoteutuksia. Sovellusalueiden muuttumisesta huolimatta nykyiset perustason arkkitehtuuriin liittyvät asiat pysyvät varmasti ajankohtaisina myös tulevaisuudessa.

Viiteluettelo

[Ahuja *et al.,* 1988] S. R. Ahuja, J. Robert Ensor, David N. Horn, The rapport multimedia conferencing system. *Conference Sponsored by ACM SIGOIS and IEEECS TC-OA on Office Information systems,* 1998, ACM Press, 1-8.

[Ahuja *et al.,* 1990] S. Ahuja, R. Ensor, S. E. Lucco, A Comparison of Sharing Mechanisms in Real-Time Desktop Conferencing Systems. *Proceedings of the Conference on Office Information Systems*, 1990, ACM Press, 238-248.

[Bass ja Coutaz, 1993] L. J. Bass, J. Coutaz, *Developing Software for the User Interface.* Addison-Wesley, 1991.

[Berlage ja Genau, 1993] Thomas Berlage, Andreas Genau, A framework for shared applications with a replicated architecture. *Proceedings of the 6th annual ACM symposium on User interface software and technology,* 1993, ACM Press, 249-257.

[Bier ja Freeman, 1991] Eric A. Bier, Steven Freeman, MMM: a user interface architecture for shared editors on a single screen. *Proceedings of the 4th annual ACM symposium on User interface software and technology,* 1991, ACM Press, 79-86.

[Brinck ja Hill, 1993] T. Brinck, R. Hill, Building Shared Graphical Editors Using the Abstraction-Link-View Architecture. *Proceedings of the Third European Conference on Computer-Supported Cooperative Work,* 1993, Kluwer Academic, 311-324.

[Crowley *et al.,* 1990] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, Raymond Tomlinson, MMConf: an infrastructure for building shared multimedia applications. *Proceedings of the 1990 ACM Conference On Computer-supported cooperative work,* 1990, ACM Press, 329-342.

[Ellis ja Gibbs, 1989] C. A. Ellis, S. J. Gibbs, Concurrency control in groupware system. *Proceedings of the 1989 ACM SIGMOD international conference on Management data,* 1989, ACM Press, 399-407.

[Elrod *et al.,* 1992] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, B. Welch, Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. *Proceedings of the ACM Conference on Human Factors in Computing Systems,* 1992, ACM Press, 599-607.

[Greenberg, 1990] S. Greenberg, Sharing Views and Interactions with Single User Applications. *Proceedings of the Conference on Office Information Systems*, 1990, ACM Press, 227-237.

[Greif *et al.,* 1986] Irene Greif, Robert Seliger, William E. Weihl, Atomic data abstractions in a distributed collaborative editing system. *Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages,* 1986, ACM Press, 160-172.

[Hill, 1992] R. D. Hill, The Abstraction-Link-View paradigm: using constraints to connect user interfaces to applications. *Proceedings of the SIGCHI conference on Human factors in computing systems,* 1992, ACM Press, 335-342.

[Hill *et al.,* 1993] R. D. Hill, T. Brinck, J. F. Petterson, S. L. Rohall, W. T. Wilner, The Rendezvous language and architecture. *Communications of the ACM,* **36** (1), 1993, ACM Press, 63-67.

[Legault, 1993] S. Legault, COOP: a development environment for multimedia synchronous CSCW applications. *In proceedings of the conference on Multimedia Communication'93*, 1993, Banff, 289-295.

[Newman-Wolfe *et al.,* 1992] R. E. Newman-Wolfe, M. L. Webb, M. Montes, Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor. *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, 1992, ACM Press, 265-272.

[Ramstein, 1993] Christophe Ramstein, COOPDRAW: a multiagent architecture for a shared graphical editor. *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing,* **2**, 1993, IBM Press, 758-766.

[Spenke ja Beilken, 1990] M. Spenke, C. Beilken, An Overview of GINA – the Generic Interactive Applications. *Proceedings of the workshop on user interface management systems and environments on User interface management and design*, 1990, Springer-Verlag, 273-293.

[Stefik *et al.,* 1987] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, L. Suchman, Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of the ACM,* **30** (1), 1987, ACM Press, 32-47.

Moniagenttijärjestelmät ja PAC-Amodeus

Rami Saarinen

Tiivistelmä

Multimodaaliset käyttöliittymät vaativat tavallisia käyttöliittymiä enemmän suoritustehoa ja resursseja. Useamman modaliteetin käsitteleminen samanaikaisesti vaatii myös rinnakkaista toiminnallisuutta. Monoliittisina järjestelminä teoteutettuna multimodaaliset sovellukset ovat olleet perinteisesti erittäin hankalia ja monimutkaisia rakennettavia ja ylläpidettäviä. Tässä artikkelissa esitellään agenttipohjainen PAC-Amodeus malli, joka on suunniteltu multimodaalisten sovellusten rakentamista silmälläpitäen. Malli on ollut kehityksessä 1990 luvun alusta saakka ja se on todettu toimivaksi useammassa sovelluksessa. PAC-Amodeuksen lisäksi esitellään agenttipohjainen ohjelmistoarkkitehtuuri, joka on rakennettu Tampereen Yliopistolla PROAGENTS-projektin yhteydessä. Lopuksi pohditaan, miten agenttiarkkitehtuuria voidaan käyttää nykyaikaisen multimodaalisen käyttöliittymän toteuttamiseksi ja miten PAC-Amodeus olisi toteutettavissa ohjelmistoarkkitehtuurin avulla.

Avainsanat ja -sanonnat: Moniagenttijärjestelmät, multimodaalisuus, fuusio, PAC-Amodeus

1. Johdanto

Erilaiset vuorovaikutustekniikat ovat olleet jo jonkin aikaa noususuhdanteessa, sillä perinteiset käyttöliittymät koetaan liian rajallisiksi ja luonnollista käyttöä vaikeuttaviksi. Myös uudet esitystavat, kuten virtuaaliympäristöt, ja uudet laitteet ja niihin perustuvat tekniikat, kuten jokapaikan tietotekniikka, vaativat käyttöliittymiltä parempia ja joustavampia vuorovaikutustekniikoita. On myös erityisryhmiä, jotka ovat tähän asti jääneet tietotekniikan käytössä ulkopuolelle, mutta jotka erityisesti hyötyisivät tietotekniikan tuomista mahdollisuuksista. Tällaiset erityisryhmät, kuten sokeat, vaativat usein erityisiä käyttöliittymiä ja erilaisten vuorovaikutustekniikoiden hyväksikäyttöä.

Useamman vuorovaikutustekniikan sallivien järjestelmien rakentaminen on perinteisesti ollut vaikeaa, mutta agenttiarkkitehtuureiden kehittymisen myötä on auennut aivan uusia mahdollisuuksia todellisten vuorovaikkutteisten järjestelmien kehittämiseen. Tässä artikkelissa esitellään jo pitkään kehityksessä olleen multimodaalisten järjestelmien rakentamiseen suunniteltu arkkitehtuurimalli, PAC-Amodeus, joka perustuu PAC-agenttien käyttämiseen mm. modaliteettien yhdistämisessä. Tämän lisäksi esitellään Tampereen Yliopistossa kehitetty agenttiarkkitehtuuri ja pohditaan miten PAC-Amodeus olisi rakennettavissa agenttialustan avulla.

2. Agenteista

Agentti voidaan määritellä usealla toisistaan poikkeavalla tavalla. Erään määritelmän mukaan agentilla on seuraavia ominaisuuksia: autonomisuus, reaktiivisuus, pysyvyys, persoonallisuus, kommunikointikyky, yhteistyökyky, päättelykyky, mukautuvuus ja liikkuvuus [Bradshaw, 1997]. Agentti on siis itsenäinen, omaa toimintaansa ohjaava ja muiden agenttien kanssa kommunikoiva jatkuvasti ajossa oleva ohjelma. Agentin katsotaan omaavan jonkinlaisia keinoja tarkkailla ja manipuloida ympäristöään, oli kyseessä sitten lämpöpatterin termostaattiagentti tai vaikkapa satojen agenttien yhteisö jollakin erityisellä agenttialustalla.

Yksinkertaisimmillaan agentti tulkitsee ympäristöstä tulevia ärsykkeitä ja reagoi niihin jollain tavalla. Tämän kaltaisia agentteja kutsutaan reaktiivisiksi agenteiksi. Agentteja, jotka yrittävät sekä ennakoida että ennustaa tulevia tilanteita ja ohjaavat toimintaansa näiden ennustusten mukaan, kutsutaan proaktiivisiksi agenteiksi. Voidaan myös ajatella että proaktiivisuus on agentin toimintaa, joka ei ole suoraa reagointia ympäristössä tapahtuvaan muutokseen, vaan muutos johtaa agentin ennakoivaan tai muulla tavalla älykkältä vaikuttavaan toimintaan. Proaktiiviseksi toiminnaksi katsotaan myös agentin kyky toimia päämääräsuuntautuneesti.

2.1. Agenttijärjestelmät

Useita toistensa kanssa kommunikoivia agentteja sisältävät järjestelmät on yleensä rakennettu siten että agentit sijaitsevat niin sanotun agenttialustan sisällä, eivätkä ne siis ole täysin itsenäisiä tietokoneella ajossa olevia prosesseja. Alustan käytöstä seuraa monia etuja. Se esimerkiksi mahdollistaa yleisen tavan kontrolloida agentin elinkaarta (agentin käynnistys, pysäytys, pysäytyksestä

jatkaminen ja suorituksen lopettaminen). Alusta voi myös tarjota erilaisia, esimerkiksi viestintään tai eri protokolliin liittyviä, palveluita agenteille. Alusta on yleensä ajossa yhdellä koneella ja täten kaikki alustan agentitkin sijaitsevat samalla koneella. Alustat voivat olla yhteydessä toisiinsa lähiverkon avulla. Joissain järjestelmissä agentti voi siirtyä alustalta toiselle.

Useista alustoista koostuvien järjestelmien tulee ratkaista se, miten agentit löytävät toisensa ja kuinka ne voivat keskustella toistensa kanssa. Monissa järjestelmissä tämä on ratkaistu keskuspalvelimen avulla, joka osaa välittää agenttien viestejä ja joka tarjoaa erilaisia palveluita agenttien ja alustojen käytettäväksi. Esimerkiksi agentin siirtyminen alustalta toiselle luultavasti hoidettaisiin keskuspalvelimen kautta, sillä keskuspalvelimen pitäisi tietää miten kohdealustaan saadaan yhteys ts. missä kohdealusta on.

2.2. Agenttien kommunikointikielistä ja standardeista

Verkkoliikenteen, ohjelmointikielten ja -paradigmojen ja hajautettujen järjestelmien kehittyessä on laadittu useita standardeja laite- ja ohjelmistoyhteensopivuuden aikaansaamiseksi. Nykyiset agenttialustat jossain määrin nojaavat tai ainakin ottavat kantaa joihinkin seuraavaksi esiteltäviin standardeihin.

2.2.1. OMG CORBA

CORBA (Common Object Request Broker Architecture) on OMG:n standardi hajautettuille järjestelmille [CORBA, 2002]. CORBA:a käyttävä ohjelma voi operoida toisen CORBA:a tukevan ohjelman kanssa verkon yli. CORBA pyrkii yleistämään objektien välisen liikennöinnin niin, että toteutuksen ohjelmointikieli tai alla oleva käyttöjärjestelmä ei vaikuta prosessiin. Objektien välinen kommunikaatio tapahtuu IIOP-protokollalla (Internet Inter-ORB Protocol).

2.2.2. KQML ja KIF

DARPA:n tukema Knowledge Sharing Effort on poikinut mm. standardit KIF ja KQML [Patil *et al*., 1993]. Ensinnäkin KIF (Knowledge Information Format) määrittelee yhteisen kielen eri tahojen väliselle tiedon jakamiselle ja kuvaamiselle. KIF perustuu ensimmäisen asteen predikaattilogiikkaan. KQML [Finin *et al.,* 1994] vuorostaan on kolmikerroksinen kommunikointiprotokolla, joka sisältää kommunikaatio- , viesti- ja sisältötason. Kommunikaatiotasossa

määritellään viestin alkeisominaisuuksia, kuten lähettäjän ja vastaanottajan tiedot. Viestitasolla kuvataan mm. mikä kommunikatiotapahtuma on kyseessä. Sisällössä vuorostaan on itse välitettävä informaatio jollakin formaalilla tavalla kuvattuna (kuten KIF). Myöhemmät sovellukset ja standardit, kuten FIPA:n ACL omaavat paljon yhteistä KQML:n kanssa.

2.2.3. FIPA

FIPA:n (Foundation of Intelligent Physical Agents) tarkoituksena oli laatia standardit agenttijärjestelmien väliselle kommunikaatiolle ja täten se ei ota kantaa agentin, tai edes agenttialustan, sisäiseen rakenteeseen. FIPA:n ensimmäinen virallinen spesifikaatio julkaistiin vuonna 1997 [FIPA00019; FIPA00003; FIPA00017], johon itsensä FIPA-yhteensopiviksi ilmoittavat alustat yleensä viittaavat. Tosin vuonna 1998 ilmestyi uusi spesifikaatio ja vuonna 2000 ilmestyi FIPA:n abstrakin arkkitehtuurin määrittely [FIPA00001]. Vuoden 2000 määrittely poikkeaa aiemmista määrittelyistä ja uusilta FIPA -alustoilta vaaditaan yhteensopivuutta vuoden 2000 määrittelylle.

3. Modaliteeteista

Modaliteetilla tässä tapauksessa tarkoitetaan tapaa antaa syötettä järjestelmälle tai saada siltä palautetta. Nykyajan vallitsevissa käyttöliittymissä käyttäjä esimerkiksi antaa syötettä hiirellä ja saa palautetta toimistaan visuaalisesti. Tämän kaltainen vuorovaikutus jättää hyödyntämättä useimmat ihmisen aisteista ja ihmisen luonnollisen tavan tehdä asioita.

Nigay ja Coutaz [1993] määrittelevät multimodaalisuudeksi sen, että järjestelmä pystyy kommunikoimaan käyttäjän kanssa usean eri kommunikaatiokanavan avulla ja se automaattisesti kykenee erottamaan ja johtamaan kommunikaation tarkoituksen. Multimodaalisuus on kahden suuntaista: järjestelmälle voidaan antaa komentoja usealla eri tavalla ja käyttäjälle tarjotaan informaatiota useampaa aistikanavaa apuna käyttäen.
Multimodaalisuutta käsiteltäessä puhutaan usein myös fuusiosta ja fissiosta. Fissiolla tarkoitetaan tapahtuman kuvaamista käyttäjälle eri palautetapojen avulla. Fuusiolla vuorostaan tarkoitetaan järjestelmän kykyä yhdistää eri kommunikaatiokanavien kautta saatuja komentoja. Coutaz ja Nigay [1993] jakavat multimodaalisuuden neljään eri luokkaan fuusion tason ja modaliteettien käyttötavan mukaan.

- Modaliteettien käyttäminen peräkkäin fuusion yhdistäessä syötteet. Tämä kuvaa multimodaalisuutta, missä tehtäviä voidaan tehdä usealla modaliteetilla siten että modalitteettien pitää tapahtua peräkkäin. Fuusio siis yhdistää eri modaalisyötteitä ketjuiksi, joista muodostuu yksittäisiä komentoja.
- Modaliteettien käyttäminen peräkkäin ilman fuusiota kuvaa multimodaalisuutta, missä vain yhtä modaliteettia voidaan käyttää ekslusiivisesti kerrallaan.
- Modaliteettien käyttäminen rinnakkain fuusion yhdisäessä ne tarkoittaa synegististä multimodaalisuutta, missä käskyjä voidaan antaa samanaikaisesti usealla modaliteetilla ja niiden fuusiosta muodostuu yksittäinen käsky.
- Ja kun modaliteetteja käytetään rinnakkain ilman fuusiota, on kyseessä multimodaalisuus, missä käskyjä voidaan antaa samanaikaisesti usealla modaliteetilla. Käskyt itsessään ovat erillisiä, muista käskyistä ja modaliteeteista riippumattomia, komentoja.

Eri modaliteetteja saatetaan myös käyttää toistensa tukemiseen ja komennon tulkinnan virhemarginaalin pienentämiseen. Multimodaalisissa järjestelmissä on usein myös modaliteetteja, jotka ovat dominantteja muihin modaliteetteihin nähden. Multimodaalisissa sovelluksissa keskitytään usein fuusioon.

4. PAC-Amodeus

PAC-Amodeus yhdistää PAC-mallin [Coutaz, 1987] Arch malliin [Bass *et al., 1992*]. Siinä missä Arch kuvaa sovelluksen jakoa käsitteellisiin komponentteihin, antaa PAC mallia rinnakkaiseen toiminnallisuuteen ja agenttihierarkioihin. Seuraavaksi käsitellään Arch ja PAC-mallit ja tämän jälkeen katsotaan miten ne on yhdistetty PAC-Amodeukseksi. Lopuksi katsotaan PAC-Amodeuksen laajennuksia ja sovelluksia.

4.1. Arch

90-luvun alussa interaktiivisille ohjelmistoille suunniteltu Arch malli [Bass *et al., 1992*] ja siitä yleistetty Slinky metamalli jakavat arkkitehtuurin osiin sen mukaan minkälaista dataa kussakin osassa käsitellään. Mallin suunnittelijat toteavat että interaktiivisessa systeemisssä käsitellään ainakin kolmen tyyppistä dataa: sovelluskohtaista dataa (esim. tietokannan nimi ja tyyppiarvoja), syöte- ja

tulostelaitteiden dataa (esim. pikseleitä ruudulla) ja erilaisia datan välimuotoja (esim. data kahdessa sarakkeessa yhden elementin valinnalla).

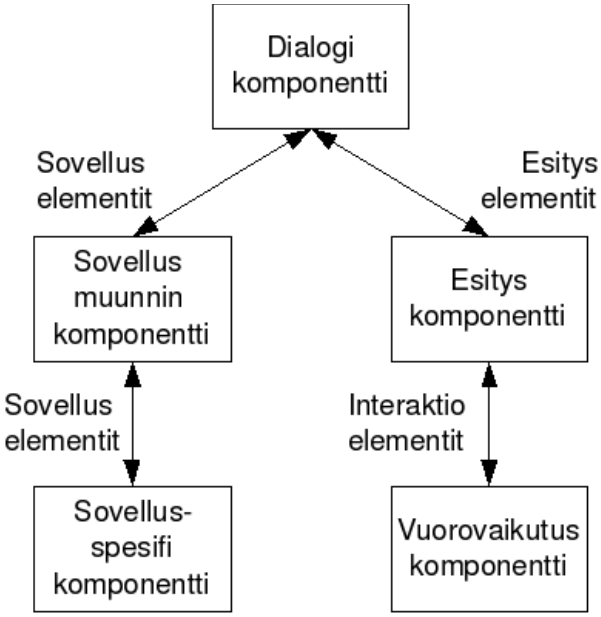
Data liikkuu kahdensuuntaisesti sovelluskohtaisten komponenttien ja syöte- ja tulostelaitteiden välillä. Sovelluskohtaiseen dataan kodistetaan muutosoperaatioita, kunnes jossain vaiheessa saavutetaan tulostelaitteiden taso. Vastavuoroisesti syötelaitteilta tulevaa dataa muokataan kunnes saadaan aikaiseksi sovelluskohtaista dataa. Käsiteltävää dataa täytyy järjestellä monella eri tasolla. Esimerkiksi käyttäjän yksittäiset toimet tulee ymmärtää osana isompaa tehtävää ja näitä toimia täytyy järjestää tehtävän mukaisella tavalla. Samoin syötelaitteilta tulevaa data täytyy käsitellä. Esimerkiksi hiiren raahaaminen samaan aikaan kun näppäin on painettu pohjaan tulee osata yhdistää samaan tapahtumaan.

Näistä puitteistä lähtien mallin suunnittelijat listaavatkin minimijoukon pakollisia operaatioita, jotka jokaisen interaktiivisen järjestelmän pitää toteuttaa. Operaatioiden tärkeys vaihtelee sovelluskohtaisesti, mutta operaatiot on jollain tavalla suoritettava.

- Hallinnoi, manipuloi ja nouda sovelluskohtainen data ja suorita muut sovelluskohtaiset toiminnot.
- Uudelleenjärjestä sovelluskohtainen data käyttölittymään soveltuvaan muotoon.
- Tarjoa tehtäväkohtainen järjestäminen. Yksi tehtävä voi koostua joukosta yksittäisiä toimia (esimerkiksi kappaleen raahaaminen paikasta toiseen) tai tehtävään voi kuulua multimodaalista vuorovaikutusta kuten puheen, katseen ja eleen yhteistulkintaa.
- Tarjoa monen näkymän välinen yhteneväisyys.
- Tee päätöksiä syöte- ja tulostemedioiden suhteen.
- Valitse interaktioelementit.
- Tarjoa fyysinen käyttöliittymä käytäjälle.
- Muodosta muunto sovelluskohtaisesta formalismista käyttäliittymän formalismiin.

Arch malli lähtee siitä liikkeelle että käyttöliittymäsuunnittelijat usein joutuvat työskentelemään tilanteessa, missä varsinainen sovellus ja käytettävä käyttöliittymäkirjasto asettavat selkeät rajoitteet käyttöliittymien suunnittelulle. Tästä seuraakin että kaarimallin (arch) peruspilarit ovat sovellusspesifi komponentti eli sovelluksen ydin ja vuorovaikutuskomponentti eli syöte- ja tulostelaitteet. Sovellusspesifi komponentti hallinnoi, manipuloi ja noutaa

sovelluskohtaista dataa ja suorittaa muut sovelluskohtaiset toiminnot. Vuorovaikutuskomponentti mahdollistaa fyysisen vuorovaikutuksen käyttäjän kanssa.



Kuva 1: Arch-malli [Bass *et al.*, 1992]

Kaaressa on kolme muuta osaa. Dialogikomponentti vastaa tehtäväkohtaisesta järjestämisestä. Se käsittelee käyttäjän toimien järjestämisen sekä sen osan sovelluskohtaisesta järjestämisestä, mikä riippuu käyttäjän toimista. Dialogikomponentti vastaa myös monen näkymän välisestä yhteneväisyydestä ja datan muuntamisen sovelluskohtaisesta formalismista käyttöliittymäkohtaiseen formalismiin ja toiseen suuntaan.

Esityskomponentti on välittäjä tai puskuri Dialogikomponentin ja vuorovaikutuskomponentin välimaastossa. Se välittää yleisluontoisia elementtejä dialogikomponentille. Tälläisiä elementtejä on esimerkiksi valinta-elementti, joka kuvaa käyttöliittymässä valikon kautta tai valintapainikkeiden avulla tehtyjä valintoja. Esityskomponentti myös päättää palautteiden esitystavasta.

Sovellusmuunnin-komponentti toimii välittäjänä sovellusspesifin komponentin ja dialogikomponentin välissä. Tässä komponentissa toteutetaan ne sovelluskohtaiset tehtävät, joita tarvitaan käyttäjän toimien tukemiseen, mutta joita ei ole sovelluspesifissä komponentissa. Komponentti välittää myös sovelluksen ytimestä lähtevät dialogitehtävät dialogikomponentille, uudelleenjärjestää sovelluskohtaista dataa ja havaitsee ja raportoi semanttiset virheet.

Vaikka sovelluselementtejä käytetään sekä sovellusspesifissä komponentissa että sovellusmuunnin-komponentissa, on niiden käyttötapa eri. Sovellusspesifissä komponentissa elementit sisältävät sovelluskohtaista dataa ja operaatioita, jotka eivät suoraan liity käyttöliittymään. Sovellusmuunnin-komponentissa taas elementti sisältää operaatioita, jotka on tarkoitettu sovelluskohtaisen datan muuntamiseksi käyttöliittymälle. Sovellusmuunnin-komponentissa voidaan esimerkiksi yhdistää listaksi sovellusspesifistä komponentista saatuja tietoalkioita.

