Komponenttipohjainen visualisointi Internetissä

Sami Somero

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Pro gradu –tutkielma
Syyskuu 2000

Tampereen yliopisto

Tietojenkäsittelyopin laitos

Sami Somero: Komponenttipohjainen visualisointi Internetissä

Pro gradu -tutkielma, 73 sivua

Syyskuu 2000

Tiedon visualisointi voidaan esittää prosessina, jossa lähtötiedoista tuotetaan erilaisten välivaiheiden kautta graafinen esitys. Tämän tutkimuksen tarkoituksena on sovittaa visualisointiprosessi komponenttipohjaiseen ohjelmistoarkkitehtuuriin siten, että jokaista prosessin välivaihetta vastaa yksi ohjelmakomponentti. Komponenttipohjainen ratkaisu sovitetaan Internet-ympäristöön, mikä tuo mukanaan kysymykset järjestelmän hajautuksesta ja mahdollisista toteutustekniikoista.

Aihepiirin teoreettisen jäsentämisen lisäksi tutkimuksessa esitellään kolme Internet-pohjaista visualisointiympäristöä ja verrataan niiden ominaisuuksia teoriaosuudessa käsiteltyihin tekniikoihin ja ratkaisumalleihin.

Lopuksi tutkitaan yksittäisen komponentin toimintaa visualisointiympäristössä ja kehitetään malli yleiskäyttöisestä, ympäristöstä riippumattomasta visualisointikomponentista. Komponentin tietovirtoja ja ympäristöä tutkimalla kehitetään komponentin rakennemalli ja siitä havainnollistuksena yleiskäyttöisen visualisointikomponentin luokkakaavio.

Avainsanat ja -sanonnat: visualisointi, komponentit, Internet, arkkitehtuurit.

Sisällys

1 Johdanto

Internetiin ja siihen liittyviin tietojärjestelmiin sisältyy valtava ja jatkuvasti kasvava määrä tietoa. Tieto on sijoittunut esimerkiksi yritysten atk-järjestelmiin, tutkimuskeskusten tietokantoihin ja sähköisten kauppapaikkojen tietojärjestelmiin. Myös Internetin käytöstä kerätään jatkuvasti uutta tietoa. Käyttäjiä ja heidän liikkeitään verkkopalveluissa seurataan ja tilastoidaan. Tämän valtavan tietomassan tutkiminen ja hyödyntäminen vaatii uusia työvälineitä, jotka ottavat huomioon Internetin hajanaisuuden, tiedon suuren määrän sekä käyttäjien huomattavasti toisistaan poikkeavat tarpeet.