Interaktioelementit kuvaavat syötteitä (näppäimistö, hiiri, puhe) ja tulosteita (pikseleitä ruudulla, tuntopalautetta, ääntä). Esityskomponentilta saatu lista voidaan esimerkiksi näyttää ruudulla pudotusvalikkona tai listarakenteena.

Esityselementit ovat virtuaalisia interaktioelementtejä, jotka kuvaavat käyttäjälle näytettävää dataa ja käyttäjän laukaisemia tapahtumia. Malli ei ota kantaa esityksen mediamuotoihin eikä tapahtumien luontiin. Esimerkiksi sovellusmuunnin-komponentilta saatu lista voi olla esityselementtinä 'yksisarakkeinen, otsikoitu ja lajiteltu lista, josta voi valita yhden rivin kerrallaan'.

Siinä missä edellä kuvattu Arch malli on suunniteltu minimoimaan tulevien muutosten vaikutus eri komponentteihin, yleistää Slinky metamalli Arch mallia siten, että eri komponenttien tärkeys ja sisältämä toiminnallisuus voi vaihdella sovelluksesta toiseen. Joissain sovelluksissa on paljon toiminnallisuutta vuorovaikutuskomponentissa, kun taas sovellusspesifi komponentti voi olla hyvinkin pieni (esimerkiksi yksinkertainen lista, jota voidaan muokata eri tavoilla). Toisissa sovelluksissa voi sovellusspesifi komponentti olla sovelluksen isoin komponentti ja dialogikomponentti lähes olematon. Painopiste kaaressa voi siis liikkua eri komponenttien välillä.

4.2. PAC

PAC-suunnittelumalli [Coutaz, 1987; Buschmann *et al.*, 2000] tulee sanoista Presentation, Abstraction, Control. Malli pohjaa agentteihin, jotka tässä tarkoituksessa ovat yksittäisiä selkeän toiminnallisuuden omaavia komponentteja. Ne eivät *välttämättä* ole itsenäisiä ja omassa säikeessä ajettavia. Nykytermistöllä PAC-agentit voidaan ehkä mieltää olioiksi ennemmin kuin agenteiksi sanan varsinaisessa merkityksessä. Suunnittelumalli eriyttää tehokkaasti toiminnallisuuden pienempiin erillisiin yksiköihin ja mahdollistaa järjestelmän joustavan muuttamisen ja uuden toiminnallisuuden lisäämisen.

Malli erottaa käyttöliittymäosan, toiminnallisen ytimen ja kommunikaatio-osan toisistaan. Käyttöliittymäosa (Presentation) toteuttaa agentin käyttäjälle näkyvän osan. Toiminnallinen ydin (Abstraction) vastaa agentin sisäisestä tietomallista ja tähän kohdistuvista operaatioista. Kommunikaatio-osa (Control) yhdistää käyttöliittymäosan ja toiminnallisen ytimen ja mahdollistaa kommunikaation toisten agenttien kanssa.

Yksittäinen PAC-agentti on vain pieni osa varsinaista PAC-mallia. PAC-mallin mukaan agentit tulee järjestää hierarkisiin suhteisiin toisiinsa nähden. Hierakiassa on yksi korkeimman tason PAC-agentti, useita alimman tason agentteja ja välitason agentteja. Ylimmän tason agentti toteuttaa toiminnallisen ytimen tai on yhteydessä siihen. Ylimmän tason agentti myös sisältää ne käyttöliittymän osat, kuten valikon tai valintaikkunat, joita olisi vaikea jäsentää jonkin alemman tason agentin vastuulle.

Ylimmän tason agentin kommunikaatio-osa mahdollistaa alemman tason agenttien pääsyn jaettuun tietoon. Alemmalta tasolta tulevat palvelupyynnöt ohjataan joko agentin toiminnalliselle ytimelle tai käyttöliittymäosalle. Kommunikaatio-osa vastaa myös käyttäjän toimien oikeellisuuden tarkistamisesta ja voi myös sisältää tehtyjen toimien historian ja peruuta-kumoa toiminnallisuuden. Lisäksi kommunikaatio-osa koordinoi alemman tason agenttien toimintoja ja pitää agenttien välisistä yhteyksistä kirjaa.

Alimman tason agentit taas edustavat muista riippumattomia komponentteja, joita käyttäjä voi useimmissa tapauksissa manipuloida. Esimerkiksi erilaiset kaaviot, talukointinäkymät ym. vuorovaikutteiset elementit ovat alimman tason agentteja. Agentti tarjoaa ja toteuttaa kaikki ne tavat millä käyttäjä voi manipuloida agenttia. Toteutus luonnollisesti tapahtuu käyttöliittymäosassa. Alimman tason agentteina voi olla myös käyttöjärjestelmän palveluita käyttäviä agentteja sekä agentteja, jotka mahdollistavat yhteydet toisiin PAC-hierarkioihin tai ulkopuolisiin palveluihin.

Välitason agentit vuorostaan toimivat alemman ja ylemmän tason agenttien välisen liikenteen välittäjänä. Ne myös toimivat alemman tason agenttien välisten suhteiden ja riippuvuuksien kuvaajina ja vastaavat niiden välisestä yhdenmukaisuudesta.

Otetaan esimerkiksi sovellus, jossa käsitellään tietokannassa sijaitsevaa demografista tietoa. Käyttäjällä on useita näkymiä tietoon. Käyttäjä voi esimerkiksi tarkastella tietoa pylväs- ja piirakkadiagrammien avulla. Käyttäjä voi myös muokata tietokannan tietoja talulukkomuodossa. Alimman tason agentteja

on kolme. Taulukkoagentti, pylväs- ja piirakkadiagrammi. Välitason agentteja on vain yksi: näkymäkoordinaattori, jonka alla diagrammit ovat (kuva 2).

Kuva 2: PAC-agenttien hierarkia

Esimerkin ylimmän tason agentin toiminnallinen ydin vastaa yhteydestä tietokantaan ja tarjoaa operaatiot tietokannan käsittelemiseen ja tiedon eheyden ylläpitämiseen. Kommunikaatio-osa järjestää alemman tason agenttien välistä liikennöintiä ja yhteistyötä. Jos käyttäjä muuttaa tietoja taulukkoagentissa, tieto muutoksesta menee ylimmän tason agentille, joka päivittää tietokannan tiedot ja lähettää näkymäkoordinaattorille viestin muutoksista. Viesti etenee luonnollisesti diagrammeille saakka, jotka päivittävät näkymänsä asianmukaisella tavalla. Ylimmän tason agentilla ei ole käyttöliittymäosiota.

Alimman tason agenttien toiminnallinen ydin tallentaa demografista dataa käyttötarkoitukseen soveltuvaan muotoon ja pitää yllä agenttikohtaista tietoa datasta (esim. esitysjärjestys). Käyttöliittymäosion vastuuna on luoda tallennetusta datasta havaittava representaatio (ääni, kaavio, kuva) ja toteuttaa kaikki mahdolliset datan manipulaatiomenetelmät. Kommunikaatio-osa mahdollistaa agentin yhteyden ylemmän tason agenttiin (diagrammin tapauksessa näkymäkoordinaattoriin ja taulukkoagentin tapauksessa ylimmän tason agenttiin). Myös agentin sisäiseen malliin kohdistuvat muutokset kulkevat kommunikaatio-osan kautta.

Esimerkkimme välitason agentin, näkymäkoordinaatorin, käyttöliittymäosa voisi tarjota käyttäjälle kontrollipanelin, mistä käyttäjä voi avata tai sulkea erilaisia demografista tietoa näyttäviä diagrammeja. Toiminnallinen ydin

vuorostaan pitäisi yllä tietoa aktiivisista näkymistä. Kommunikaatio-osa vastaa alemman tason agenttien koordinoinnista välittäen esimerkiksi tietoa jaetun tietomallin muutoksista alemman tason agenteille. Välitason agentti välittää myös alemman tason agenttien jaettuun malliin tekemät muutokset ylemmälle tasolle. Agentin kommunikaatio-osa vastaa niiden alemman tason agenttien luonnista ja poistamisesta, jotka ovat seurausta käyttöliittymäosan kontrollipanelin käytöstä.

PAC-malli on myös helposti mukautettavissa monisäikeiseksi. Toisaalta malli saattaa lisätä järjestelmän monimutkaisuutta ja tehottomuutta, mikäli agenttien määrää ei osata kontrolloida. Agentin kommunikaatio-osan toteutus saattaa myös muodostua vaativaksi.

4.3. Arch, PAC ja PAC-Amodeus

Arch malli ei ota kantaa siihen miten dialogikomponentissa tapahtuva tehtävien järjestäminen voitaisiin tehdä. Coutaz ja Nigay esittävätkin PAC-Amodeus mallin [Nigay and Coutaz, 1993; Nigay and Coutaz, 1996], joka yhdistää PAC-mallin Arch mallin dialogikomponenttiin. PAC-Amodeus mallin lähtökohtana on multimodaalisen vuorovaikutuksen mahdollistaminen sovelluksessa. Kuten aiemmin mainittiin, multimodaalista vuorovaikutusta on monenlaista, mutta eri modalitettien yhdistäminen eli fuusio antaa käyttäjälle enemmän ilmaisuvoimaa ja lisää käyttäjän vuorovaikutusmahdollisuuksia järjestelmän kanssa. PAC-mallin upottaminen dialogikomponenttiin mahdollistaa modaliteettien yhdistämisen luontevalla tavalla.

Nigay ja Coutaz [1993] lajittelevat fuusion kolmeen eri ryhmään: leksikaalinen, semanttinen ja syntaktinen fuusio. Leksikaalinen fuusio tapahtuu laitetason rajapinnassa eli vuorovaikutuskomponentissa. Esimerkiksi shift-näppäimen alas painaminen saman aikaisesti kun hiirellä valitaan listasta useampia elementtejä voidaan laskea leksikaaliseksi fuusioksi ja se tulee käsitellä vuorovaikutuskomponentissa.

Syntaktinen ja semanttinen fuusio taas tapahtuu dialogikomponentissa. Syntaktisessa fuusiossa koostetaan syötteitä kokonaisen komennon aikaansaamiseksi. Käyttäjän osoittaessa kohtaa dokumentissa ja sanoessa 'laita kommentti' muodostaa hän yhden komennon, joka koostuu kahdesta erillisestä toiminnosta ja modaliteetista. Syntaktinen fuusio kokoaa nämä kaksi toimintoa yhdeksi järjestelmän ymmärtämäksi komennoksi. Semanttinen fuusio yhdistää komentoja uusiksi tapahtumiksi. Käyttäjä voi esimerkiksi piirtää viivaa ja

määritellä viivan väriä. Nämä komennot voivat tapahtua samanaikaisesti ja semanttisen fuusion avulla tuloksena on kaksivärinen viiva.

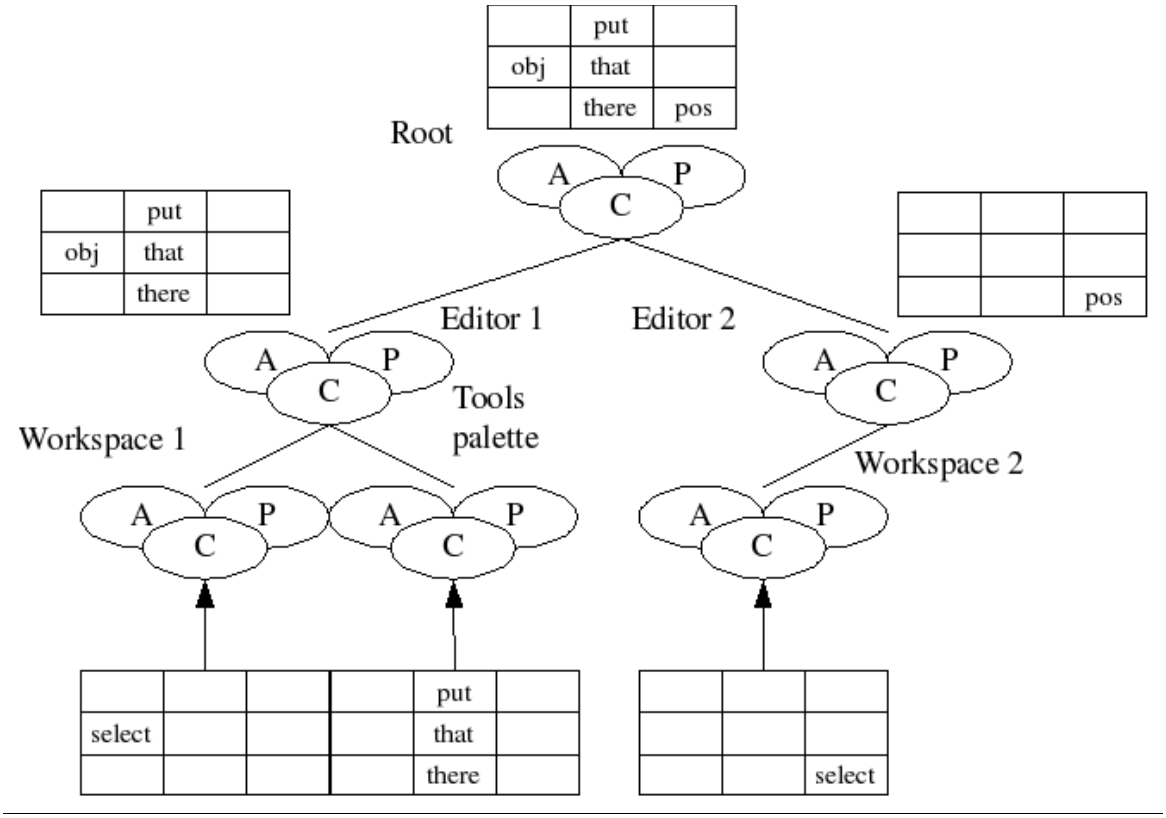
Komentojen tai toimintojen yhdistäminen vaatii yhteisen, yleistetyn, tietomallin. PAC-Amodeuksessa yhteisen tietomallin virkaa hoitaa 'sulatusuuni' (melting pot). Sulatusuuni on kaksiulotteinen rakenne, jonka x-akselilla kuvataan aikaa ja y-akselilla kuvataan käyttäjän laukaisemat tapahtumat. Sulatusuunin y-akselin tapahtumat ovat dialogikomponentin ymmärtämässä yleistetyssä rakenteellisessa muodossa [Nigay and Coutaz, 1993]. Tapahtuman sijainti y-akselila määrittää sen suhteen muihin sulatusuunin tapahtumiin. Esimerkiksi valitulla kohteella ja sen määränpäällä on omat sijaintinsa y-akselilla. Muunnos käyttäjän laukaisemista tapahtumista yleiseen muotoon tapahtuu sekä vuorovaikutus- että esityskomponentissa (kuva 6).

PAC-Amodeus mallissa dialogikomponentti sisältää useita PAC-agenttien hierarkioita. PAC-agentti on yhteydessä sovellusmuunninkomponenttiin toiminnallisen ytimensä kautta (Abstraction) ja esityskomponenttiin vuorovaikutusosansa kautta (Presentation). Yhdellä agentilla voi olla useita yhteyksiä edellä mainittuihin komponentteihin. Dialogikomponentissa tapahtuu ylimmän tason fuusio sulatusuunielementtien avulla. Fuusioon vaikuttavat sekä sulatuuunien muodostama aika-ikkuna että sulatusuunien tapahtumien välinen suhde.

Esimerkkinä grafiikkaeditoria, missä on tuki puheen tunnistukselle ja hiiren tapahtumille [Nigay and Coutaz, 1993]. Sovelluksessa on useita erillisiä piirto-alueita eli työpöytiä. Työpöytien lisäksi käyttäjällä on käytössään työkalupaletti. Kullakin työpöydällä niin kuin työkalupaletillakin on oma alimman tason PAC-agenttinsa dialogikomponentissa. Agenttihierarkiassa on myös välitason agentteja, Editoreita, jotka toimivat alemman tason agenttien koordinoijina ja niiltä tulevien sulatusuunielementtien yhdistäjinä (kuva 3).

Käyttäjä siirtää valitsemansa kohteen työpöydältä 1 työpöydälle 2 sanomalla ‘laita tuo tuonne’ ja samanaikaisesti valitsemalla kohteen ja sen määränpään. Työpöytä-1 agentti saa sulatusuunielementin esityskomponentilta ja muuntaa siinä olevan valinta toiminnon viitteeksi valittuun elementtiin. Samanaikaisesti työpöytä-2 agentti saa myös esityskomponentilta sulatusuunielementin ja muuntaa siinä olevan valinta komennon vastaamaan sijaintia työpöydällä. Työkalupaletti saa käyttäjän antaman käskyn sulatusuunielementtinä esityskomponentilta ja lähettää sen eteenpäin yhdistävälle editori-1 agentille. Editori-1 agentti yhdistää työpöytä-1 agentilta saadun elementin työkalupaletin

elementin kanssa ja lähettää yhdisteen ylemmäksi agenttihierarkiassa. Samanaikaisesti editori-2 agentti saa työpöytä-2 agentin sulatusuunielementin ja koska muuta syötettä ei ole tulossa, ohjaa se sen hierarkian ylemmälle tasolle. Juuritasolla kaksi saatua elemettiä yritetään yhdistää. Jos yhdistetty elementti noudattaa fuusion sääntöjä, muodostuu siitä kokonainen käsky, joka lähetetään toiminnalliselle ytimelle.



Kuva 3: Fuusion suorittaminen PAC-agenteilla [Nigay and Coutaz, 1993]

4.4. PAC-Amodeuksen laajennukset

PAC-Amodeus on ollut kehityksessä ja käytössä hyvin pitkän aikaa, ja täten siihen on tehty useita parannuksia ja laajennuksia. Dialogikomponentti ja erityisesti siinä tapahtuva fuusio on keskeisessä osassa Nigayn ja Coutazin tekemässä laajennuksessa [1995]. He lisäävät kontekstin fuusioon vaikuttavien tekijöiden joukkoon. Tämän lisäksi he jakavat sulatusuuneihin kohdistuvat fuusiot kolmeen eri luokkaan: mikrotemporaalinen, makrotemporaalinen ja kontekstuaalinen fuusio.

Mikrotemporaalinen fuusio, jota yritetään ensimmäiseksi, yrittää sulatusuunien yhdistämistä jos ne ovat ajallisesti lähellä toisiaan (sulatusuunien

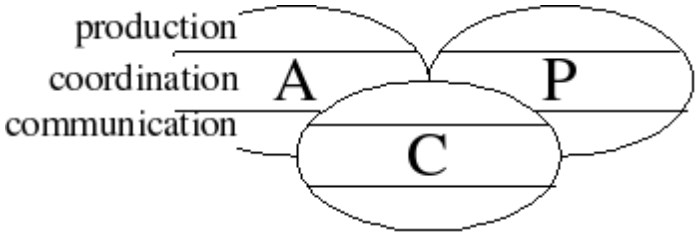
ajat limittyvät) ja jos ne ovat rakenteellisesti yhteensopivia. Makrotemporaalinen fuusio yrittää yhdistää sulatusuunit, jotka ovat muodostuneet samanaikaisesti tai peräkkäin, mutta jotka järjestelmä on käsitellyt peräkkäisessä järjestyksessä tai joita on viivästytetty resurssipulan vuoksi (prosessoriaika, muisti yms.). Fuusio tapahtuu, jos sulatusuunit ovat rakenteellisesti yhteensopivia, kuuluvat samaan ajalliseen jaksoon ja jos niiden ajat eivät limity. Kontekstuaalinen fuusio yhdistää rakenteellisesti yhteensopivat sulatusuunit ottamatta huomioon ajallisia tekijöitä.

Fuusion suorittaa dialogikomponentin osana oleva fuusiomoottori. Moottori on rakennettu niin että se suosii samanaikaisten sulatusuunien yhdistämistä ja osaa peruuttaa jo aiemmin muodostettuja fuusioita, mikäli uusi sulatusuuni mahdollistaa paremman fuusion aikaisempien sulatusuunien kanssa.

Jokaista tehtävää kohtaan on olemassa agenttihierarkia. Kunkin hierarkian juuri-agentti pitää yllä tietoa sulatusuuneista ja niistä PAC-agenteista, jotka ovat kiinnostuneet niistä. Tämä mahdollistaa samanaikaisten toisistaan riippumattomien samanaikaisten tapahtumien käsittelyn.

Pohjimmiltaan fuusio tapahtuu samalla tavalla kuin edellä mainitussa esimerkissä. Erona on valmiin komennon käsittely. Edellisessä esimerkissä käyttäjä valitsi kohteen hiirellä, sanoi 'laita tuo tuonne' ja osoitti määränpäätä hiirellä. Uusi fuusio luo yhteen sopivista sulatusuuneista uuden PAC-agentin. Agentin ydinosa sisältää komennon sisällön ja voi päivittää näkymää esitysosionsa kautta. Kun komento on valmis agentti lähettää komennon sovellismuunninkomponentille.

PAC* [Calvary *et al.,* 1997] vuorostaan yhdistää PAC-Amodeuksen ja Salberin apilamallin (clover). Apilamalli löytää ryhmätyöohjelmistoista kolme sovelluskohtaista toimintoa: tuotanto, koordinaatio ja kommunikaatio [Salber, 1995]. PAC*:n agentin osat jakaantuvat PAC-jaon lisäksi näihin kolmeen toimintoon (kuva 4). Esimerkkinä Calvary tarjoaa jaettua SASSE-editoria [Baecker *et al.*, 1992], missä muiden käyttäjien sijainti dokumentissa näkyy erillisessä vierityspalkissa. Vierityspalkin avulla käyttäjä voi myös avata audio-video yhteyden toiseen käyttäjään.



Kuva 4: PAC*-agentti [Calvary *et al.,* 1997]

PAC-Amodeus malliin mukautettuna agentin toiminnallisen ytimen (Abstraction) tuotanto-osa sisältäisi tiedon paikallisen käyttäjän sijainnista dokumentissa. Esitysosan (Presentation) tuotanto-osa vuorostaan viittaa tavalliseen vierityspalkkiin, jonka avulla käyttäjä voi navigoida dokumentissa. Kommunikaatio-osan (Control) tuotanto-osa yhdistää esitysosan viertyspalkin ja toiminnallisen ytimen sijaintitiedon.

Agentin toiminnallisen ytimen koordinaatio-osa sisältää tiedon dokumenttia selailevista muista käyttäjistä ja näiden sijainnista dokumentissa. Esitysosan koordinaatio-osa viittaa erityiseen vierityspalkkiin, missä toisten käyttäjien sijainti on havaittavissa ja kommunikaatio-osan koordinaatio-osa yhdistää vierityspalkin ja nämä käyttäjätiedot.

Toiminnallisen ytimen kommunikaatio-osa saattaa sisältää esimerkiksi käyttöoikeustietoja tai audio-video kommunikaation laatuun vaikuttavaa tietoa. Esitysosan kommunikaatio-osa viittaa etäkäyttäjän audio-video dataan.

Apilamalli on toteutettavissa myös siten että yksittäistä agenttia ei jaetakaan apilamalliin, vaan muodostetaan agenttiryppäitä, joissa on yhteydessä saman agentin apilamallin edustajat. Eli tuotanto-osalla on oma PAC agentti jne. Laurillau ja Nigay [2002] laajentavat PAC*-mallia edelleen liittämällä se Dewanin yleiseen ryhmätyöohjelmistoarkkitehtuuriin [Dewan, 1999].

4.5. PAC-Amodeukseen perustuvat sovellukset

MATIS (Multimodal Airline Travel Information System) on lentoreittien ja aikataulujen selaamiseen tehty ohjelma, jota voi käyttää puheen, hiiren, näppäimistön ja suoran vaikuttamisen avulla [Nigay and Coutaz, 1996]. Järjestelmä tukee useamman modaliteetin käyttöä samanaikaisesti, mutta sallii myös vain yhden modaliteetin käytön. Käyttäjä siis voi antaa saman komennon esimerkiksi puheen tai hiiren avulla tai niiden yhditelmällä. Järjestelmä myös tukee päällekkäisiä tehtäviä, eli käyttäjä voi keskeyttää komennon, jota on

antamassa, antaa uuden komennon, ja palata kesken jääneen komennon antamiseen.

VoicePaint on grafiikkaeditori, johon on liitetty puheen tunnistuslaite [Nigay and Coutaz, 1993]. Piirtäessään grafiikkaelementtejä hiirellä, käyttäjä voi puheen avulla vaikuttaa grafiikkaymäristöön. Hän voi esimerkiksi vaihdella kynän, taustan tai etualan väriä, viivan paksuutta tai täyttökuviota.

NoteBook sovellus on elektroninen kirja [Nigay and Coutaz, 1993]. Käyttäjä voi lisätä, poistaa, selata ja muokata tesktipohjaisia muistiinpanoja. Sovelluksessa voi käyttää sekä hiirtä että puhetta komentojen antamiseen. Tekstin syöttö tapahtuu kirjoittamalla. Käyttäjä voi myös lisätä sivuja sivujen väliin.

CoVitesse [Laurillau and Nigay, 2002] on ryhmätyöohjelmisto, joka mahdollistaa WWW-sivujen navigoinnin yhteistyössä. Käyttäjät, jotka jakavat saman informaatiosisällön, voivat tehdä yhteistyötä tiedon etsimisessä. CoVitesse ei hyödynnä useita modaliteetteja, vaan lähinnä soveltaa edellä mainittua PAC-Amodeuksen apila-laajennusta.

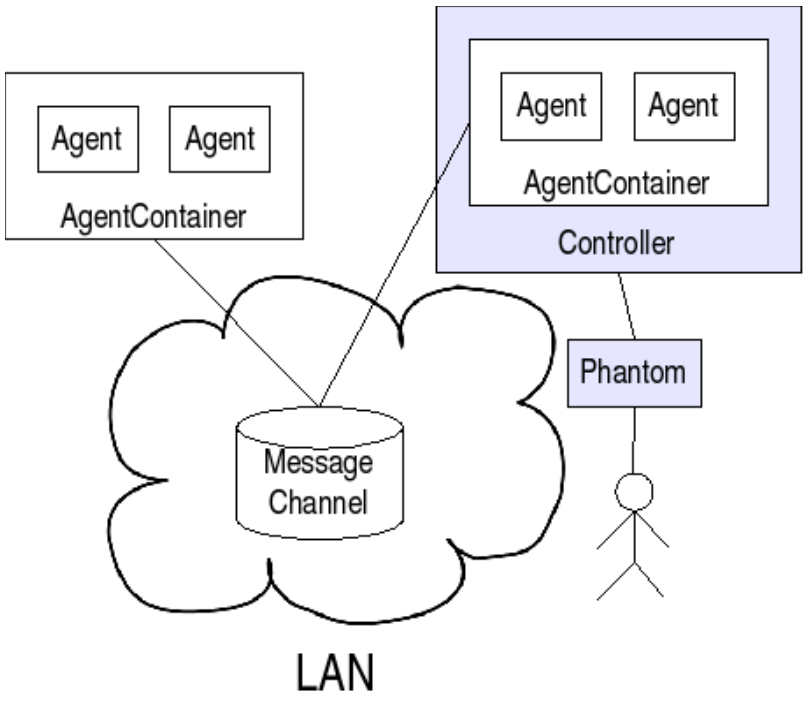
5. PROAGENTS-agenttiarkkitehtuuri

Tampereen Yliopistolla PROAGENTS-hankeen yhteydessä rakennettiin kevyt agenttialusta. Alusta suuniteltiin käytettäväksi heterogeenisessa lähiverkossa, johon liitettynä voi olla erilaisia koneita ja käyttöjärjestelmiä [Saarinen *et al.*, 2005]. Pohjimmainen tarkoitus alustan luontiin oli tarve sekä hajauttaa suoritukseltaan raskaat toimitukset lähiverkon koneiden kesken että yhdistää tuntopalautelaite PHANToM [Sensable, 2005] agenttijärjestelmään. Alusta haluttiin pitää myös helposti muokattavana ja helppotajuisena. Yksinkertaisen hajautetun agenttijärjestelmän saa rakennettua muutamassa tunnissa perehtymättä alustan rakenteeseen tarkemmin. Agenttienvälinen kommunikaatio noudattaa löysästi FIPA:n spesifikaatioita [FIPA00019; FIPA00003; FIPA00017].

Agenttialusta pyrittiin pitää mahdollisimman riippumattomana sovelluskohteista. Täten alusta itsessään ei ota kantaa agenttien välissä liikkuvaan tietoon, vuorovaikutustekniikoihin tai erilaisten modaliteettien käyttöön. Tarkoitus on että eri projekteissa käytettäisiin olemassa olevia agentteja ja mahdollisesti lisättäisiin uutta, multimodaalistakin, toiminnallisuutta agenttien muodossa.

PROAGENTS-projektissa rakennettiin opetusohjelma sokeille lapsille. Ohjelman teemoja oli aurinkokunta, Maa, Maan suhde aurinkoon ja vuodenajat, maan kerrokset ja ilmakehä. Koska pääkohderyhmä oli sokeat lapset, keskityttiin puuttuvan aistin korvaamiseen tuntopalautteen avulla. Multimodaalisuuden pääpaino sovelluksessa ei siis ollut useamman modaliteetin käyttö syötteen antamiseksi niin kuin edellä mainituissa PAC-Amodeus malliin perustuvissa sovelluksissa, vaan puuttuvan aistin korvaaminen ja useamman aistin käyttäminen palautteessa. Sovellusta voi käyttää myös näkevät lapset täysipainoisesti.

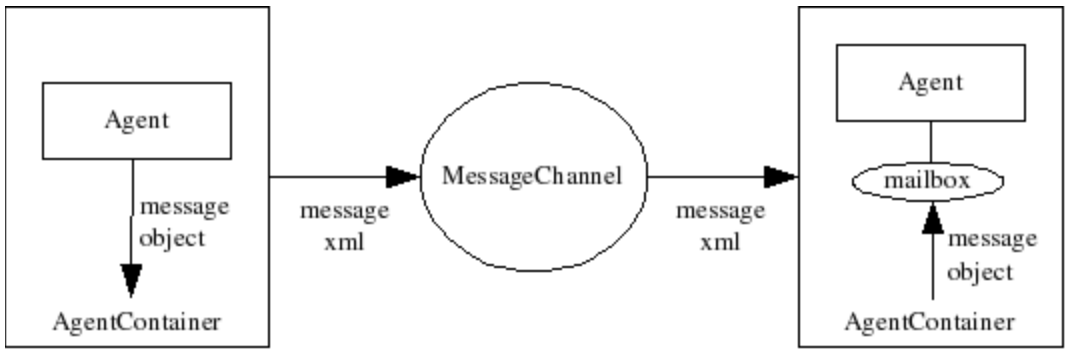
Agenttialusta voidaan hajauttaa usealle lähiverkon koneelle erillisten agenttisäiliöiden (AgentContainer) muodossa. Agentit itsessään ovat säikeitä, jotka ovat ajossa agenttisäiliön sisällä. Agenttisäiliöt ovat yhteydessä toisiinsa keskuspalvelimen, viestikanavan (MessageChannel), kautta (kuva 5). Sekä viestikanavalla että säiliöillä on yksinkertainen konfigurointitiedosto, jonka muokkaaminen riittää niiden toimintakuntoon saattamiseksi.



Kuva 5: Yleiskuva agenttialustasta

Viestintä tapahtuu pääosin keskitetysti viestikanavan kautta, ellei viestin vastaanottaja ole samassa säiliössä. Agentti luo viesti-olion ja antaa viestin agenttisäiliölle lähetettäväksi. Agenttisäiliö muuttaa viestin XML-muotoon ja lähettää sen viestikanavalle. Viestikanava lähettää viestin edelleen säiliölle, jossa

vastaanottaja sijaitsee ja lopulta vastaanottava säiliö muuntaa viestin takaisin viesti-olioksi ja toimittaa sen vastaanottavan agentin postilaatikkoon (kuva 6). Saman säiliön sisällä viesti siirtyy viesti-oliona. Yleinen XML-viestimuoto mahdollistaa eri ohjelmointikielillä tehtyjen ja myös eri käyttöjärjestelmissä ajossa olevien agenttisäiliöiden yhteyden samaan viestikanavaan ja täten myös toisiinsa. Tällä hetkellä agenttisäiliöstä on sekä C++- että Python-toteutukset.



Kuva 6: Viestin kulku agenttijärjestelmässä

Monimutkaisemmissa järjestelmisssä tulee tarpeelliseksi antaa agentille mahdollisuus omien tietoliikenneyhteyksien muodostamiseen sillä viestikanavasta muodostuu nopeasti järjestelmän pullonkaula. Arkkitehtuuri mahdollistaakin suoran yhteyden toiseen agenttiin sekä TCP/IP- että UDP-tietoliikenneprotokollien avulla. Arkkitehtuuri tosin ei ota kantaa siihen miten yhteys saadaan aikaiseksi. Agentti esimerkiksi voi aloittaa dialogin toisen agentin kanssa viestikanavan avulla ja sopimukseen päästyään agentit voisivat keskustella suoraan keskenään.

5.1. MessageChannel

Viestikanava (MessageChannel) vastaa hakemistopalveluista. Käynnistyessään sekä agenttisäiliöt että agentit kirjautuvat viestikanavalle. Viestikanava vastaa uusien säiliöiden nimeämisestä ja säiliö saakin ainutlaatuisen nimensä kirjautumisprosessin päätteeksi. Agentti vuorostaan lähettää käynnistyessään viestikanavalle tiedot tarjoamistaan palveluista. Sekä agentit että säiliöt myös poiskirjautuvat lopettaessaan.

Viestikanava pitää kirjaa olemassa olevista säiliöistä, säiliöiden tukemista tietoliikenneprotokollista, agenteista ja agenttien tarjoamista palveluista. Yleisin viestikanavan tarjoama toiminnallisuus on tietyn palvelun tarjoavien agenttien kysyminen. Eli agentti voi kysyä viestikanavalta, mitkä agentit tarjoavat esimerkiksi lokipalvelua. Vastauksena agentti saa listan palvelun tarjoavista

agenteista, jota se voi käyttää jatkossa esimerkiksi kommunikaatiotapahtumia käynnistäessään.

Sen lisäksi että agentti voi lähettää viestejä toisille agenteille, voi agentti myös lähettää viestin kaikille tietyn palvelun tarjoaville agenteille asettamalla palvelun nimen viestin vastaaottajaksi. Viesti voidaan myös osoittaa kaikille agenteille asettamalla viestin vastaaottajaksi 'broadcast'.

5.2. AgentContainer

Agenttisäiliö (AgentContainer) toimii agenttien ajo-alustana ja tarjoaa niille yksinkertaiset postipalvelut. Postipalveluihin kuuluu mm. agentin omakohtainen postilaatikko ja yksinkertaiset viestin lähetysoperaatiot. Säiliö myös kontrolloi agentin elinkaarta mahdollistaen agenttien luonnin, tuhoamisen, pysäyttämisen ja pysäytyksestä jatkamisen. Agenttisäiliö antaa agentille nimen agentin luontihetkellä.

Agenttisäiliöllä on konfiguraatiotiedosto, missä määritellään viestikanavan yhteystiedot ja käynnistettävien agenttien konfiguraatiotiedostot. Agenttisäiliö voi luoda agentteja myös ajon aikana.

5.3. Agentit

Käynnistyksen yhdeydessä agentille voidaan antaa parametreja. Parametrit määritellään agentin konfiguraatiotiedostossa, josta löytyy myös tieto käynnistettävästä agentista. Agenttia ajetaan erillisessä säikeessä. Agentti voi myös käynnistää uusia säikeitä, mutta tällöin agentin tulee itse vastata niiden tuhoamisesta.

Järjestelmän tarjoama agenttien yläluokka tarjoaa muutamia valmiita toimintoja. Tuhoutuessaan agentti automaattisesti kirjautuu pois viestikanavasta ja ilmoittaa tuhoutumisestaan säiliölleen. Tämän lisäksi agentilla on rajapinta, jonka avulla sille voidaan helposti määritellä agentin tarjoamat palvelut ja ne voidaan rekisteröidä viestikanavaan. Palvelu on käytännössä vain merkkijono, jonka avulla agentti voi mainostaa ominaisuuksiaan (mm. tarjoamiaansa palveluja) muille.

5.4. Kontrolli ja PHANToM

Kontrolli (Control) toimii pääohjelmana, joka myös mahdollistaa agenttien yhteyden sovellukseen sekä käyttäjään. Kontrolli on rakennettu käytettäväksi kolmiuloitteisessa tuntopalauteympäristössä, jossa käytetään PHANToM

tuntopalautelaitetta [Sensable, 2005]. Tässä ympäristössä sovellusohjelma voidaan nähdä joukkona virtuaalimaailmoja, joista on pääsy toisiin virtuaalimaaimoihin. Jokainen virtuaalimaailma määritellään erillisessä VRML (Virtual Reality Markup Language) määrittelytiedostossa [VRML, 1997]. Virtuaalimaailman VRML-määrittelytiedosto voi sisältää myös Python-skriptikieltä. Virtuaalimaailma ladataan osaksi jo olemassaolevaa näkymägraafia, jossa on valmiina jaettuja elementtejä, jotka mahdollistavat lähinnä kahdensuuntaisen liikenteen kontrollin ja Python-skriptikielen välillä. On siis mahdollista vaikuttaa virtuaalimaailman skriptattuun toiminnallisuuteen ja skripteistä voidaan lähettää esimerkiksi viestejä agenteille.

Kontrolli tarjoaa rajapinnan, jonka avulla agentit voivat esimerkiksi olla vuorovaikutuksessa käyttäjän kanssa. Tällä hetkellä rajapinta mahdollistaa käyttäjän informoinnin jostain asiasta ja yksinkertaisen kyllä-ei kysymyksen esittämisen käyttäjälle. Kummassakin tapauksessa PHANToM-laitteen kynää tärisytetään ja soitetaan pieni merkkiääni interaktioyrityksen merkiksi. Käyttäjä voi joko hyväksyä tai evätä tulevan viestin. Jos käyttäjä ei reagoi viestiin, tulkitaan se tietyn ajan kuluessa kieltäytymiseksi. PROAGENTS-projektin viesteissä käytetään vain äänipalautetta, eli viestit soitetaan käyttäjälle hänen niin halutessa.

Suoranaista navigointia helpottavaa äänipalautetta ei ole toteutettu PROAGENTS-projektissa, mutta rajapinta mahdollistaa äänien suoran soiton eri äänikanavilla. Esimerkiksi opasteäänet ja huomioäänet soitetaan tämän rajapinnan kautta.

Kontrollin rajapinnan avulla agentit voisivat myös antaa haptista palautetta käyttäjälle. PROAGENTS-projektissa toteutettiin vain kynän liikuttaminen tiettyyn kordinaattiin, mutta olisi myös mahdollista vaikuttaa virtuaalimaailmaan suoraan tai esimerkiksi toteuttaa dynaamista haptista palautetta.

5.4.1. PROAGENTS-Minimaailmat ja agentit

Kuten edellä mainittiin sovelluksessa on viisi eri teemaa, jotka jokainen muodostaa oman virtuaalimaailmansa. Kutsumme jokaista virtuaalimaailmaa minimaailmaksi. Minimaailmat on suunniteltu siten että niissä on mahdollisimman helppo navigoida ilman näköaistia. Puuttuvaa aistia korvaamaan on otettu tunto- ja äänipalaute. Suurin osa tuntopalauteesta on

staattisesti kuvattu minimaailmojen määrittelyn yhteydessä, kun taas äänipalaute on osittain staattista ja osittain dynaamista.

Minimaailmojen suunnittelu pohjaa mm. Patomäen *et al.* [2004] ja Sjöströmin [2001] löytöihin. Pyrimme esimerkiksi pitämään virtuaalimaailmojen objektit mahdollisimmaan yksinkertaisina niin kuin Patomäki ehdottaa. Sjöström vuorostaan ehdottaa, että sokeille tehdyissä virtuaalimaailmassa tulisi olla kiintopisteitä, jotka helpottavat käyttäjän navigointia. Me suunnittelimme sovelluksen niin että käyttäjä joutuu kulkemaan kuudennen minimaailman, keskusaseman, kautta mennessään minimaailmasta toiseen. Käyttäjä on aina korkeintaan yhden askeleen päässä keskusasemasta. Myös PHANToM laitteen kynäosa siirretään aina samaan kohtaan minimaailmasta toiseen siirryttäessä. Tämä estää ikävien tuntopalautepiikkien tapahtumisen ja antaa käyttäjälle aina saman lähtöpisteen minimaailman tutkimiseen. Sovelluksen käyttö aloitetaan keskusasemalta, mistä on siis pääsy kaikkiin muihin minimaailmoihin.

Minimaailmoissa on tuntopalautetta käytetty asioiden havainnoillistamisen lisäksi myös käytön helpottamiseen. Esimerkiksi Maa-minimaailmassa haptiseen laitteeseen on liitetty jousivoima, joka vetää kynää kevyesti kohti Maan keskipistettä. Tämä havainnoillistaa painovoimaa ja auttaa käyttäjää löytämään mielenkiintoiset asiat minimaailmasta. Aurinkokunta minimmailmassa vuorostaan käyttöalueen syvyttä on rajattu tasolla, jonka pinnalla tutkittavat planeetat ja kiertoradat ovat. Kiertoradat ovat uurteita tasolla ja planeetat pallomaisia, lievästi magneettisia, elementtejä.

Minimaailmat ovat vain osa PROAGENTS-sovellusta. Suurin osa sovelluksen toiminnallisuudesta sijaitsee agenteissa, jotka ovat yhteydessä kontrolliin viestikanavan kautta. Tärkeimmät PROAGENTS-sovelluksessa rakennetut agentit ovat: filtteriagentit, tietokanta-agentti, sääntökone-agentti ja pedagoginen agentti.

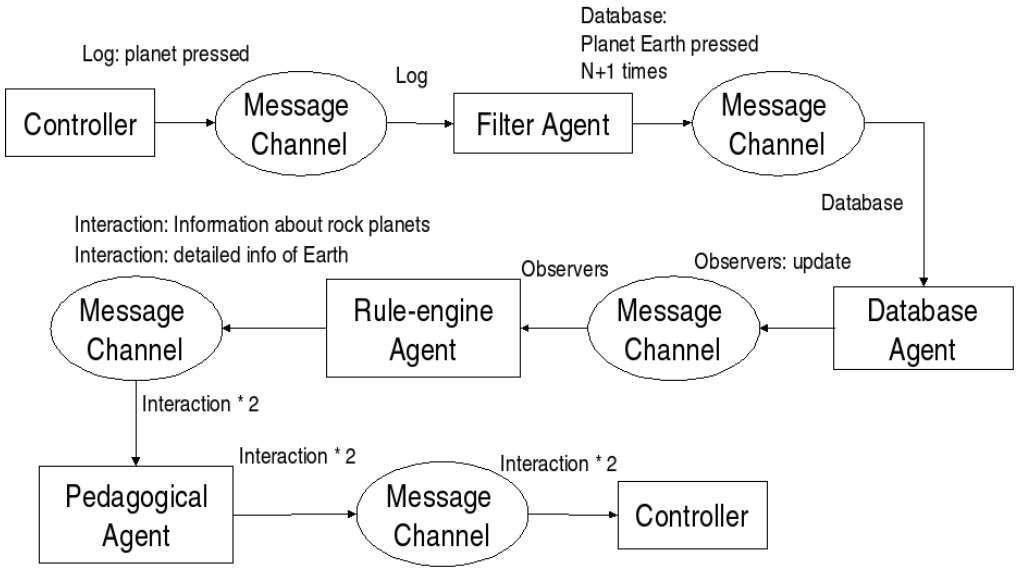
Filtteriagentit ovat agentteja, jotka sisältävät yhden tai useamman filtterin. Filtteri on toiminnaltaan hyvin rajattu ja pieni komponentti, joka reagoi vain tiettyihin asioihin. Filtteri voi muuttaa saamaansa dataa ja lähettää sitä eteenpäin toisille (filtteri) agenteille. Yhden agentin sisällä olevat filtterit voidaan ketjuttaa siten että kun yksi filtteri regoi tulevaan dataan, jätetään muut filtterit väliin. Filttereitä käytetään esimerkiksi järjestelmässä kulkevien lokitietojen tarkkailuun ja tietokanta-agentin tietojen päivittämiseen. Filtterit soveltuvat myös erinomaisesti eri syötelaitteiden integroimiseen järjestelmään.

Tietokanta-agentti vuorostaan pitää yllä sovelluksen jaettua tietoa. Muut agentit voivat kysyä ja muuttaa tietokannan tietoalkioita. Agentti voi myös ilmoittaa olevansa kiinostunut tietystä tietoalkiosta ja sille lähetetään välittömästi tieto tietoalkion muutoksista.

Sääntökone-agentti vastaa sovelluksen dynaamisesta toiminnallisuudesta ja lähes kaikesta interaktiosta käyttäjän kanssa. Sääntökone-agentti on tiukasti yhteydessä tietokanta-agenttiin ja tarkkaileekin kaikkia sen tietoalkioita. Sääntökone-agentti sisältää CLIPS-sääntökoneen [CLIPS, 2005], jonka työmuistissa tietoalkiot ovat. Sääntökone koostuu sarjasta sääntöjä. Sääntö koostuu ehto-osasta ja toiminta osasta. Ehto-osan toetuduttua sääntö voi laueta, jos se on kaikista laukeavista säännöistä sopivin. Säännöt kuvaavat lähes kaiken käyttäjälle näkyvän agenttitoiminnallisuuden, eli laukeavat säännöt voivat saada aikaiseksi yhteydenoton käyttäjään ja informaation tarjoamisen käyttäjälle.

Kaikki agenttien käyttäjälle lähettämät interaktioviestit kulkevat pedagogisen agentin kautta. Pedagoginen agentti vastaa interaktion mielekkyyden tarkistamisesta ja kontekstin säilymisestä. PROAGENTS-projektissa pedagoginen agentti on hyvin ohut ja periaatteessa lähettää vain kaikki viestit edelleen kontrollille, mutta agentilla voisi olla esimerkiksi kontekstuaalista tietoa tutkinnan alla olevasta aiheesta ja vaikkapa erilaisia opetussuunnitelmia. Jos esimerkiksi käyttäjän katsotaan juuri oppineen asia A, niin pedagoginen agentti voisi tutkia, mitkä asiat olisi hyvä tietää ennen asian A oppimista ja tarjota niitä käyttäjän tutkittavaksi. Toisaalta agentti voisi myös tarjota sellaisia asioita, jotka ovat riippuvaisia juuri opitusta asiasta. Näin olisi mahdollista säilyttää opittavien asioiden loogiset yhteydet ja konteksti.

Mitä sitten todella tapahtuu PROAGENTS-ympäristössä ajon aikana? Otetaan esimerkki käyttötilanteesta, missä käyttäjä on tutkimassa aurinkokuntaa. Hän on löytänyt Venuksen ja Marsin ja on kuullut niistä tietoa. Tämän lisäksi käyttäjä on painanut kumpaakin planeettaa ja näin saanut planeetoista lisätietoa. Käyttäjä löytää maan kiertoradan ja saapuu sitä seuraten Maan kohdalle. Käyttäjälle soitetaan lyhyt kuvaus Maasta ja tämän jälkeen käyttäjä painaa Maata haluten kuulla planeetasta lisätietoja. Kuvassa 7 on kaavio viestinnästä agenttialustan sisällä.



Kuva 7: Agenttien välinen vuorovaikutus

Ensinnäkin viesti siitä että käyttäjä on painanut Maa-planettaa lähtee kaikille lokiagenteille. Järjestelmässä on filtteriagentti, jonka filtterit tutkailevat lokiviestejä. Agentti etsii lokiviesteistä tietokannassa olevaan jaettuun tietoon vaikuttavia tapahtumia. Saadessaan viestin filtteri päättää, että tietokanta-agentin tietoa 'maa planeettaa painettu *n* kertaa' tulee päivittää ja näin lähettääkin tietokanta-agentille päivityspyynnön. Tietokanta-agentti saa päivityspyynnön, päivittää tiedon tietokantaan ja lähettää kaikille tiedon tarkkailijoille viestin muutoksesta. Kuten aiemmin todettiin, sääntökone-agentti kuuntelee kaikkea jaettua tietoa ja näin saakin tiedon muutoksesta. Muuttuneiden tietojen ansiosta sääntökoneessa laukeaa kaksi sääntöä. Ensinnä laukeaa interaktiotapahtuma planeetan lisätietojen soittamisesta. Toiseksi laukeaa sääntö interaktiotapahtumasta, missä käyttäjälle kerrotaan lisää kiviplaneetoista. Molemmat interaktioviestit lähetetään pedagogiselle agentille, joka lähettää viestit edelleen kontrollille.

Kontrolli ei ole välttämätön agenttijärjestelmän toiminnan kannalta. Agenttialustaa laajennetaan yleisesti lisäämällä uusi toiminnallisuus uusien agenttien muodossa tai vanhojen agenttien laajennuksena. PROAGENTS-järjestelmää voidaan myös laajentaa lisäämällä ja muuttamalla kontrollin rajapintoja. Esimerkiksi uusi laite voidaan lisätä uutena agenttina tai integroituna kontrolliin, joka tarjoaa rajapinnan laitteeseen. Uuden agentin lisääminen on suositeltavaa, sillä silloin sitä voidaan käyttää muissakin kuin PROAGENTS-projetin sovelluksissa. Agentit lisätään alustaan liitännäisinä (plug-in) eli

jaettuina kirjastoina. Näin agenttien tekijä voi keskittyä pelkästään agenttien tekemiseen.

6. PROAGENTS PAC-Amodeus mallina

Siinä missä PAC-Amodeus on malli interaktiivisten multimodaalisten sovellusten kehittämiseen, on kehittämämme agenttialusta konkreettinen agenttialusta, joka määrittää vain alimman tason toiminnallisuuden ja mahdollistaa agenttien helpon käytön. Alustalla voidaan yhtä hyvin toteuttaa niin pikaviestimiä, jaettuja kalentereita, mediakeskuksia kuin multimodaalisia opetusohjelmiakin. Täten PAC-Amodeuksen ja agenttiarkkitehtuurin suora vertaaminen onkin järjetöntä. Seuraavaksi esitetään muutamia ratkaisuja siihen miten agenttialustan avulla voisi rakentaa PAC-Amodeusta vastaavan järjestelmän.

Kuten muistamme koostuu PAC-Amodeus sekä Arch mallista että PAC-agenteista. Arch mallin vuorovaikutuskomponentti voidaan nähdä erillisenä agenttina, joka on liitetty tarkkailemaan tiettyä syötelaitetta ja mahdollisesti myös käyttämään sitä palautteeseen. Graafinen käyttöliittymäkin on vuorovaikutusagentti: sitä käytetään vain palautteeseen. Vuorovaikutusagentti voi olla joko tavallinen agentti tai filtteri. Jokainen vuorovaikutusagentti lähettää tiedon syöttestä esitysagentille tai -agenteille, jotka vastaavat esityskomponenttia Arch-mallissa. Esitysagentti tietää siihen yhteydessä olevat vuorovaikutusagentit ja näiden ominaisuudet. Tätä tietoa apuna käyttäen esitysagentti voi valita mieleisensä vuorovaikutusagentit palautteen antamiseen. Esitysagentti yleistää vuorovaikutuskomponenteilta tulevia tapahtumia ja lähettää ne edelleen dialogikomponentin toiminnallisuudesta vastaaville agenteille.

Dialogikomponentin toiminnallisuuden voisi toteuttaa esimerkiksi filttereiden avulla niin että kukin alimman tason filtteri kuuntelee esitysagenteilta tulevia sulatusuunien vastineita ja osaa lähettää käsittelemiään viestejä eteenpäin toiselle filtterille. Näin saadaan aikaiseksi filttereiden ketjuja, jotka vastaavat PAC-hierarkioita. Ketjun viimeisin filtteri toimittaa valmiin komennon adapteriagentille (sovellusmuunninkomponentille), joka muuntaa viestin sovelluskohtaiseksi komennoksi ja lähettää sen sovelluksen toiminnallisesta ytimestä vastaavalle agentille tai agenteille.

Yksittäisen PAC-*agentin* toteuttamiseen on myös useita erilaisia mahdollisuuksia:

- PAC-jako voidaan yhdistää samaan agentin toteutukseen.
- Agentti voi myös sisältää oman olion toiminnalliselle ytimelle ja käyttöliittymäosalle. Kommunikaatio-osaa vastaa periaatteessa agentin (säikeen) pääsuoritussilmukka joka vastaanottaa ja lähettää viestejä ja koordinoi muiden osien toimintaa. Käyttöliittymäosaa ja toiminnallista ydintä vastaavat oliot voidaan myös tehdä säikeiksi, jolloin ne ovat jatkuvasti suorituksessa.
- Jokainen osa voidaan myös toteuttaa erillisinä agentteina, jolloin kommuni-kaatioagentti koordinoi käyttöliittymäagenttia ja ydinagenttia. Kolmikko olisi muihin vastaaviin agentteihin yhteydessä kommunikaatioagentin kautta.

Viestikanavaa voidaan toisaalta laajentaa siten, että siihen voidaan esimerkiksi rakentaa Blackboard-tyylinen toiminnallisuus. Tällöin jokainen PAC-agentti hakisi sieltä juuri saapuneita sulatusuunien vastineita. Tai sitten PAC-Agentit ilmoittavat kaikki tarjoavansa samaa palvelua ja sulatusuunien vastineet lähetetään kaikille palvelun tarjoaville agenteille.

Filtterit voidaan korvata tavallisilla agenteilla ongelmitta. Filttereiden käyttö on ainoastaan mielekästä silloin, kun filttereiden toiminta on niin pientä ja tarkkaan rajattua että niiden toteuttaminen agentteina olisi vaivaan nähden ylimitoitettua.

Järjestelmää voisi laajentaa lisäämällä siihen käyttäjäprofiiliagentin, joka pitäisi kirjaa esimerkiksi käyttäjän ominaisuuksista, taitotasosta ja mieltymyksistä. Vuorovaikutusagentti voisi käyttää tätä tietoa hyväkseen valitessaan palautemo-daliteetteja.

7. Yhteenveto

PAC-Amodeus on voimakas malli, joka antaa hyvän lähtökohdan multimodaalisten agenttipojaisten sovellusten sunnittelulle. Arch mallin viisiosainen rakenne autaa suunnitelijaa jakamaan vastuualueita agenttien kesken ja näin agentit pysyvät selkeinä ja pienikokoisina. PAC-malli vuorostaan saattaa auttaa suunnittelijaa monimutkaisien jäerjestelmien agenttien vuorovaikutusten suunnittelussa. Toiminnallisuus voidaan jakaa agenttihierarkioihin, jotka ovat toisistaan riippumattomia ja vastaavat yhdestä järjestelmän osa-kokonaisuudesta.

PAC-Amodeus ei määrittele järjestelmässä liikkuvan datan ominaisuuksia muuten kuin sulatusuunin kohdalla. Malli ei myöskään ota kantaa tiedon siirtämiseen tai eri komponenttien sijaintiin ja niiden välisiin suhteisiin (samalla koneella, lähiverkon koneilla, sama prosessi, erilliset prosessit). Sovelluksen tekijän on joko itse rakennettava vastaavat toiminnallisuudet tai hänen on käytettävä jotain valmista järjestelmää; Esimerkiksi tässä artikkelissa esiteltyä agenttiarkkitehtuuria. Agenttiarkkitehtuuri soveltuu hyvin PAC-Amodeuksen mukaisen systeemin rakentamiseen ja tarjoaa useita lähestymistapoja mallin toiminnallisuuden saavuttamiseksi. Agenttipohjaisuus tarjoaa mahdollisuuden järjestelmän helppoon laajentamiseen ja muokkaamiseen.

Agenttiarkkitehtuuri on toisaalta niin yleinen ja avoin että se jättää sovelluksen tekijälle paljon toteutettavaa. Arkkitehtuuri ei esimerkiksi määrittele kuin pienen joukon protokollia, jotka lähinnä vastaavat alustan sisäisestä toiminnasta. Suunnittelijan onkin siis itse suunniteltava agenttien väliset kommunikaatioprotokollat ja vuorovaikutussuhteet. Onneksi suunnittelijan avuksi on laadittu erilaisia standardeja agenttien välisen kommunikaation toteuttamiseen. Esimerkiksi FIPA on suunnitellut moniagenttijärjestelmiin sopivia kommunikaatioprotokollia [FIPA00003].

Suunnittelijan on huomioitava myös erilaiset synkronointiin liittyvät asiat, kuten fission aikaansaama useamman modaliteetin käyttö palauutteeseen. Jos käytetään esimerkiksi ääntä ja kuvaa saman aikaisesti, olisi hyvä että ne olisivat synkronoituja keskenään.

PAC-Amodeus ei toki ole ainoa tapa toteuttaa multimodaalisia järjestelmiä, mutta se tarjoaa hyvin testatun ja pitkään kehitetyn työkalun, joka itsessään pohjaa tukevasti hyviksi todettuihin malleihin. Malli kannattanee ottaa käyttöön vasta isommissa ja monimutkaisemissa järjestelmissä, sillä sen toteuttaminen on kuitenkin suhteellisen vaativaa ja aikaa vievää. Pienemmissä multimodaalisissa sovelluksissa pärjää yksinkertaisemmillakin malleilla, kuten tässäkin artikkelissa on esitetty.

Viitteet

[Bass *et al., 1992*] Len Bass, Ross Faneuf, Reed Little, Niels Mayer, Scott Reed, Robert Seacord, Martha R. Szczur, A Metamodel for the Runtime Architecture of an Interactive System, SIGCHI Bulletin, 24, 1, 32-37.

[Bradshaw, 1997] Jeffrey M. Bradshaw, An introduction to software agents. In: Jeffrey M. Bradshaw (eds.), *Software Agents*, AAAI Press / MIT Press, 3-46.

[Buschmann *et al.*, 2000] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.

[Calvary *et al.*, 1997] Gaëlle Calvary, Jöelle Coutaz, Laurence Nigay, From Single-user Architectural Design to PAC*: A Generic Software Architecture Model for CSCW. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* ACM Press, 1997, 242-249.

[CLIPS, 2005] CLIPS: A Tool for Building Expert Systems. http://www.ghg.net/clips/CLIPS.html

[CORBA, 2002] Common Object Request Broker: Core Specification. Available as http://www.omg.org/cgi-bin/apps/doc?formal/02-12-02.pdf

[Coutaz, 1987] Coutaz, Jöelle, PAC, an Object Oriented Model for Dialog Design. *Proceedings of Interact'87*. 431-436.

[Dewan, 1999] Dewan, P., Architectures for collaborative applications. In: Beadouin-Lafon (eds.), *Computer Supported Cooporative Work*, John Wiley & Sons, 1999, 169-194.

[Finin *et al.*, 1994] Finin Tim, Fritzson Rich, McKay Don and McEntire Robin, KQML -- A Language and Protocol for Knowledge and Information Exchange, *13th International Distributed Artificial Intelligence Workshop,* 28 - 30.

[FIPA00001] FIPA Abstract Architecture specifications. Draft, 2002. Available as http://www.fipa.org/specs/fipa00001/SC00001L.html

[FIPA00019] FIPA 97 Part 1 Version 2.0: Agent Management Specification. Draft, 2001. Available as http://www.fipa.org/specs/fipa00019/ OC00019A.html

[FIPA00003] FIPA 97 Part 2 Version 2.0: Agent Communication Language. Draft, 2001. Available as http://www.fipa.org/specs/fipa00003/ OC00003A.html.

[FIPA00017] FIPA 97 Part 3 Version 1.0: Agent Software Integration Specification. Draft, 2001. Available as http://www.fipa.org/specs/fipa00012/OC00012A.html

[Laurillau and Nigay, 2002] Yann Laurillau and Laurence Nigay, Clover Architecture for Groupware. *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, ACM Press, 2002, 236-245.

[Nigay and Coutaz, 1991] Laurence Nigay and Jöelle Coutaz, Building User Interfaces: Organizing Software Agents. *Proceedings of Esprit'91*, ACM Press, 1991, 707-719.

[Nigay and Coutaz, 1993] Laurence Nigay and Jöelle Coutaz, A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion. *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, 1993, 172-178.

[Nigay and Coutaz, 1995] Laurence Nigay and Jöelle Coutaz, A Generic Platform for Adressing the Multimodal Challenge. *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co, 1995, 98-105.

[Patil *et al.*, 1993] Patil Ramesh S., Fikes Richard E., Patel-Schneider Peter F., McKay Don, Finin Tim, Gruber Thomas, Neches Robert, The DARPA Knowledge Sharing Effort: Progress Report, *Proceedings of the Third International Conference of Principles of Knowledge Representation and Reasoning,* Morgan Kaufmann, 1993, 777-788.

[Patomäki *et al.,* 2004] Saija Patomäki, Roope Raisamo, Jouni Salo, Virpi Pasto, Arto Hippula, Experiences on haptic interfaces for visually impaired young children. *Proceedings of the 6th International Conference on Multimodal Interfaces (*ICMI'04), ACM Press, 2004, 281-288.

[Saarinen *et al.,* 2005] Rami Saarinen, Janne Järvi, Roope Raisamo, Jouni Salo, Agent-based Architecture for Implementing Multimodal Learning Environments for Visually Impaired Children. *Proceedings of the 7th International Conference on Multimodal Interfaces (*ICMI'05), ACM Press, 2005, 309-316.

[Salber *et al.*, 1995] Daniel Salber, Jöelle Coutaz, Laurence Nigay, Extending the Scope of PAC-Amodeus to Cooporative Systems, *ACM SIGOIS Bulletin, Special issue: workshop write-ups and positions papers from CSCW'94*, 15 (3), 1995, 30-34.

[Sensable, 2005] SensAble Technologies Inc. http://www.sensable.com/

[Sjöström, 2001] Calle Sjöström, Designing haptic computer interfaces for blind people. *In Proceedings of Sixth International Symposium on Signal Processing and its Applications* (ISSPA 2001), IEEE, 2001.

[VRML, 1997] Part 1 of ISO/IEC 14772-1:1997. Information technology - Computer graphics and image processing - The Virtual Reality Modeling Language (VRML) - Part 1: Functional specification and UTF-8 encoding. Available as http://tecfa.unige.ch/guides/vrml/vrml97/spec/.

Monen käyttäjän käyttöliittymäarkkitehtuurit

Matti Voutilainen

Tiivistelmä

Tutkielmassa tarkastellaan aluksi millaisia ongelmia syntyy monen käyttäjän järjestelmissä ja mitä vaatimuksia se aiheuttaa verrattuna yhden käyttäjän järjestelmään. Järjestelmässä voi olla monta käyttäjää joko samaan aikaan, tai eri aikaan. Eri aikaan tapahtuvissa toimissa ei yleensä synny ongelmia näytön suhteen, mutta komentojen korjaaminen on ongelma. Mitä tapahtuu muiden komennoille, jos joku muuttaa aikaisempia käskyjään. Samanaikaisessa käytössä tulee lisäksi huolehtia näytön hallinnasta.

Toimintaa ja vaatimuksia kuvataan aluksi esimerkkien avulla. Samalla pyritään pohtimaan keinoja millä tavoin näitä ongelmia voidaan ratkaista. Lopuksi kuvataan muutamia käytännön sovellutuksia joilla monen käyttäjän käyttöliittymä on ratkaistu.

Avainsanat ja -sanonnat: samanaikaisuus, kerroksellisuus, yhteistoiminta ja hallinta.
CR-luokat: C2.1, D1.3, H4.3, H5.2, H5.3

1. Johdanto

Ihmisten kesken ja ihmisen ja erilaisten koneiden välillä täytyy tapahtua jonkinlaista kommunikointia, että yhteistyö olisi mahdollista. Kommunikointia tapahtuu myös koneiden välillä, monet automaatiojärjestelmät ovat hyvä esimerkkejä tämäntapaisesta toiminnasta. Toimintoja ohjaavat päätökset, jos nyt voidaan sanoa koneiden tekevän päätöksiä, syntyvät aina ulkopuolelta saadun tiedon tai tietojen perusteella.

Päätöstä tekevän järjestelmän on helppo toimia, jos tietoa tulee vain yhdestä paikasta ja tieto on on/off-tyyppistä, koska päätöskin on yleensä joko/tai. Monissa tapauksissa tullaan aivan hyvin tällä toimintaperiaatteella. Yhä enemmän kuitenkin toiminnot muuttuvat siihen suuntaan, että tietoa tulee hyvin monesta suunnasta, tiedot voivat olla hyvinkin ristiriitaisia, tiedon merkitykseen

vaikuttaa tuleeko se ennen vai jälkeen jonkun toisen tiedon, järjestelmän ulos antama syöte vaikuttaa toisten järjestelmien kautta sen saamaan seuraavaan tietoon. Pörssimeklari on esimerkki tästä ns. monen käyttäjän järjestelmästä. Hän saa tietoa monen kanavan kautta ja monenlaista tietoa, tieto voi olla puolueetonta tai manipuloimaan pyrkivää lisäksi hänen toimintonsa vaikuttavat toisten meklarien päätöksiin. Käyttöliittymänä ympäristöön toimii korvat, silmät, suu ja näppäimistöllä olevat sormet. Meklarin on ennen päätöksen tekoa arvioitava kunkin tiedon luotettavuus ja painoarvo suhteessa muihin tietoihin. Sääennusteiden tekojärjestelmät toimivat myös monen erilaisen ja eritaholta tulevan tiedon varassa. Yksittäisen meteorologin tai järjestelmän ennuste ei kuitenkaan vaikuta lopputulokseen, jos sää muuttuu sateiseksi, se tapahtuu ennustajasta ja hänen ennusteestaan huolimatta.

Suuria monen käyttäjän järjestelmiä esiintyy yhteiskunnan hallinnossa ja liike-elämässä. Suuruuden lisäksi niissä voi ongelmana monenlaiset käyttäjät, koska kaikilla ei välttämättä hyviä tietoteknisiä valmiuksia. Siitä huolimatta ulos tulevan tiedon on oltava virheetöntä esim: matkojen varausjärjestelmä ei saa myydä samaa hotellihuonetta ainakaan kolmea kertaa enempää.

Tietojärjestelmään voi tietoa tulla monelle eri tavalla ja eri muodossa. Erilaiset anturit voivat kerätä: ääntä, valoa, hajuja, lämpötilaa, fyysisiä mittoja, nopeutta, painoa jne. Ihminen voi tuottaa sinne tietoa näppäimistön ja hiiren kautta. Tiedot voivat tulla jonossa tai yhtä aikaa, jolloin järjestelmän on kyettävä päättämään missä järjestyksessä se tietoja lukee. Tiedot on saatava myös samanmuotoisiksi ennen niiden lopullista käsittelyä.

2. Mahdolliset käyttäjät

2.1. Konekäyttö

Automaatiojärjestelmät ovat yhteydessä toisiinsa ja niiden toiminta on oltava hyvin koordinoitua vaikka niillä ei olisikaan ulospäin näkyvää käyttöliittymää muuten, kuin ohjelmointia varten.

Sahoilla on käytössä järjestelmä, joka pyrkii optimoimaan tukista saatavan sahaustuloksen. Osana tähän järjestelmään kuuluu särmäyksen optimointi järjestelmä. Järjestelmä mittaa kanttaamattoman laudan pituuden ja leveyden sekä kapeneman lisäksi se ”näkee” puutavarassa olevat oksat ja muut viat. Näiden tietojen perusteella optimointiohjelma määrittää kuinka pitkän ja leveän

laudan se kanttaamattomasta aihiosta tekee. Lisäksi on huomioitava mahdolliset prioriteetit vaatimukset. Lopputulosta säätelee kolme vaatimusta ”asiakasta”: pituus, leveys sekä oksat ja muut viat, näiden tietojen avulla ohjelmiston on kyettävä maksimoimaan taloudellinen tulos. Järjestelmässä ei ole ulospäin näkyvää käyttöliittymää, koska yhden laudan käsittelyyn on aikaa 1 – 3 sekuntia, joten ihminen ei ehtisi reagoida mitenkään päätöksen tekoon. Järjestelmän sisällä on sen oma sisäinen käyttöliittymä, joka on yhteydessä tietokantaan ja ohjeistaa terien liikkeitä ja siirtää valmiin laudan tiedot kirjanpitoon.

2.2. Ihminen käyttäjänä

Ihminen voi saada tietoa ainoastaan aistiensa välityksellä. Hän voi antaa tietoa ulospäin ainoastaan erilaisilla liikkeillä, puhuminenkin on liikettä, joka tuottaa ääntä. Kykymme vastaanottaa tietoa on paljon monipuolisempi kuin tuottaa sitä. Koneen ja ihmisen välisen käyttöliittymän on puolin ja toisin sopeuduttava tähän tosiasiaan. Ihmistä emme voi juurikaan muokata, joten konetta on sopeutettava.

Jalkapallossa yleensä ainakin kaksi pelaajaa kamppailee pallosta, kumpi saa sen haltuunsa. Monen käyttäjän järjestelmässä on samantapainen tilanne: kuka hallitsee tilannetta, kenen komennot jäävät voimaan, joskaan ei avoimesti niin kilpailullinen. Konferenssi ja opetusjärjestelmissä henkilöt ovat hyvinkin kaukana toisistaan. Se kenen kuva näkyy ja ääni kuuluu toisille, on ratkaistava tavalla tai toiselle, joku voi toimia puheenjohtajana ja hallita kaikkien näkymää tai asia hoidetaan ohjelmallisesti niin, että esim. kovaäänisin osallistuja saa järjestelmän haltuunsa. Näissä tilanteissa käyttäjiä on yleensä hyvin rajallinen määrä.

Jos käyttäjiä on suuri määrä, on niiden hallinnointi järjestettävä. Samaa tietoa voi pyrkiä käsittelemään monta käyttäjää yhtä aikaa esim: pankkitilille voi tulla samaan aikaan merkintöjä monesta lähteestä yhtä aikaa ja joku voi silloin tehdä virheen ja pyrkiä samalla korjaamaan sen. Järjestelmä ei saa toimia siinäkään tilanteessa virheellisesti. Komentojen kumoamisesta ja uudelleen suorittamisesta on kuvaus Jussi Rantalan seminaarityössä, jossa hän kuvaa GINA-järjestelmää. Mukaan voi tulla käyttäjiä, joilla on eritasoisia oikeuksia toimia järjestelmässä, kaikki nämä on tunnistettava. Tietojen käsittelyn tai varastoinnin nopeus on oltava ainakin periaatteessa riippumaton käyttäjien lukumäärästä. Toisaalta jokaisen käyttäjän oma näyttö on oltava pääsääntöisesti käyttäjän hallinnassa ainakin niiltä osin, kun hän tuottaa siihen komentoja. Näyttö voi yhtenäinen tai sitten jokaiselle käyttäjällä on alueensa.

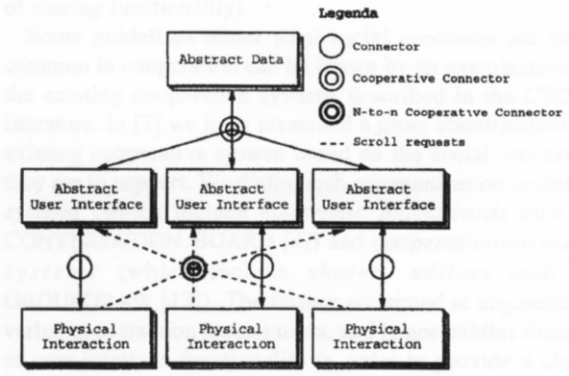
Keskustelupalsta on yksinkertainen monen käyttäjän järjestelmä, jonne jokainen tuottaa vuorollaan tekstiä ja kaikkien näytöt toimivat reaaliajassa.

3. Arkkitehtuurin vaatimukset

Kun monen käyttäjän järjestelmät alkoivat yleistyä suuressa määrin 1980-luvun lopulla, ne olivat pääsääntöisesti kaksitasoisia, käyttäjä oli omalta koneeltaan suoraan yhteydessä tietokantaan, yleensä vielä ns. tyhmällä päätteellä. Käyttäjien määrät olivat vähäisiä ja käyttäjät olivat hyvin koulutettuja tehtäviinsä, joten suuremmilta ongelmilta, ehkä odotusaikoja lukuun ottamatta vältyttyiin. Laitteiden käyttöjärjestelmät ovat kehittyneet tuon ajan jälkeen huomattavasti, Unix on käyttöjärjestelmä [Immonen 2004], joka on alun perin suunniteltu monen käyttäjän järjestelmäksi.

Käyttäjien ja käsiteltävien tietojen määrän lisääntyessä kaksi tasoisilla järjestelmillä ei tulla enää toimeen varsinkaan, kun käyttäjien taidot ovat hyvin monenlaisia. Käyttöliittymän ja varsinaisen tietokannan välissä on oltava yksi tai useampia sovelluslogiikkakerroksia, joista ainoastaan alimmainen kerros on yhteydessä tietokantaan. Välikerrokset hoitavat myös mahdollisesti tarvittavat käyttäjien väliset yhteydet. Nämä viestit voivat olla tarkoitettuja kaikille mukanaolijoille tai sitten ne voivat olla kahdenkeskisiä, jolloin muut eivät tiedä niistä. Sovelluslogiikka kerrosten käyttö parantaa myös tietoturvaa, koska käyttäjä ei pääse suoraa tietokantaan.

Kuvan 1, mukaisessa arkkitehtuurissa kontrolloidaan kunkin käyttäjän alhaalta päin tulevia pyyntöjä, jotka eivät kohdistu omaan rajapintaan. Tällöin jokaisella käyttäjällä on ensisijaisesti hallinta omaan koneeseen ja toissijaisesti samassa järjestelmässä olevien toisten koneeseen.

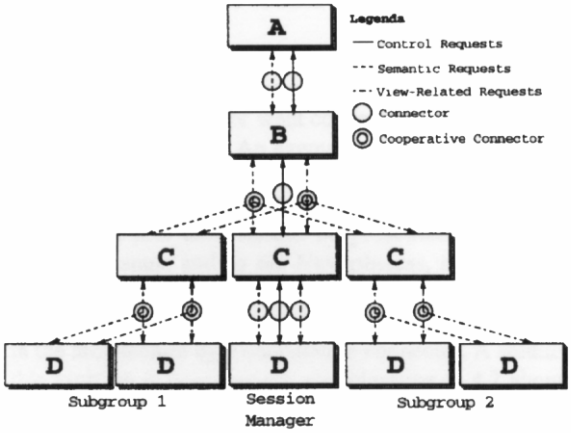


Kuva 1.Järjestelmä joka rajaa kullekin käyttäjälle erilaiset oikeudet eri koneisiin.

Monitasoratkaisuilla saadaan ns. tilaton ratkaisu joka estää turhan resurssien sitomisen. Tilaton ratkaisu tarkoittaa [Flink 2001] ”Tilattomalla tarkoitetaan, että esim: ohjelmakutsu sovelluspalvelimelle luo olion, joka suorittaa halutun toiminnon, jonka jälkeen olion varaama muisti vapautetaan. Asiakaskoneelta ei siis voi luoda oliota ja myöhemmin kutsua sen metodeja. Tällä tavoin liikenne verkossa pysyy pienempänä ja sovelluspalvelimen resursseja säästyy, koska muistia ei tarvitse varata turhaan.”

Monissa ryhmätilanteissä on tarpeen, että joko kykenee kontrolloimaan ryhmän käyttäytymistä, kuten aikaisemmin todettiin konferenssien ja opetustilanteiden osalta. Kuvassa 2 esitetty arkkitehtuuri kuvaa järjestelmää, jossa tasolla D on käyttäjät ja kunkin alaryhmän johtajat. Alaryhmän johtaja kykenee kontrollimaan oman ryhmänsä jäsenten toimia, mutta ei muiden ryhmien jäseniä ja ryhmiä kokonaisuudessaankaan.

Tapahtuman vetäjä kykenee halutessaan kontrolloimaan jokaisen ryhmän näyttöä ja puuttumaan ryhmän toimintaa.



Kuva 2. Järjestelmä jolla kunkin ryhmän vetäjä voi säädellä oman ryhmänsä koneita.

4. Erilaisia monen käyttäjän ratkaisuja

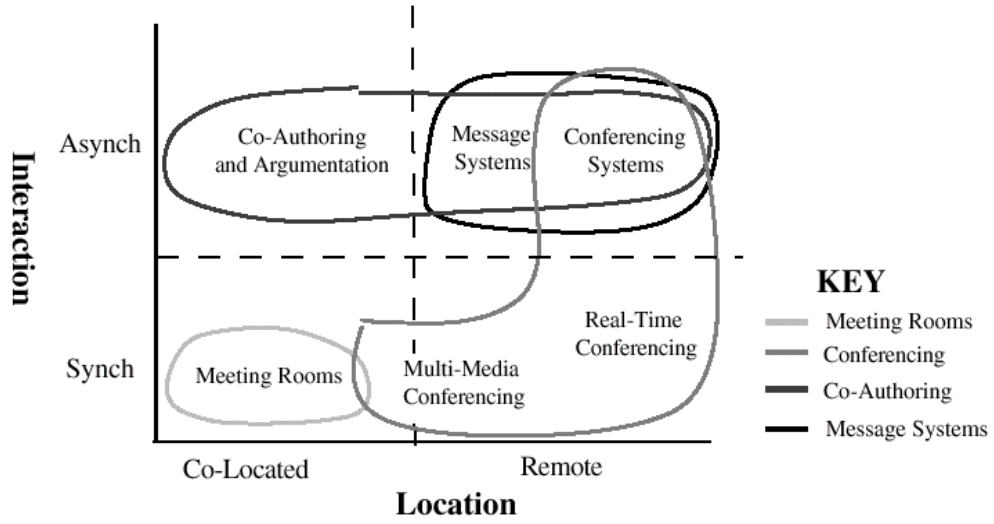
4.1. Yhteistyöympäristöt

Tietotekniikka on laajentanut tilan käsitettä aivan uusiin ulottuvuuksiin. Se on tuonut käsitteen ”älykäs” tila. Yksi malli älykkäästä tilasta on [Kainulainen 2005] seminaarityö, jossa kuvataan miten tilan tapahtumista luodaan äänimaisema. Käsitteen määrittely on kuitenkin huomattavasti vanhempi kuin tietotekniikan varsinainen läpimurto. Käsite muotoiltiin jo 1960-luvulla [Yhtymp 2005]

”Stanford Research Insituutissa esitettiin tietokone-ihminen malli, joka lisäsi ryhmätyöskentelyn tehokkuutta. Kuvassa 3 on esitetty malli, joka oli ensimmäinen visio ”älykkäästä tilasta” MIT esitteli 1970-luvulla ”mediahuoneen”, jossa tietokonetta ohjattiin puheella ja eleillä koneen vastatessa tekstein ja kuvin.”

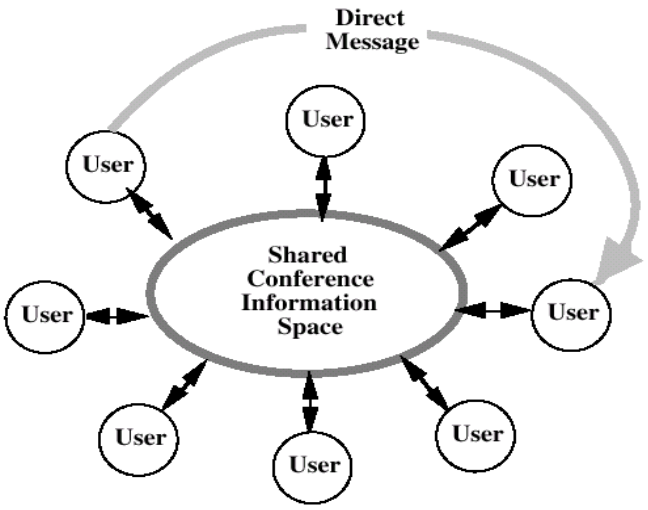
Älykkään tilan tulee kyetä ottamaan vastaan ja ymmärtämään hyvinkin erilaisilla tavoilla tuotettua tietoa. Piilossa oleva käyttöliittymä seuraa ihmisten välistä keskustelua ja osaa itse tarjota pyytämättä omia palveluksiaan. Puheentunnistus on käyttöliittymän tärkein osa. [Yhtymp 2005] ”Yksittäisten ihmisten puheen tunnistaminen on nykyään mahdollista, mutta kone ei vielä osaa seurata keskustelua. Eleet ovat tärkeä osa normaalia keskustelua ja niiden tunnistaminen on tärkeää.”

Seuraavassa suora lainaus, jossa kuvataan ns. älykkään tilan toimintaa. [Yhtymp 2005] ”Luovassa yhteistyössä keskustellaan, vaihdetaan tietoa ja kehitellään yhdessä ideoita. Aivoriihissä osallistujien toiminta on vapaamuotoista ja muuttuvat tilanteet *vaativat nopeaa tietojen vaihtoa osallistujien välillä. Ideat tulevat ja kehittyvät nopeasti toisten osallistujien ajatuksia edelleen kehittämällä*. Suunnittelu ja kehitystehtävät voidaan myös toteuttaa ennalta sovitulla toimintatavalla. Tällöin annetut tehtävät tehdään omassa tahdissa, riippumatta siitä, kuinka muiden työt etenevät. Järjestelmä seuraa kaikkien töiden etenemistä ja tiedottaa siitä osallistujia. Ensin kuvattu tapa välittää tietoa on synkroninen, jälkimmäinen taas asynkroninen. Molempia tarvitaan yhteistyöjärjestelmissä.”



Kuva 3. Älykkään tilan malli 1970-luvulta.

Reaaliaikaisessa tilanteessa tietoa pitää voida lähettää tarpeen mukaan kaikille tai ainoastaan valituille henkilöille. Tarpeen tullen on kyettävä äänestämään. Puheenjohtajan on voitava ohjata konferenssin kulkua ja tehdä muutoksia yhteiselle työalueelle. Järjestelmän on toimittava samalla tavalla riippumatta siitä ovatko osanottajat samassa huoneessa vai eripuolella maailmaa, PC:n, kännykän tai jonkun muun kommunikointivälineen ääressä.



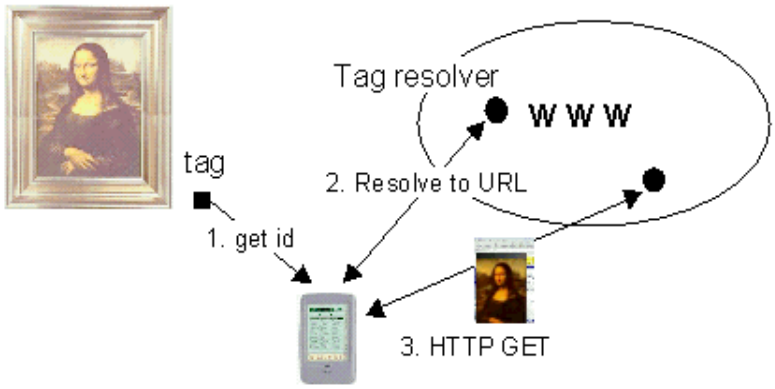
Kuva 4. Järjestelmä konferenssin ohjaukseen, joka mahdollistaa yhteisen muistion.

Kommentointijärjestelmät, jotka on esitetty kuvassa 4, tarjoavat monen osapuolen väliset kommentoinnit ja neuvottelut. Mallin perusperiaatteena ovat keskustelut suunnitteluprosessien osallistujien välillä. Jokaiselle osallistujalla on oma näkökulma asioihin. Yhteinen ratkaisu saadaan neuvottelujen päätteeksi.

Yhteisjulkaisujärjestelmissä käyttäjät lukevat julkisesti saatavilla olevan julkaisun ja lisäävät siihen omat huomionsa. Kommentteja voi olla kahdenlaisia. Yksityisiä, jolloin ne näkyvät vain tekijälle tai julkisia jolloin ne näkyvät kaikille, joille niiden halutaan näkyvän.

Kuten edellä todettiin voi konferenssin osanottaja tai muu käyttäjä olla missä tahansa, mutta hän voi myös kiinnostua mistä tahansa ja tarvita tiedot kiinnostuksen kohteesta nopeasti päätelaitteeseensa. Järjestelmä ylläpitää kohteiden ja paikkojen tunniste tietoja palvelimella. Käyttäjän halutessa tietoa jostakin asiasta hän lähettää tunnisteen tiedot ja saa palautteena kohteeseen liittyvä www-sivu.

Kuvassa 5 on tilanne, jossa joku on kiinnostunut Mona-Liisasta.

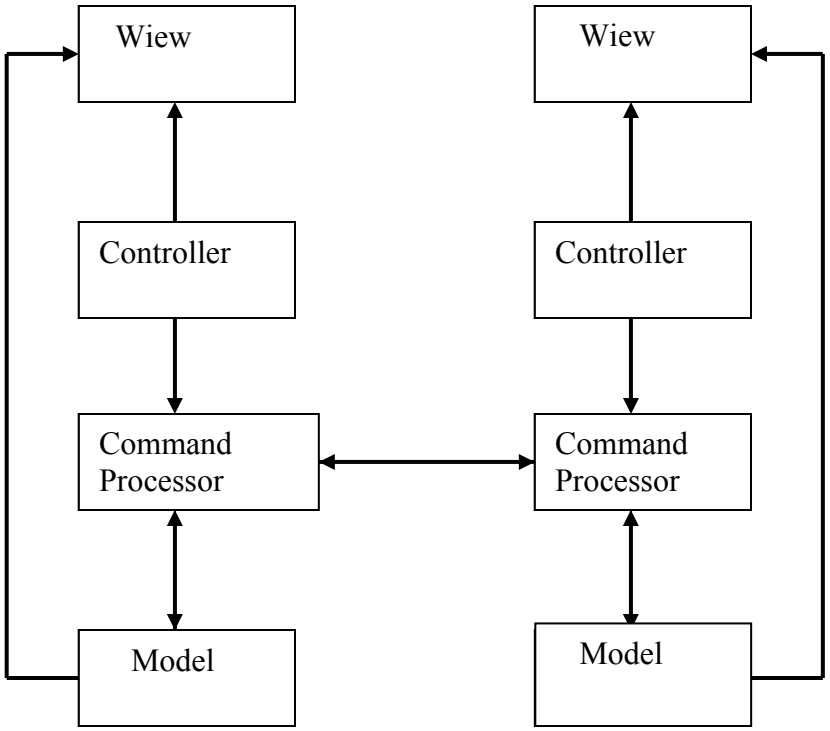


Kuva 5. Yhteishakujärjestelmä joka perustuu kohteessa (Mona Lisa) olevaan tunnisteeseen.

4.2. MVC-malli

Varsinainen sovellusohjelmisto voidaan erottaa käyttöliittymästä, jolloin järjestelmän uudelleen muotoilu ja muokattavuus helpottuvat huomattavasti. Tätä asiaa käsittelee Seminaarityössään Juuso Näsi [Näsi 2005], kuten myös MVC-mallia.

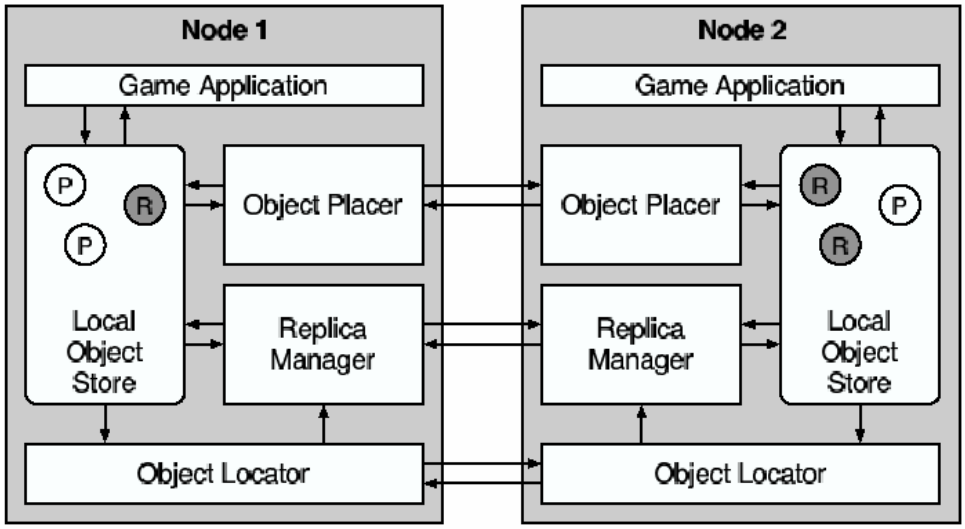
MVC-malli ei ole alun perin kehitetty monen käyttäjän malliksi [CSCW2003], mutta sitä on muokattu kuvan 6 kaavion mukaiseksi, jolloin sen mahdollista toimia monen käyttäjän järjestelmänä.



Kuva 6. MVC-malli.

4.3. Colyseus

Pelit, joissa voi olla yhtä aikaa monta osallistujaa ovat vaativia ympäristöjä, koska pelaajan menestys voi olla kiinni siitä kuinka hänen komentojaan käsitellään. Pelaaminen [Colyseus 2005] tapahtuu yhdessä paikassa, johon pelaajat voivat olla yhteydessä verkon välityksellä. Tapahtumia on seurattava 10 – 20 kertaa sekunnissa, tämä verraten alhainen päivitys taajuus, johtuu pelihahmojen hitaudesta.

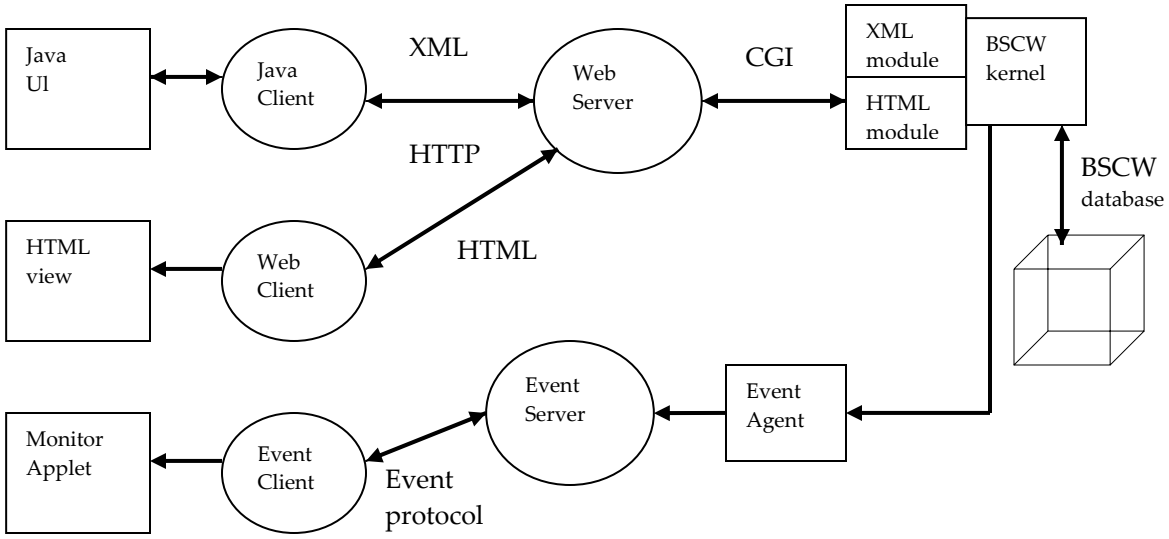


Kuva 7. Colyseuksen kolmiosainen rakenne, joka pitää pelaajat tasa-arvoisina.

Kuvassa 7 on kuvattu colyseuksen rakenne, joka muodostuu kolmesta komponentista. Yksi huolehtii kohteiden siirrosta, yksi paikantamisesta ja yksi näytön päivittämisestä toisten pelaajien osalta.

4.4. BSCW (Basic Support for Cooperative Work)

BSCW-järjestelmä, kuva 8, on suunniteltu tutkijoiden käyttöön ja tarjoaa synkronisia ja asynkronisia välineitä. Järjestelmää käytetään myös oppimisalustana Moodlen tai WebCT:n tapaan. [BSCW 2005] ”BSCW:tä ei kenties voi luokitella varsinaiseksi oppimisalustaksi, vaan yhteistoiminnallisen ryhmätyön välineeksi.” Järjestelmä on rakenteeltaan modulaarinen, joten siihen on helppo lisätä uusia ominaisuuksia.



Kuva 8. BSCW-järjestelmään voidaan liittää erilaisia komponentteja ja ominaisuuksia.

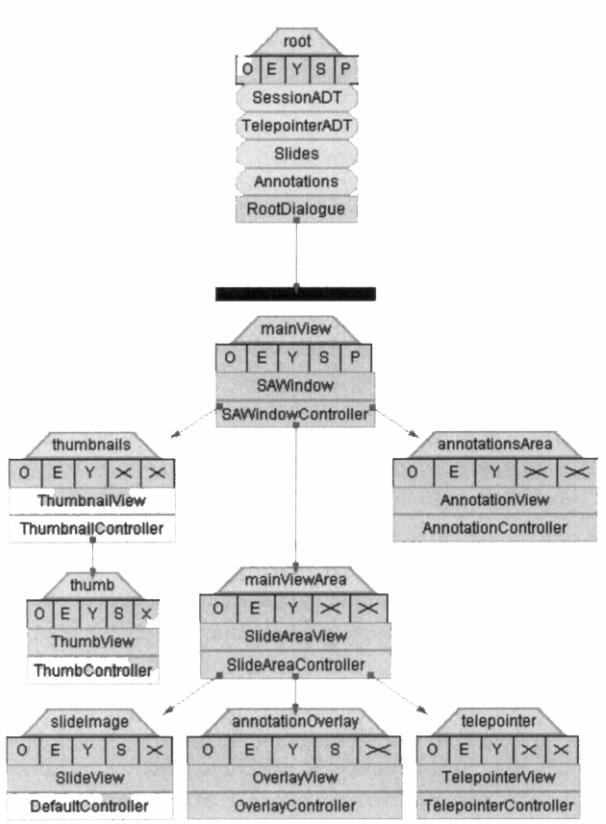
[Yhtymp 2005] ”Järjestelmä käyttää rajapintana Common Gateway Interface (CGI). Palvelimen ja asiakkaan välinen käyttöliittymä on toteutettu HTML-sivuina ja käyttäen HTTP-protokollaa sivujen osoittamiseen. Koska HTML ei ole kovin hyvä käyttöliittymän toteuttamiseen, on myös Javalla toteutettu XML:ää käyttävä käyttöliittymä nykyisin saatavilla. Toiminnallisuuden parantamiseksi BSCW:ssä on WWW-palvelimen lisäksi ns tapahtuma palvelin (event server) ja Javalla toteutettu monitorisovellus (monitor applet), jolla voidaan seurata muiden käyttäytymistä järjestelmässä.”

4.5. Sudenkorento (Dragonfly)

Seuraavassa käydään läpi miten ohjelmisto ja laitearkkitehtuurit ovat monen käyttäjän järjestelmissä riippuvaisia toisistaan, kuten ne ovat kaikissa muissakin järjestelmissä tosistaan ja kaikilla tasoilla.

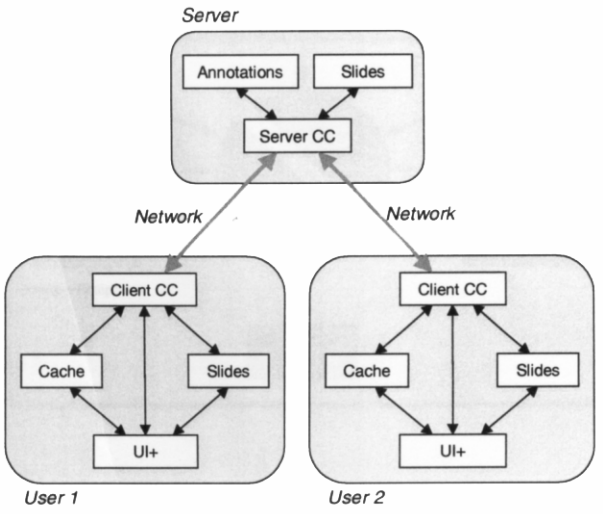
Järjestelmä toimii täysin reaaliaikaisesti, jokaisen käyttäjän toiminnot ovat kaikkien nähtävissä samalla hetkellä.

Kuvassa 9 on sudenkorento järjestelmän ohjelmistoarkkitehtuuri. Järjestelmä on hyvin vahvasti hierarkkinen. Tällä menettelyllä on haluttu varmistaa, että komponenttien keskinäinen toiminta on koko ajan hallinnassa.



Kuva 9. Sudenkorentomalli on hyvin hierarkkinen.

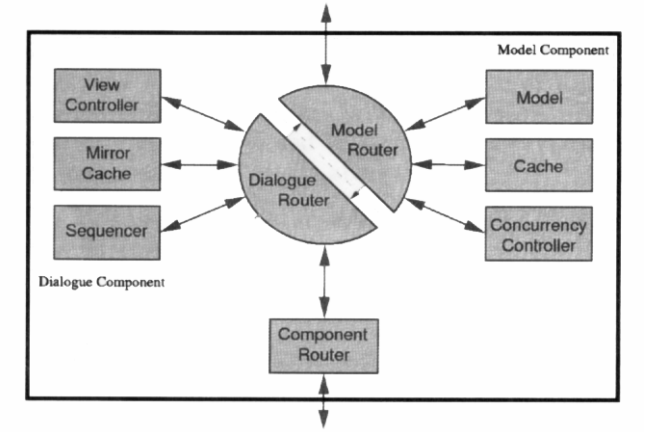
Kuva 10 kuvataan, että laitearkkitehtuuri rakentuu tai ainakin pitäisi rakentua ohjelmisto arkkitehtuurin ehdoilla.



Kuva 10. Laitteiston ja ohjelmistoarkkitehtuurin suhde.

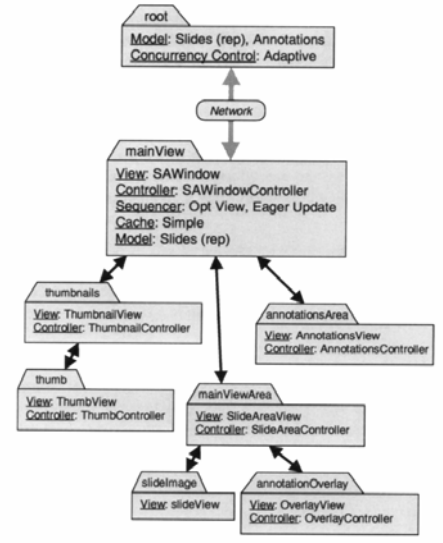
Kuvassa 11 näkyvät komponenttien ja toimintojen väliset suhteet. **Näytön** valvonta on järjestelmällä niin kauan kunnes käyttäjä ottaa sen haltuunsa. **Mirror** komponentti huolehtii, että kaikki tulevat tiedot ovat järjestelmän ymmärrettävissä. **Jaksottaja** huolehtii, että tiedot ja näytön päivitys tapahtuvat oikeassa järjestyksessä.

Mallinnus osa tuottaa tarpeen vaatiessa kopion muista sudenkorennon komponenteista. **Cache** osa varastoi tehtävä pyynnöt ja tiedot kunnes niitä tarvitaan muualla.



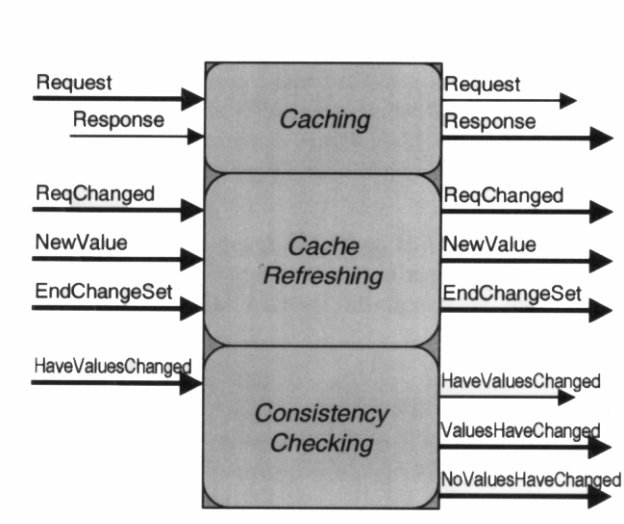
Kuva 11. Sudenkorennon komponentit ja niiden väliset suhteet.

Kuten jo aikaisemmin todettiin, järjestelmä on vahvasti hierarkkinen, sama koskee myös näyttöjen hallintaa, kuva 12.



Kuva 12. Sudenkorennon näyttöjen hallinnan hierarkia.

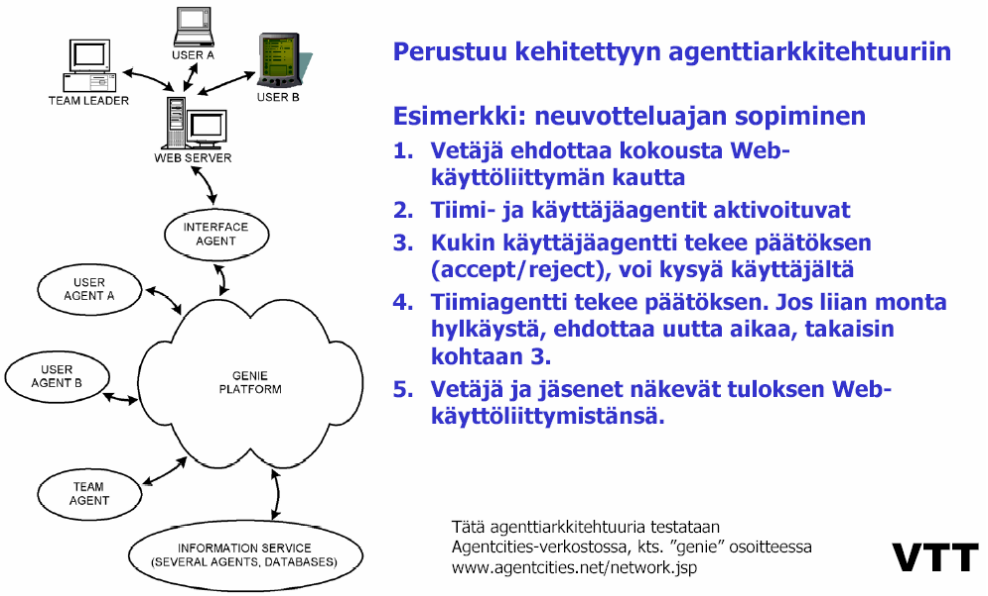
Kuvassa 13 on kuvattu millä tavoin tapahtumien käsittely tapahtuu järjestelmän sisällä



Kuva 13. Tietojen käsittely ja muutokset sudenkorennon sisällä.

4.6. Genie arkkitehtuuri

Oulun yliopistossa on kehitetty yhdessä monien muiden toimijoiden kanssa järjestelmää, joka näkyy kuvassa 14, rakenne on itsessään aktiivinen, kuten alla oleva kalenteri osio osoittaa. [Riekki 2003]



Kuva 14. Genie-arkkitehtuurin rakenteen ja toiminnan periaate.

Genie: kerää tietoa käyttäjistä ja ympäristöstä, tunnistaa tilanteen kerätystä tiedosta, päättelee saadun tiedon perusteella mitä palveluja käyttäjä tarvitsee, suorittaa tarvittavat toimenpiteet.

5. Yhteenveto

Tulevaisuudessa yhteiskäytössä olevat ohjelmistot tulevat olemaan yhä merkittävämmässä asemassa. Näihin aikoihin saakka yhteisohjelmistoja ovat käyttäneet lähinnä ”ammattilaiset” tai innokkaat tietotekniikan harrastajat. Syynä tähän on ollut näiden järjestelmien ainakin näennäinen käyttämisen vaikeus. Tämä käyttämisen este poistuu tai ainakin vähenee, kun väestön yleinen tietotekninen osaaminen nousee ja laitteiden suorituskyky paranee. Ryhmäohjelmien puolesta puhuu myös tietoliikenneyhteyksien hinta. Kiinteä kuukausimaksu internet liittymästä houkuttaa käyttämään sitä aikaveloitettua, ainakin vielä tällä hetkellä, puhelinta enemmän. Ryhmätilanteessa voi jokaisen näytöllä näkyä muiden mukana olijoiden kasvot ja samalla voidaan käsitellä kaikille yhteistä aluetta. Tietenkin näytön koko asettaa rajoituksia kuinka suuri joukko tämän tapaisessa toiminnassa voi olla mukana. Kun ohjelmistojen kehittyvät sen laatuisiksi, että kaikki yhteisessä keskustelussa olevat tuntevat olevansa samassa olohuoneessa eikä ohjelmiston käyttäminen hallitse tilannetta, tulee niistä jokamiehen oikeuksia.

Yhteiskäyttö ohjelmistot lisääntyvät todennäköisesti, myös mobiililaitteissa ja sulautetuissa järjestelmissä [Kähkönen 2003], varsinkin jos niiden käyttö halpenee tai niihin tulee kiinteitä kuukausi maksuja jolloin kustannukset eivät rajoita niiden käyttöä.

Käyttöliittymätutkimus etenee varmaankin moneen suuntaan, mutta yksi suunta on määritelty [USIX 2003] seuraavasti ”Tiedon ja kokemusten jakaminen (desin for experience) on ajankohtainen käyttöliittymätutkimukseen liittyvä teema. Tällöin käyttöliittymätutkimuksessa huomioidaan yhteisön toimintatapoja, ideoita ja kielenkäyttöä sekä toimintatapaa verkossa. Nämä tekijät pyritään kehittämään niin, että ne tukevat käyttäjän oman identiteetin mieltämistä ja rakentamista. Tähän liittyvät mm. osakulttuurit kuten esimerkiksi mitä murretta kyseisessä yhteisössä puhutaan. Aiheeseen liittyvän tutkimuksen tuloksia voi soveltaa esimerkiksi niissä kunnallisissa palveluissa, joissa sosiaalinen vuorovaikutus on toimintatapana.”

Viiteluettelo

[Anderson 2000] Gary E. Anderson, T.C. Nicholas Graham ja Timothy N. Wright, Dragonfly: *Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems*, Queen University, Canada, 2000

[BSCW 2005] http://www.cs.helsinki.fi/group/vertti/vertti/opal3.shtml

[Colyseus 2005] Ashwin R. Bharambe Jeff_ Pang ja Srinivasan Seshan *A Distributed Architecture for Interactive Multiplayer Games 2005* http://reports-archive.adm.cs.cmu.edu/anon/2005/CMU-CS-05-112.pdf

[CSCW2003] Mike Blackstock, http://www.cs.ubc.ca/~michael/publications/CSCWArchitecture.pdf

[Flink 2001] Tommi Flink, *Gradu, Siirtyminen monitasoarkkitehtuuriin: Microsoft.net tarjoamat mahdollisuudet*, Jyväskylän yliopisto 2001

[Forssel 2005] http://193.167.110.132/marko.forsell/Kalvot/22_Oliosuunnittelu.pdf

[Gimo Dig] *Technology watch*, http://gimodig.fgi.fi/watch.php

[Immonen 2004] Markku Immonen, *Käyttöjärjestelmien vertailu* http://www.saunalahti.fi/wpoet/fin/os.htm

[Kainulainen 2005] *Seminaarityö, Äänimaisematyyppiset käyttöliittymäarkkitehtuurit,* Tampereen yliopisto 2005.

[Koskimies & Mikkonen 2005] Kari Koskimies ja Tommi Mikkonen, *Ohjelmistoarkkitehtuurit*, Gummerus, Jyväskylä 2005

[Kähkönen 2003] Juha Kähkönen 2003, *Sulautetun ohjelmiston päivitys siirtoverkkosolmussa* http://users.tkk.fi/~jekahkon/d/sulautetun.pdf

[Meisalo & Sutinen & Tarhio 2003] Veijo Meisalo, Erkki Sutinen ja Jorma Tarhio, *Modernit oppimisympäristöt*, Tietosanoma, Helsinki 2003

[Näsi 2005] Juuso Näsi, *Seminaarityö Sovelluslogiikan eriyttäminen käyttöliittymästä*, Tampereen yliopisto 2005.

[Paoli 1996] Flavio De Paoli, Andrea Sosio, *Requirements for a Software Architecture Supporting Cooperative Multi-User Interaction,* University degli Studi, Milano 1996

[Prosa 2004 UML] *Prosa 2004 UML mallinnin – vaatimuksista suoritettavaan visuaaliseen malliin*, Insoft, http://www.insoft.fi/fin/pr2Prosa.htm

[Riekki 2003] Jukka Riekki 2003, *APRIKOOSI USIXista eteenpäin* http://websrv2.tekes.fi/opencms/opencms/OhjelmaPortaali/Paattyneet/USIX/fi/Dokumenttiarkisto/Viestinta_ja_aktivointi/Seminaarit/

Loppuseminaari/JukkaRiekki.pdf

[USIX 2003] *Usix-käyttäjäkeskeinen tietotekniikka 1999 – 2003*
http://www.tekes.fi/julkaisut/USIX_loppuraportti.pdf

[Yhtymp 2005] http://www.helsinki.fi/~amertani/ytymp/ytymp/htm

[Äyräväinen 2000] Seppo Äyräväinen, *Keinotodellisuuden soveltamismahdollisuudet sotilaskoulutuksessa*
http://www.tml.tkk.fi/Publications/Thesis/2000_ayravainen.pdf
http://www.prosessori.fi/es00/arkisto/PDF/LAAJENNU.PDF

Mobiililaitekeskeiset jokapaikan tietotekniikan käyttöliittymäarkkitehtuurit

Pasi Välkkynen

Tiivistelmä

Mobiililaitteet ovat nousseet merkittävään rooliin päätelaitteina jokapaikan tietotekniikassa. Esittelen tässä tutkielmassa seitsemän jokapaikan tietotekniikan järjestelmää, jotka hyödyntävät matkapuhelinta tai PDA-laitetta käyttöliittymänään ja päätelaitteenaan. Esitellyt järjestelmät demonstroivat erilaisten tekniikoiden, kuten paikalliskommunikaation, RFID:n, Bluetoothin ja antureiden käyttöä mobiililaitteen kautta.

Avainsanat ja -sanonnat: Mobiililaite, jokapaikan tietotekniikka, arkkitehtuuri, paikalliskommunikaatio, RFID, Bluetooth, matriisikoodi

1. Johdanto

Matkapuhelimet ja PDA:t (*Personal Digital Assistant*) ovat merkittävässä roolissa jokapaikan tietotekniikan sovelluksissa, koska niitä on lähes kaikilla käyttäjillä [Siegemund ja Flörkemeier, 2003] ja niiden mahdollistamat toiminnot ovat lisääntyneet. Mukana kulkevissa päätelaitteissa on tyypillisesti käyttöjärjestelmä, jolle muutkin kuin puhelimen valmistaja voivat luoda sovelluksia. Niissä on monipuoliset verkkoyhteydet ja verkkoa käyttäviä sovelluksia, kuten sähköpostiohjelmia ja WWW-selaimia. Mobiililaitteella pystyy käyttämään käytännössä samoja sovelluksia kuin pöytäkoneellakin, joskin näytön pieni koko ja syöttölaitteet rajoittavat toimintoja. Esimerkiksi monet WWW-sivut eivät mahdu pienelle näytölle. Lisäksi niiden verkkoyhteydet ovat usein hitaampia kuin pöytäkoneiden ja niissä on vähemmän tietojenkäsittelykapasiteettia.

Näille laitteilla on kehitetty vastapainona erilaisia pieniin laitteisiin soveltuvia syöttölaitteita ja paikalliskommunikaatiomahdollisuuksia (*local connectivity*), joiden avulla käyttäjä voi olla laitteensa kanssa vuorovaikutuksessa lähiympäristönsä palveluiden kanssa. Matkapuhelimien ja PDA-laitteiden lisäksi on olemassa muitakin ympäristönsä kanssa kommunikoivia mobiililaitteita, kuten

rannetietokoneet[6] ja monet terveydenhuollon laitteet, mutta toistaiseksi nämä laitteet ovat olleet erikoistuneita rajatuille sovellusalueille eikä niihin ole ollut yleisesti saatavilla kehitystyökaluja, jotka olisivat sallineet kenen tahansa kehittää niihin omia ohjelmistoja. Tämän takia keskityn tässä tutkielmassa matkapuhelimiin ja PDA-laitteisiin perustuviin käyttöliittymiin.

Tämän tutkielman tarkoituksena on esitellä erilaisia mobiililaitekeskeisiä jokapaikan tietotekniikan arkkitehtuureita. Seuraavissa luvuissa kuvaan niille yhteisiä laitteistoarkkitehtuureita sekä seitsemän erilaista mobiililaitteella käytettävää jokapaikan tietotekniikan järjestelmää. Tavoitteenani on, että yhdessä nämä erilaiset järjestelmät antavat kuvan mobiililaitekeskeisten jokapaikan tietotekniikan arkkitehtuurien tilasta.

2. Laitteistoarkkitehtuureista

Tyypillistä mobiililaitekeskeisille jokapaikan tietotekniikan sovelluksille on siis matkapuhelimen tai PDA:n kaltainen laite, jota ei ensisijaisesti käytetä viestimiseen pitkien matkojen päähän tai laitteen omien toimintojen (esimerkiksi kalenterin) käyttöön laitteen omalla näppäimistöllä tai muilla perinteisillä syöttölaitteilla. Sen sijaan päätelaite kommunikoi lähiympäristöön upotettujen tietoteknisten laitteiden kanssa ja niiden avulla mahdollistaa joko käyttäjän kontekstiin (usein paikka) perustuvia palveluita tai luonnollisempia vuorovaikutustapoja kuin perinteiset syöttö- ja tulostuslaitteet. Koska jokapaikan tietotekniikan sovellukset on tyypillisesti hajautettu päätelaitteen lisäksi ympäristöön, varsinainen käyttöliittymä sekoittuu usein järjestelmän koko toiminnallisuuteen.

2.1. Tageihin perustuvat sovellukset

Yksi merkittävä ryhmä mobiililaitteisiin perustuvia vuorovaikutustapoja jokapaikan tietotekniikassa ovat erilaiset tagit ja niiden lukijat. Tagi on tyypillisesti johonkin fyysiseen esineeseen liitetty hyvin pieni laite tai tunniste, joka sisältää jonkin yksinkertaisen tiedon ja joka voidaan lukea sopivalla lukijalla. Yleisiä tageja ovat RFID-tagit (*Radio Frequency Identification*) ja viivakoodit. Tagin sisältämä tieto voi olla yksinkertainen tunnistenumero, jolla esine voidaan erottaa muista esineistä, mutta tagi voi sisältää myös muuta tietoa, esimerkiksi esineeseen liittyvän WWW-sivun osoitteen (URL, *Universal Resource Locator*). Tagin avulla voidaan siis yhdistää fyysinen esine ja siihen liittyvä digitaalisessa muo-

[6] Katso esimerkiksi http://www.suunto.com/.

dossa oleva tieto tai toiminto siten, että kun käyttäjä lukee mobiililaitteellaan esineessä olevan tagin, sen sisältö aktivoituu ja käyttäjä saa esimerkiksi esineeseen liittyvää tietoa.

Tagien yleisin käyttö on tällä hetkellä logistiikassa. Kuluttajalaitteisiin ollaan kuitenkin integroimassa tagilukijoita, koska tagit nopeuttavat vuorovaikutusta tai laitteiden välisen vuorovaikutuksen aloittamista moninkertaisesti tagittömiin ratkaisuihin verrattuna. Tagit mahdollistavat luonnollisemman vuorovaikutuksen jokapaikan tietotekniikan palveluiden kanssa kuin perinteisten syöttölaitteiden, kuten näppäimistön käyttäminen. Yksi tärkeimmistä vaikuttavista tekijöistä on mahdollisuus maksaa pieniä maksuja koskettamalla maksutagia.

Tagi voidaan toteuttaa monella erilaisella teknologialla. RFID on yleinen ja halpa sähköisten tagien toteutustekniikka. Erityisesti passiiviset RFID-tagit ovat nousemassa merkittävään asemaan, koska ne saavat kaiken tarvitsemansa energian lukijalaitteesta eivätkä siten tarvitse erillistä paristoa. Toinen merkittävä teknologia ovat erilaiset optiset tagit, joita esimerkiksi viivakoodit ovat. Optinen tagi on kameralla (tai muulla optisella laitteella) luettava tunniste, jonka merkittävä etu on se, että tagi voidaan tulostaa paperille tai jopa esittää näytöllä. Viivakoodin tietosisältö on hyvin pieni, mutta esimerkiksi kaksiulotteiset matriisikoodit voivat sisältää riittävästi tietoa vaikkapa URL:n tallettamiseksi. Tagi voidaan toki toteuttaa myös monimutkaisemmilla teknologioilla, kuten Bluetooth-laitteella.

2.2. Ulkoiset laitteet

Tagien lisäksi mobiililaitteet voivat kommunikoida monien muiden lähiympäristön tai pidemmän kantaman laitteiden kanssa. Yleisiä jokapaikan tietotekniikan toteutustapoja ovat muun muassa erilaiset infrapuna- ja Bluetooth-laitteisiin perustuvat palvelut, jotka tarjoavat tyypillisesti lähiympäristöönsä liittyviä toimintoja. Nämä laitteet voivat toimia myös majakoina, eli *beaconeina*, jolloin ne lähettävät aktiivisesti sisältämäänsä tietoa (tai palvelutarjousta) käyttäjän päätelaitteelle.

Mobiililaitteisiin on mahdollista saada lisäksi tietoa pidemmän kantaman ulkoisista laitteista. Matkapuhelimia voidaan paikantaa niiden käyttämien tukiasemien perusteella ja myös langattomissa lähiverkoissa liikkuvat mobiililaitteet

voidaan paikantaa. Lisäksi puhelimeen voidaan liittää GPS-paikannuslaite[7], joka myös mahdollistaa paikkaan sidottuja palveluita. Mobiililaitteisiin on siis tarjolla syöttölaitteita, jotka hyödyntävät niiden liikkuvuutta ja osaltaan paikkaavat niiden pöytäkoneita heikompia perinteisiä syöttöominaisuuksia.

2.3. Anturit

Kolmas jokapaikan tietotekniikalle tyypillinen laitetyyppi ovat erilaiset anturit, joita voi olla paitsi ympäristössä niin myös itse päätelaiteessa. Erityisesti suoraan päätelaitteeseen liitettyjä antureita voidaan käyttää perinteisten syöttölaitteiden lisänä ja tukena. Esimerkiksi kiihtyvyysanturit mahdollistavat eleiden käyttämisen syötteenä. Käyttöliittymän lisäksi antureilla on käyttöä sellaisenaankin, esimerkiksi huoltomies voi lukea päätelaitteellaan seinän sisään sijoitetun RFID-tagiin liitetyn kosteusanturin.

3. CoolTown

WWW on ollut pitkään pelkästään virtuaalinen tila, jonka osilla eli WWW-sivuilla yms. on ollut hyvin vähän tekemistä fyysisen maailman kanssa [Kindberg *et al*. 2002]. Tietokannoissa ja WWW:ssä on kuitenkin valtavat määrät fyysiseen maailmaan liittyvää tietoa, mutta tämä tieto on näiden kahden maailman erillisyyden takia ollut myös erillään fyysisistä vastineistaan. Kindbergin *et al*. mukaan molemmat maailmat olisivat rikkaampia jos ne olisi sidottu tiukemmin yhteen.

3.1. Web Presence

Kindbergin *et al*. [2002] CoolTown perustuu *Web presence* -käsitteeseen. *Web presence* on ihmisten, paikkojen tai esineiden esiintymä webissä, teknisesti siis kyseistä entiteettiä edustaa WWW:ssä jokin resurssi (yleensä WWW-sivu), jolla on osoite (URL) ja johon pääsee käsiksi tavallisella HTTP-protokollalla (*Hypertext Transfer Protocol*).

CoolTown on tavallaan *middleware*-komponentti web-sisältöjen saamiseksi kannettaviin laitteisiin erilaisten tagien, paikkatietojen yms. avulla. Kindberg *et al.* eivät varsinaisesti kuvaa käyttöliittymän ohjelmistoarkkitehtuuria vaan koko järjestelmän arkkitehtuuria. CoolTownissa (kuten muissakin jokapaikan tietotekniikan järjestelmissä) käyttöliittymän raja on kuitenkin hyvin hämärä ja

[7] Esimerkiksi http://www.nokia.fi/puhelimet/lisalaitteet/xpressongps/index.html.

koko *Web presence* voidaan nähdä käyttöliittymänä ja rajapintana fyysisen ja digitaalisen maailman välillä.

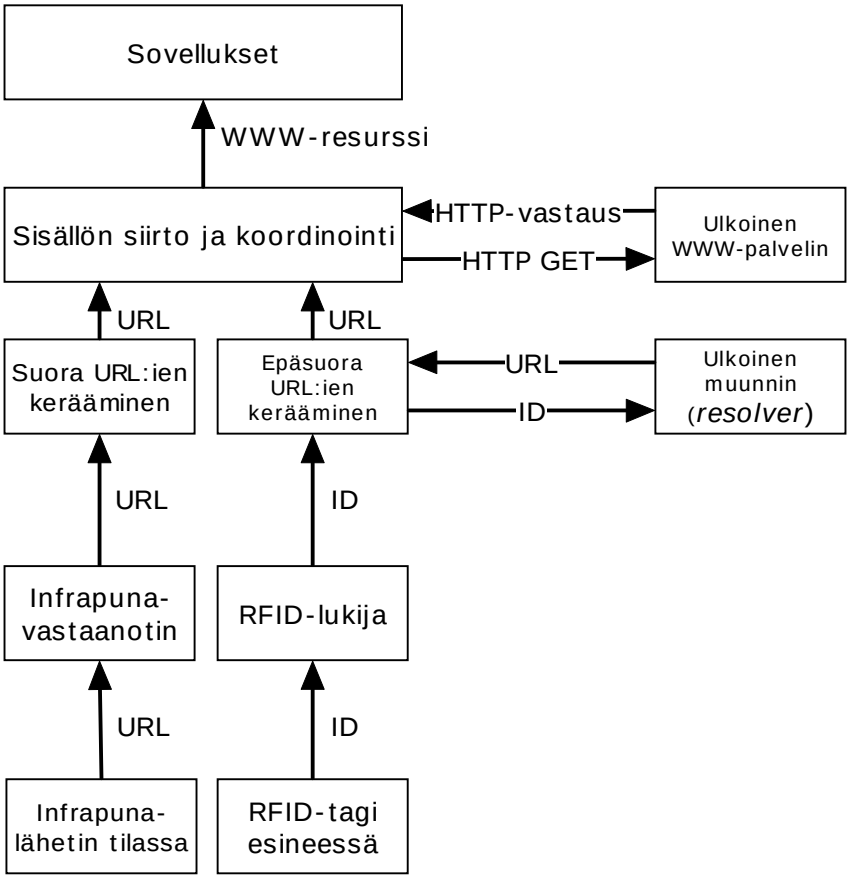
Web presence tarkoittaa kuitenkin muutakin kuin WWW-sivua. Kindberg *et al.* ovat määritelleet kolme muuta tekijää:

1. Järjestelmä itse tukee fyysisen ja digitaalisen maailman sitomista toisiinsa. Tämä voi tapahtua esimerkiksi automaattisesti paikkoihin liittyvien linkkien keräämisellä kun käyttäjä saapuu kyseiseen paikkaan.
2. Esineiden kontrollointi niiden *Web presence*:n kautta. Esimerkiksi talon valaistusta voi ohjata webin yli jos talolla/valoilla on tätä tukeva *Web presence*.
3. Esineet, jotka siihen kykenevät, voivat vaihtaa web-sisältöä keskenään ilman ihmiskäyttäjää.

CoolTownin käyttäjäkokemus perustuu suurelta osin linkkien keräämiseen mobiililaitteeseen itse esineistä. Lisäksi, joissain tapauksissa langaton verkko, jossa käyttäjä liikkuu, pystyy tarjoamaan muiden siihen liittyneiden kohteiden URL:t. CoolTownissa käytetään sekä aktiivista että passiivista linkkien keräämistä. Esineisiin voidaan liittää aktiivisia infrapunamajakoita (*beaconeita*) jotka lähettävät tietojaan koko ajan ympäristöön tai passiivisia RFID-tageja, jotka käyttäjän on itse luettava. Tätä eroa voidaan myös hyödyntää: jos tarjolla oleva linkki on yleisesti hyödyllinen sen paikassa, se voidaan lähettää käyttäjän päätelaitteeseen, mutta jos se liittyy rajatumpaan kontekstiin, voi olla parempi että käyttäjä lukee sen itse.

3.2. Arkkitehtuuri

CoolTownin kuvattu arkkitehtuuri käsittelee koko infrastruktuuria, jolla fyysiseen esineeseen tms. liittyvä WWW-sivu saadaan päätelaitteeseen. WWW on valittu toteutustekniikaksi, koska se on laajalle levinnyt, sille on olemassa standardit, se tukee hyvin kerroksittaisia sovelluksia ja sen käyttöliittymä (WWW-selain) on jo entuudestaan käyttäjille tuttu. *Web presence* -infrastruktuurin tehtäviksi jäävät lisätä WWW:hen *presenceä* tukevat kerrokset.



Kuva 1. CoolTownin arkkitehtuuri esineeseen tai paikkaan sidotun WWW-resurssin esittämiseksi.

CoolTownin arkkitehtuurin (Kuva 1) alimmalla tasolla ovat *web presence* -pisteiden löytämiseen tähtäävät toiminnot, eli esineisiin ja paikkoihin liittyvien URL:ien kerääminen. Jotta käyttö olisi helppoa ja vaivatonta, URL:t on saatava automaattisesti päätelaitteeseen. Tämä tapahtuu siihen liitetyillä ”antureilla”, jotka itse asiassa ovat RFID-lukija ja infrapunaportti, jonka avulla tiloissa olevat infrapunamajakat voivat lähettää tietoa PDA:han. URL:ien kerääminen voi olla joko suoraa tai epäsuoraa. Suorassa esine pystyy välittämään URL:n, joka johtaa sen *Web presenceen* kun taas epäsuorassa esine välittää jonkin tiedon (esimerkiksi tunnisteen tai koordinaatit). Tämän tiedon avulla ja mahdollisesti käyttäjän konteksti huomioon ottaen määritetään ulkoisella palvelimella varsinainen URL.

CoolTownin arkkitehtuurin toisella tasolla ovat palvelut, joilla siirretään sisältöä eri entiteettien välillä. Suurin osa CoolTownin tietoliikenteestä käyttää tavallista HTTP-protokollaa paitsi tavallisten WWW-sivujen siirtämiseen niin

myös esimerkiksi laitteiden kontrollointiin. Kindberg *et al*. ovat myös luoneet *eSquirt*-protokollan, jonka avulla esineet voivat kommunikoida suoraan lähietäisyydeltä. Käyttäjä voi esimerkiksi ”*eSquirtata*” PDA:sta WWW-sivun projektoriin osoittamalla sitä, jolloin projektori esittää kyseisen WWW-sivun. Tässä operaatiossa siirretään laitteiden välillä vain esitettävän resurssin URL.

Arkkitehtuurin ylimmällä tasolla ovat itse sovellukset ja infrastruktuuri, jolla voidaan tarjota erilaisia paikkoihin liittyviä palveluita ja tukea liikkuvaa käyttäjää.

4. Matkapuhelin, Bluetooth-laitteet ja RFID-tagit

Siegemund ja Flörkemeier [2003] kuvaavat matkapuhelimen, Bluetooth-laitteiden (*BTnode* eli Bluetooth-node) ja RFID-tagien avulla toteutetun jokapaikan tietotekniikan arkkitehtuurin ja sen päälle rakennettuja sovelluksia.

4.1. Sovellukset

Siegemund ja Flörkemeier ovat toteuttaneet kolme sovellusta esittelemään heidän vuorovaikutustapojaan. Ensimmäisessä sovelluksessa kananmunakennoon on lisätty antureita havaitsemaan jos se putoaa tai ei ole asetettujen lämpötilarajojen sisällä. Jos tällainen tapahtuma havaitaan, kenno lähettää SMS-viestin. Kennoon on liitetty BTnode, jossa on kiihtyvyys- ja lämpötila-anturit ja tämä node lähettää tarvittaessa viestin Bluetooth access pointin kautta. Käyttäjälle (kontaktihenkilölle) lähetettävässä viestissä on myös mahdollisuus vastauksen lähettämiseen kennolle SMS-viestinä, johon kenno voi taas puolestaan vastata käyttäjälle. Käyttäjien matkapuhelimiin on lisäksi liitetty RFID-tagi, jonka kennon BTnodeen langattomasti yhdistetty RFID-lukija pystyy lukemaan matkan päästä ja node saa näin tagin kautta selville kuka sopivista käyttäjistä on lähistöllä ja mikä on hänen puhelinnumeronsa. Sovelluksen tarkoituksena on havainnollistaa augmentoidun esineen aloittamaa vuorovaikutusta.

Toisessa sovelluksessa Siegemund ja Flörkemeier esittelevät käyttäjän aloittamaa vuorovaikutusta. Tässä sovelluksessa erilaisia esineitä on augmentoitu BTnodeilla, joilla jokaisella on oma puhelinnumero ja joita voi ohjata tekstiviesteillä. Jotta käyttäjän ei tarvitsisi itse näppäillä puhelinnumeroita ja komentoja, nodet lähettävät nämä tiedot itse matkapuhelimeen. Koska nodeja voi olla ympäristössä paljon, kaikkia niitä ei kuitenkaan voi esittää käyttäjän matkapuhelimessa. Tämän takia käyttäjän kontekstista kerättyä tietoa käytetään päättämään, mitkä esineistä ovat relevantteja käyttäjälle.

Kolmas sovellus on lääkekaappi, joka auttaa käyttäjää lääkkeiden hallinnassa. Kaappi pystyy muistuttamaan lääkkeen ottamisesta, sen sisältöä voi tarkastella etäyhteyden kautta ja se kykenee varoittamaan käyttäjää vanhentuneista lääkkeistä. Lääkepakkaukset on varustettu RFID-tageilla ja kaapissa on RFID-lukija, joka on yhteydessä kaapin Bluetooth-nodeen. Kaappi kommunikoi Bluetoothin välityksellä käyttäjän matkapuhelimen kanssa kun tämä on riittävän lähellä. Erikoista tässä sovelluksessa on, että käyttäjän puhelinta käytetään tiedonsiirtoon kaapin ja ulkoisten palvelimien välillä; kaappi käyttää siis puhelinta ikään kuin parasiittisesti.

4.2. Arkkitehtuuri

Siegemundin ja Flörkemeierin arkkitehtuurissa matkapuhelimella on kolme roolia:

1. Bluetooth-nodet käyttävät matkapuhelinta infrastruktuurin kanssa kommunikoimiseen. Matkapuhelin toimii porttina GSM-verkkoon ja sitä kautta nodejen käyttämille ulkoisille palvelimille. Tämä rooli ei näy käyttäjälle.
2. Augmentoidut esineet ottavat yhteyttä matkapuhelimeen Bluetoothin avulla ja esittävät käyttäjälle hälytyksiä, viestejä ja ilmoittavat myös olemassaolostaan ja tarjoamistaan palveluista.
3. Käyttäjä voi ottaa puhelimella Bluetoothin yli yhteyden nodeen ja lähettää sille komentoja tai kysellä siltä tietoja. Vuorovaikutukseen käytetään *interaction stub*:eja, joita nodet lähettävät puhelimille ja jotka sisältävät nodejen yhteystiedot ja niiden tukemat komennot. Nämä stubit sisältävät tarvittaessa noden etäyhteystiedot, jos siihen saa yhteyden esimerkiksi SMS-viestin avulla Bluetoothin kantaman ulkopuolelta.

Kontekstitietoa käytetään määrittämään, mitä palveluita käyttäjälle esitetään, eli mitkä *interaction stub*:it näytetään matkapuhelimen näytöllä. Bluetooth-nodet voivat lisäksi olla RFID-lukijan kautta yhteydessä RFID-tageilla varustettuihin esineisiin. Puhelin ei siis suoraan kommunikoi tagitettujen esineiden kanssa.

Lääkekaappisovelluksen arkkitehtuurissa oleellista ovat 1) käyttäjän päätelaitteen toimiminen paitsi käyttöliittymänä, myös access pointtina ja 2) augmentoitujen esineiden digitaalisten vastikkeiden (*data shadow*) löytyminen paitsi

fyysisestä ympäristöstä, myös verkossa olevalta palvelimelta, jolloin ne ovat aina käytettävissä riippumatta käyttäjän sijainnista.

5. MSE – Mobile Service Explorer

Toye *et al*. [2004] ovat kehittäneet optisiin tageihin perustuvan vuorovaikutustekniikan ja sen toteuttavan asiakas/palvelin -ohjelmiston paikallisten mobiilipalveluiden kehittämiseen matkapuhelimille. Heidän vuorovaikutustekniikkansa perustuu matriisikoodien lukemiseen matkapuhelimen kameralla ja tagien perusteella luotaviin Bluetooth-yhteyksiin. CoolTownissa fyysisiä hyperlinkkejä käytettiin assosioimaan WWW-sivuja fyysisten esineiden kanssa. Toyen *et al*. työssä tageja käytetään myös luomaan yhteyksiä paikallisiin Bluetooth-palveluihin.

5.1. Vuorovaikutustekniikka

Kun käyttäjä näkee matriisikoodin ympäristössä, hän voi käynnistää kamerapuhelimessaan *Mobile Service Explorer* (MSE) -sovelluksen. MSE esittää puhelimen näytöllä kameran kuvaa ja kun puhelin havaitsee tagin, se korostaa sen näytöllä. Kun tagi on korostettuna, sen voi valita painamalla puhelimen valintapainiketta.

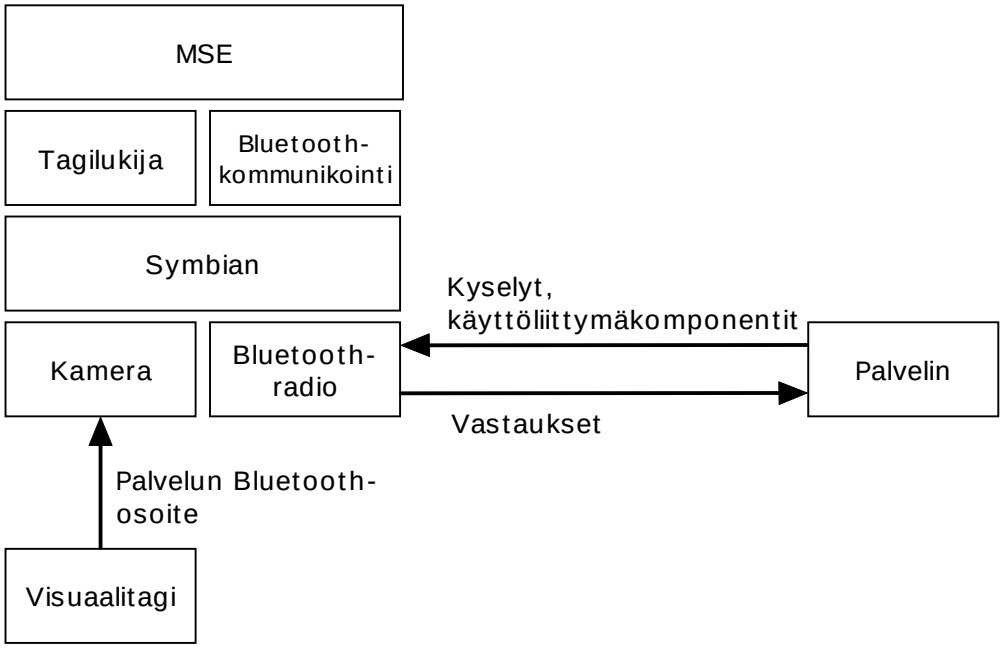
Käyttäjä on vuorovaikutuksessa mobiilipalvelun kanssa ensisijaisesti osoittamalla ja valitsemalla tageja. Tagin valinta aiheuttaa joko tiedon esittämisen puhelimessa tai jonkin toiminnon käytettävässä palvelussa. Paikallinen palvelu ja sen ominaisuudet määrittävät, mitä vasteita ja miten käyttäjä saa. Yhdessä esimerkkisovelluksessa osa paikallista palvelua on julkinen näyttö, jolloin osa vasteista esitetään puhelimen omalla ja osa julkisella näytöllä.

Tagien ja suurten julkisten näyttöjen käyttö auttaa käyttäjää tulemaan toimeen matkapuhelimen pienen näytön ja näppäimistön kanssa. Samalla palveluita voi personoida automaattisesti lukemalla käyttäjätietoja suoraan puhelimesta sen sijaan että käyttäjän tarvitsisi itse syöttää niitä. Palveluiden käytössä muodostuva tieto voidaan myös tallettaa heti puhelimeen ilman käyttäjältä vaadittavia toimia.

5.2. Arkkitehtuuri

Toye et al. [2004] ovat kehittäneet matriisikoodeihin perustuvan asiakas/palvelin -ohjelmiston paikallisten mobiilipalveluiden kehittämiseen matka-

puhelimille. Palvelinohjelmisto käyttää matkapuhelimen kanssa kommunikointiin Bluetoothia, jonka lyhyt kantama tekee palveluista paikallisia.



Kuva 2. Palveluyhteyden muodostavat komponentit MSE:ssä.

Asiakasohjelmisto (*client*) on MSE ja sitä ajetaan käyttäjän matkapuhelimessa. MSE toimii Symbian Series 60 -alustalla natiivina C++-sovelluksena. Käyttäjä ei tarvitse MSE:n lisäksi muita ohjelmistoja puhelimeensa; kaikki sovelluskohtaiset toiminnot tulevat palvelimelta (Kuva 2). Tarkoituksena on samalla helpottaa sovelluskehitystä, koska näin kehittäjän ei tarvitse hallita tunnetusti monimutkaista Symbian-ympäristöä. MSE:llä on kolme perustoimintoa:

1. tagien lukeminen,
2. palveluiden personointi ottaen huomioon käyttäjän yksityisyysasetukset, ja
3. käyttöliittymäelementtien renderöinti puhelimen näytölle ja käyttäjän syötteiden lähettäminen palvelimelle.

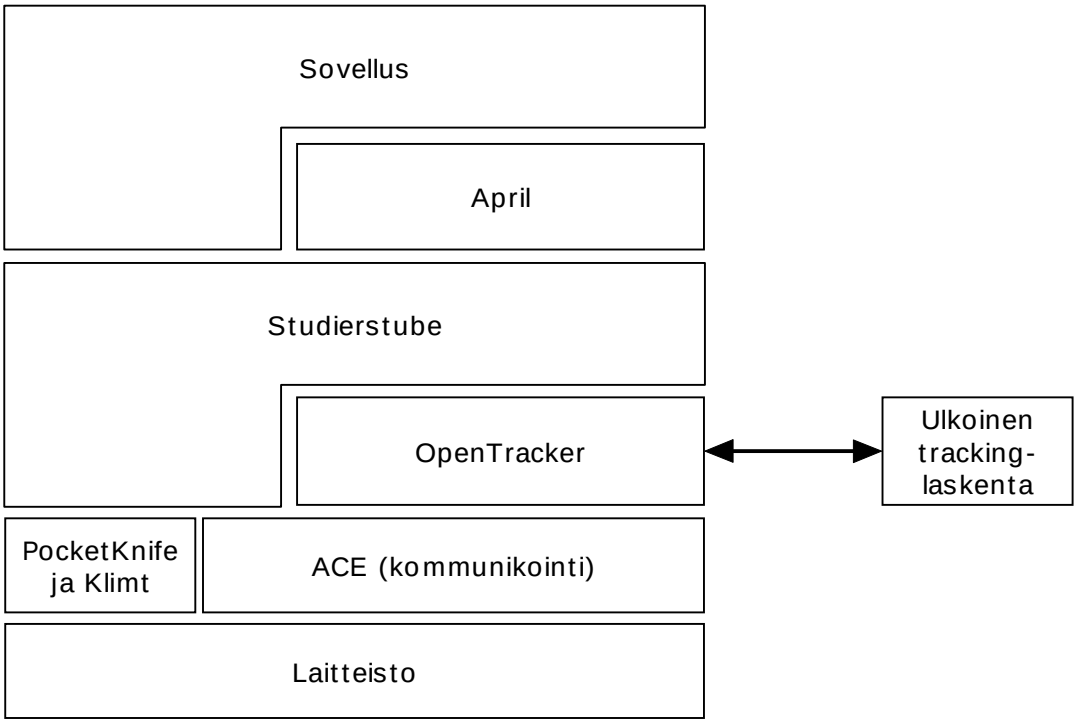
Tagit sisältävät paikallisen palvelun Bluetooth-osoitteen (*Bluetooth Device Address*) ja mahdollisesti sovelluskohtaista tietoa. MSE lukee ja tulkitsee tagilukija-moduulin avulla palvelun Bluetooth-osoitteen ja ottaa yhteyttä tagin määrittelemään osoitteeseen. Kun MSE on luonut yhteyden palveluun, se vastaa palvelun lähettämiin kutsuihin. Kutsuja on neljänlaisia:

1. Palauta viimeisimmät tagit (mukaan lukien niiden orientaatio).
2. Palauta viimeisimmät näppäinten painallukset.
3. Esitä käyttöliittymäkomponentti puhelimen näytöllä ja palauta käyttäjän antama vaste.
4. Palauta tieto käyttäjästä (esimerkiksi puhelinnumero).

MSE pitää kirjaa tietyistä käyttäjän tiedoista, kuten puhelinnumerosta ja sähköpostiosoitteesta sekä käyttäjän yksityisyysasetuksista. Yksityisyysasetukset määräävät, mitä tietoja MSE saa tai ei saa lähettää palvelulle ja mihin sen on kysyttävä erikseen lupa käyttäjältä.

6. Lisätty todellisuus

Wagner *et al*. [2005] ovat luoneet lisätyn todellisuuden sovelluksia (*Augmented Reality, AR*) varten PDA-laitteille arkkitehtuurin, joka, perustuu päätelaitteeseen liitettyyn kameraan ja siitä saadun kuvan augmentointiin. Arkkitehtuurin tarkoituksena on tukea vuorovaikutteisten, monen käyttäjän ja infrastruktuurista riippumattomien sovellusten luomista. Laitteistoalustana ovat kaupallisesti saatavilla olevat PDA:t. Lisätyn todellisuuden sovellukset ovat olleet useimmiten silmikkonäyttöihin (*Head-Mounted Display, HMD*) perustuvia, mutta tämä arkkitehtuuri perustuu kameralla varustettuun PDA-laitteeseen. PDA:n kamera kuvaa laitteen edessä näkyvää ympäristöä ja lisää näytölle ympäristön esineisiin liittyvää grafiikkaa, jolloin käyttäjä ikään kuin katsoo PDA:n näytön läpi ympäristöä ja siten näkee lisätyt ominaisuudet suoraan ympäristössä. Aikaisemmat HMD-näyttöihin perustuvat sovellukset ovat olleet myös riippuvaisia juuri niitä varten luoduista infrastruktuureista. Wagnerin *et al.* mukaan on olemassa tarve infrastruktuuririippumattomalle arkkitehtuurille, joka ei vaatisi käyttäjältä tavallista halpaa PDA:ta erikoisempia laitteita. Wagnerin *et al.* arkkitehtuuri perustuu Studierstube-komponenttiin, joka on sovitettu toimimaan PDA:ssa karsimalla siitä tarpeettomat osat pois (Kuva 3).



Kuva 3. Lisätyn todellisuuden arkkitehtuuri

Asennon seuraaminen (*tracking*) on yksi tärkeimmistä toiminnoista AR-sovelluksissa. Studierstube käyttää OpenTracker -middleware-komponenttia PDA:n asennon havainnoimiseen. OpenTracker pystyy toimimaan hajautettuna komponenttina, eli se voi suorittaa tarvittavan laskennan itse tai käyttämään laskentaan verkosta löytyviä palveluita jos PDA:n oma prosessoriteho ei riitä käytettävälle sovellukselle. OpenTracker pystyy käyttämään useimpia *tracking*-laitteita (myös verkon yli) ja se tarjoaa Studierstubelle yhtenäisen rajapinnan niiden käsittelyyn.

Sekä Studierstube että OpenTracker käyttävät ACE-komponenttia (*Adaptive Communication Environment*) tietoliikenteen käsittelyyn. Koska Wagnerin *et al*. järjestelmä on tarkoitettu hajautetuksi järjestelmäksi, verkkoyhteydet ovat erityisen tärkeitä sille. ACE tarjoaa Studierstubelle ja OpenTrackerille laitteistoalustasta riippumattomat verkkopalvelut ja pääsyn käyttöjärjestelmän palveluihin, kuten säikeisiin ja jaettuun muistiin.

Studierstuben päällä voidaan käyttää vielä APRIL-komponenttia, joka tarjoaa yksinkertaistetun korkean tason kielen AR-sovellusohjelmointiin. Wagner *et al.* kehittivät lisäksi kaksi omaa komponenttia, jotka ovat PocketKnife ja 3D-grafiikkarenderöijä Klimt. PocketKnife on kokoelma luokkia, jotka abstrahoivat

käyttöliittymälaitteet erityisesti graafisen käyttöliittymän helppoa toteutusta varten. Erityistä Wagnerin *et al.* arkkitehtuurissa on, että kaikki komponentit ovat vapaasti saatavilla olevia avoimen lähdekoodin ohjelmistoja, jotka he ovat yhdistäneet yhdeksi kokonaisuudeksi.

7. MIMOSA

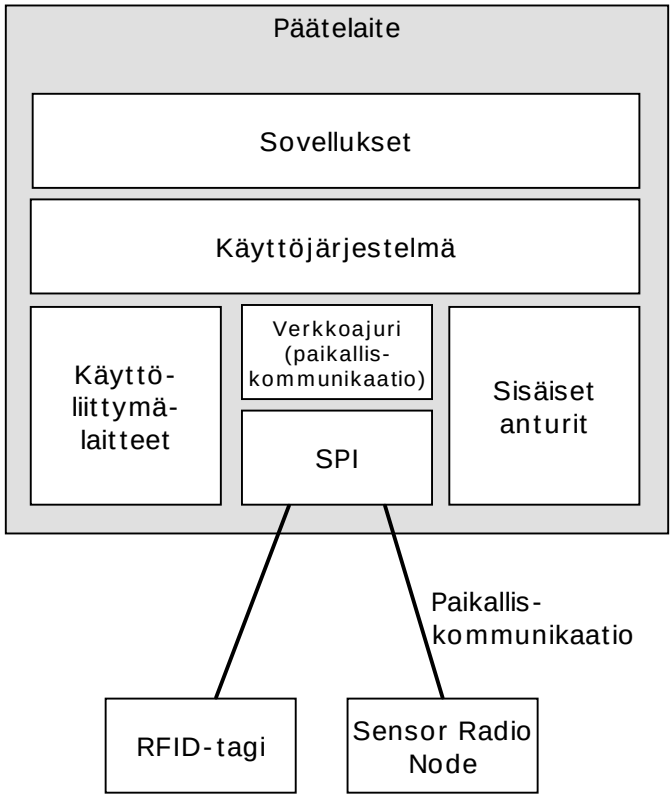
OMAS (*Overall MIMOSA architecture specification* [Lappeteläinen *et al.*, 2004]) määrittelee avoimen rajapinnan jokapaikan tietotekniikan demonstraatioiden ja prototyyppien toteuttamista varten. OMAS:n tarkoituksena on tarjota arkkitehtuuri, johon voi liittää uusia paikalliskommunikaatio- (*local connectivity*), anturi- ja käyttöliittymälaitteita. Arkkitehtuuri on tarkoitettu nimenomaan demonstraatioiden toteuttamiseen eikä tuotantoalustaksi.

MIMOSA:ssa matkapuhelin on henkilökohtainen käyttöliittymä jokapaikan tietotekniikkaan. Se toimii myös porttina antureiden, anturiverkkojen, julkisten verkkojen ja Internetin välillä. MIMOSA-arkkitehtuurissa on neljä laitetyyppiä:

1. päätelaite (matkapuhelin),
2. Sensor Radio Node (Bluetooth-laite),
3. aktiivinen RFID-tagi (anturilla tai ilman), ja
4. passiivinen RFID-tagi (anturilla tai ilman).

Kaikilla ulkoisilla laitteilla (2-4) on paikalliskommunikaatioyhteydet, joiden avulla ne liittyvät päätelaitteeseen.

Päätelaitteen arkkitehtuurin tärkeimmät osat ovat paikalliskommunikaatiomoduuli, käyttöliittymälaitteet ja suoraan päätelaitteessa olevat anturit. Käyttöjärjestelmä (Linux tai Symbian) tarjoavat sovelluksille ohjelmointirajapinnan näiden osien käyttämistä varten.



Kuva 4. MIMOSA-arkkitehtuurin tärkeimmät osat.

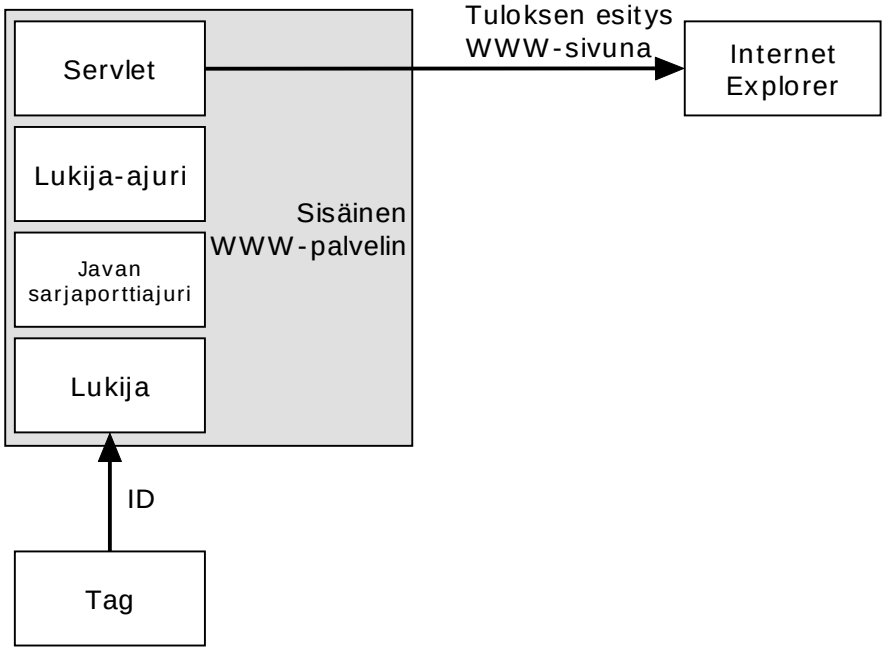
Paikalliskommunikaatiomoduuli jakaantuu kahteen osaan. Alimmalla tasolla on SPI-ajuri (*Serial Peripheral Interface*) ja sen päällä verkkoajuri (*Network driver*). SPI tarjoaa yksinkertaisia datan luku- ja kirjoituspalveluita verkkoajurille, jota taas sovellukset käyttävät. Anturirajapinta on samankaltainen: alimpana on yksinkertaisia luku- ja kirjoitusoperaatioita tarjoava taso ja sen päällä yleisempi laiteajuri sovellusohjelmointia varten.

Sovelluksia varten OMAS:n päätelaitteessa on sovellusrajapinta (*application interface*). Se tarjoaa sovelluksille paikalliskommunikaation ja antureiden käytön. Paikalliskommunikaatioon kuuluu muiden laitteiden (antureiden ja muiden päätelaitteiden) löytäminen, yhteyksien luominen ja sulkeminen. Anturimoduuli tarjoaa antureiden löytämisen, konfiguroinnin ja niiden lukemisen sekä puhelimeen liitetyille että ulkoisille antureille.

8. PointMe, TouchMe, ScanMe

Olemme toteuttaneet prototyypin, joka emuloi passiivisia etäluettavia RFID-tageja [Tuomisto *et al.* 2005]. PDA-laitteeseen liitetyllä prototyypillä voi lukea ta-

geja osoittamalla (PointMe) tai koskettamalla (TouchMe), tai lukea koko ympäristön tagit kerralla (ScanMe) ja valita sitten haluamansa tagin PDA:n näytöltä. Prototyypin tarkoituksena on demonstroida ja tutkia vuorovaikutusmahdollisuuksia uusien etäluettavien RFID-tagien kanssa. PDA:han on liitetty lukijamoduuli, joka on yhteydessä tageihin. Lukijassa on lisäksi infrapuna- ja laserosoittimet tagin osoittamista varten ja tageissa anturit, jotka havaitsevat osoitussäteen.



Kuva 5. Tagilukija-servlet

Päätelaitteessa on sekä WWW-selain että -palvelin ja sen toiminnallisuus on toteutettu sisäisessä WWW-palvelimessa ajettavana Java-servlettinä. PDA:lla ajettava servlet kommunikoi lukijalaitteen kanssa lukija-ajurin ja sarjaporttiajurin kautta. Kun käyttäjä painaa lukijalaitteessa olevaa nappia ja lukija saa lukemaltaan tagilta vastauksen, se lähettää viestin PDA:lle. Jos luvun tuloksena oli yksi linkki, servlet avaa käyttöjärjestelmän palveluiden avulla WWW-selaimen tagista luettuun osoitteeseen. Jos tuloksena oli useampi linkki (esimerkiksi osoitussäde osui kahteen tagiin yhtä aikaa) tai käyttäjä luki koko ympäristöä, servlet avaa selaimeen HTML-sivun, jossa löydetyt linkit näkyvät listana ja josta käyttäjä voi jatkaa selailua tai yrittää uutta osoitusta tai kosketusta. Tähän tarkoitukseen haluttiin käyttää muuta menetelmää kuin modaalista dialogia, joka pakottaisi käyttäjän vastaamaan dialogin kysymykseen sen sijasta että käyttäjä voisi

halutessaan yrittää tarkempaa osoitusta tai kosketusta ilman graafisen käyttöliittymän käyttöä. Tavoitteena on ollut vähentää muiden vuorovaikutustapojen käyttö minimiin, jotta käyttäjää ei kahlittaisi PDA:n rajallisilla syöttölaitteilla.

Selain pystyy avaamaan myös muita tiedostotyyppejä kuin HTML-sivuja ja HTTP:tä. Jos linkki osoittaa esimerkiksi videotiedostoon, se avataan mediasoittimessa tai jos HTTP:n (http://osoite) sijasta linkissä on mailto-schemellä sähköpostiosoite (mailto:osoite), käynnistetään sähköpostiohjelma.

9. Tag Manager

Tag Manager on Symbian-matkapuhelimiin tarkoitettu middleware-komponentti, joka tukee erilaisia tagiteknologioita ja tageja hyödyntäviä sovelluksia. Keräsen *et al*. [2005] mukaan käyttäjät haluavat personoida tapaa, jolla heidän puhelimensa reagoi tietyn tagin lukemiseen ja että reaktio tagin lukemiseen voidaan tehdä myös kontekstista riippuvaksi. Tätä varten on luotava ohjelmistoarkkitehtuuri, joka mahdollistaa tageihin perustuvien palveluiden käyttämisen. Tag Manager tarjoaa sovelluksille myös ohjelmointirajapinnan lukijalaitteisiin.

9.1. Vaatimuksia arkkitehtuurille

Keränen *et al*. ovat toteuttaneet useita erityyppisiä tageihin perustuvia sovelluksia matkapuhelimelle. Näistä he ovat tunnistaneet neljä sovellusluokkaa:

1. tietosovellukset,
2. viestintäsovellukset,
3. laitteiden välinen kommunikointi, ja
4. maksusovellukset.

Analysoimalla näitä ja kirjallisuudesta löytyviä sovelluksia, he ovat päätyneet seuraaviin vaatimuksiin tagisovelluksia tukevalle arkkitehtuurille:

1. Samassa tagissa on voitava olla useita palveluita toisistaan riippumattomista lähteistä.
2. Arkkitehtuurin on tuettava erilaisia teknologioita, kuten optisia ja RFID-tageja.
3. Arkkitehtuurin on tuettava parasiittisia palveluita, joissa sovellus käyttää tagia johonkin muuhun kuin mihin se on alun perin tarkoi-

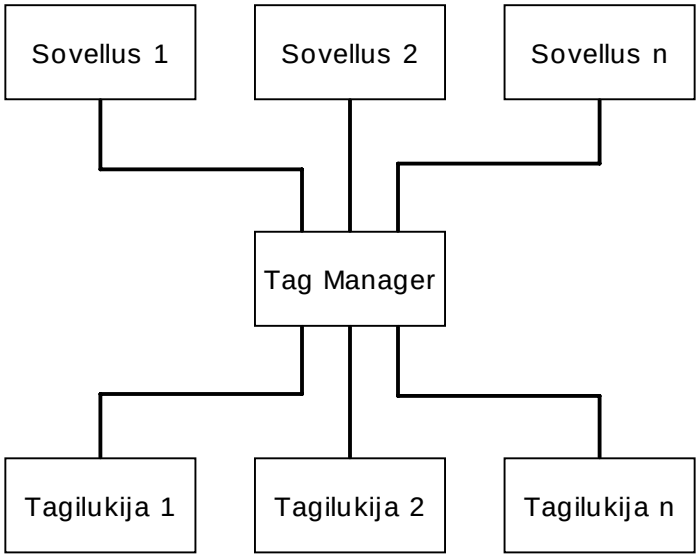
tettu. Esimerkkinä kirjoittajat mainitsevat Skannerz-pelin, joka luo viivakoodeista pelihahmoja.

4. Tagin valinnan on otettava huomioon aktiivisena oleva sovellus.
5. Arkkitehtuuri ei saa rajoittaa sovellusten käyttöliittymäsuunnittelua.
6. Tagit on voitava tulkita (*resolve*) paikallisesti.
7. Arkkitehtuurin on tuettava jo olemassa olevia infrastruktuureita.

Tag Managerin avulla on mahdollista käyttää olemassa olevaa infrastruktuuria tagin kanssa, esimerkiksi kirjan ISBN-numeroa koodattuna kirjassa olevaan viivakoodiin voidaan hyödyntää palveluissa, jotka käyttävät ISBN-numeroa (esimerkiksi kirjan haku kirjaston tietokannasta). Kyseinen viivakoodi kuitenkin identifioi kirjan yleisesti ja sitä voidaan tarvittaessa käyttää muissakin sovelluksissa.

9.2. Arkkitehtuuri

Tag Manager on toteutettu siten, että palvelut (sovellukset) ja tagilukijat liittyvät Tag Manageriin käyttäen käyttöjärjestelmän tarjoamaa prosessien välistä kommunikointia (IPC) Tag Managerin kanssa kommunikointiin (Kuva 6).



Kuva 6. Tag Manageriin voi liittyä useita sovelluksia tagien vastaanottajiksi ja useita lukijoita tagitiedon tuottajiksi.

Tagilukijat syöttävät tagin sisällön Tag Managerille aina kun tagi luetaan. Kun Tag Manager vastaanottaa tagin, se kysyy jokaiselta siihen liittyneeltä palvelulta, mikä kyseisen palvelun prioriteetti kyseiselle tagille on. Prioriteetit ovat:

1. *Cannot consume* (tagi ei ole relevantti palvelulle).
2. *Ready to consume* (tagi on relevantti palvelulle, mutta palvelu ei ole aktiivisena).
3. *Active consumption* (tagi on relevantti palvelulle ja palvelu on aktiivisena).

Aktiivisuus tarkoittaa tässä sitä, että kyseinen sovellus on näkyvissä käyttäjälle. Symbian Series 60:ssa vain yksi sovellus kerrallaan on näkyvissä eli aktiivisena. Tag Manager valitsee tagin vastaanottajaksi palvelun, joka ilmoittaa tagille suurimman prioriteetin ja jos saman prioriteetin palveluita on useita, se kysyy käyttäjältä, mille palvelulle tagi annetaan. Kaikki Tag Manageriin liittyneet palvelut saavat luetun tagin sisällön tutkittavakseen, eli jokaisen palvelun on itse tutkittava, onko kyseinen tagi relevantti sille.

10. Yhteenveto

Olen esitellyt tässä tutkielmassa seitsemän erilaista jokapaikan tietotekniikan mobiililaitekeskeistä arkkitehtuuria. Hewlett-Packardin laboratorioiden CoolTown [Kindberg *et al*., 2002] on yksi ensimmäisistä laajoista järjestelmistä ja yksi merkittävimmistä. Siegemund ja Flörkemeier [2003] ovat luoneet RFID-tagien Bluetooth-laitteiden ja matkapuhelinten avulla ennen kaikkea vuorovaikutustapoja mutta myös arkkitehtuurin kontekstitietoisten palveluiden toteuttamista varten. Esimerkkinä uusista matriisikoodeihin ja kameroilla varustettuihin matkapuhelimiin perustuvista järjestelmistä on MSE [Toye *et al*., 2004]. Toinen optisiin tageihin perustuva järjestelmä [Wagner *et al*., 2005] esittää monimutkaisemman PDA-perustaisen järjestelmän, jolla voidaan luoda lisätyn todellisuuden sovelluksia. OMAS [Lappeteläinen *et al*., 2004] ja oma RFID-emulaattorimme [Tuomisto *et al*., 2005] ovat erityisesti RFID-tagien avulla tapahtuvan vuorovaikutuksen tutkimiseen tarkoitettuja järjestelmiä. OMAS pitää lisäksi sisällään mahdollisuuden liittää itse päätelaitteeseen erilaisia antureita, joiden lukemia voidaan käyttää syötteenä, esimerkiksi kiihtyvyysanturin lukemista voidaan tunnistaa päätelaitteella tehtyjä eleitä. Lopuksi esittelin Tag Manager -arkkitehtuurin [Keränen *et al*., 2005], joka on tarkoitettu yhdistämään samaan

päätelaitteeseen erilaisia tagitekniikoita. Edellä esitetyt arkkitehtuurit ovat hyvin tiukasti sidottuja myös niiden laitteistoarkkitehtuureihin, koska käyttöliittymä mobiililaitekeskeisyydestä huolimatta ei muodostu vain päätelaitteesta. Oleellisessa roolissa ovat myös erilaiset ympäristöön upotetut laitteet, joiden kanssa päätelaite kommunikoi, ja joiden kanssa myös käyttäjä on vuorovaikutuksessa, joko suoraan tai päätelaitteen kanssa.

Mobiililaite ikään kuin kaukosäätimenä lähiympäristön toimintoihin ei ole kuitenkaan aina optimaalinen ratkaisu. Käyttäjän olisi usein helpompaa käyttää suoraan omia eleitään ja oikeasti luonnollisia vuorovaikutustapoja tai parhaimmassa tapauksessa yksinkertaisesti vain toimia ympäristössä käyttäen sen esineitä tavalliseen tapaan. Tämä lähestymistapa toimii kuitenkin parhaiten ympäristöissä, jotka ovat vahvasti instrumentoituja erilaisilla antureilla, kameroilla jne. ja joissa on käytettävissä paljon tietojenkäsittelykapasiteettia. Esimerkiksi paineanturein varustettu lattia, kamera ja mikrofoni (mielellään vielä äänen suunnan paikantava) huoneen joka nurkassa ja paljon prosessoritehoa mahdollistavat monenlaisia sovelluksia joiden käyttö on helpompaa ja luonnollisempaa kuin huoneen toimintojen ohjailu matkapuhelimella. Tilanne kuitenkin muuttuu kun tietotekniikkaa halutaan upottaa kaikkialle (esimerkiksi ulkotiloihin) ja kun käytännölliset ja taloudelliset tekijät vaikuttavat asiaan. Tällöin (ainakin ennustettavissa olevassa tulevaisuudessa) verkkoa hyödyntävä ja käyttäjän itsensä mukana kulkeva kaukosäädin, joka pystyy toimimaan yksinkertaisessa yhteistyössä ympäristöön upotettujen pienten ja halpojen laitteiden kanssa, mahdollistaa laajalle levitettävän – käytännössä joka paikan – tietotekniikan. Tämä ei tietenkään sulje pois erityisten vahvasti tietoteknisten tilojen (esimerkiksi kokoushuoneet) mahdollisuutta.

Viiteluettelo

[Keränen *et al*., 2005] Heikki Keränen, Lauri Pohjanheimo ja Heikki Ailisto, Tag Manager: a Mobile Phone Platform for Physical Selection Services. IEEE International Conference on Pervasive Services 2005, IEEE, 2005, 405-412.

[Kindberg *et al.*, 2002] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra ja Mirjana Spasojevic, People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, Volume 7, Issue 5 (October 2002). Springer, 2002, 365-376.

[Lappeteläinen *et al*., 2004] Antti Lappeteläinen, Heikki Nieminen, Mikko Vääräkangas ja Hannu Laine, Overall MIMOSA architecture specification (OMAS). MIMOSA Deliverable D2.1, kesäkuu 2004. Saatavilla myös osoitteesta *http://www.mimosa-fp6.com/*.

[Siegemund & Flörkemeier, 2003] Frank Siegemund ja Christian Flörkemeier, Interaction in Pervasive Computing Settings using Bluetooth-Enabled Active Tags and Passive RFID Technology Together with Mobile Phones. *First IEEE International Conference on Pervasive Computing and Communications*, IEEE, 2003.

[Toye *et al*., 2004] Eleanor Toye, Anil Madhavapeddy, Richard Sharp, David Scott, Alan Blackwell ja Eben Upton, Using camera-phones to interact with context-aware mobile services. University of Cambridge, Computer Laboratory, Technical Report Number 609 (UCAM-CL-TR-609), joulukuu 2004. Saatavilla myös osoitteesta *http://www.cl.cam.ac.uk/TechReports/*.

[Tuomisto *et al*., 2005] Timo Tuomisto, Pasi Välkkynen ja Arto Ylisaukko-oja, RFID Tag Reader System Emulator to Support Touching, Pointing and Scanning. *Pervasive 2005*, Advances in Pervasive Computing, Austrian Computer Society, 2005, 85-88.

[Wagner *et al*., 2005] Daniel Wagner, Thomas Pintaric, Florian Ledermann ja Dieter Schmalstieg, Towards Massively Multi-user Augmented Reality on Handheld Devices. *Pervasive 2005*, Lecture Notes in Computer Science 3468, Springer-Verlag, 2005, 208-219.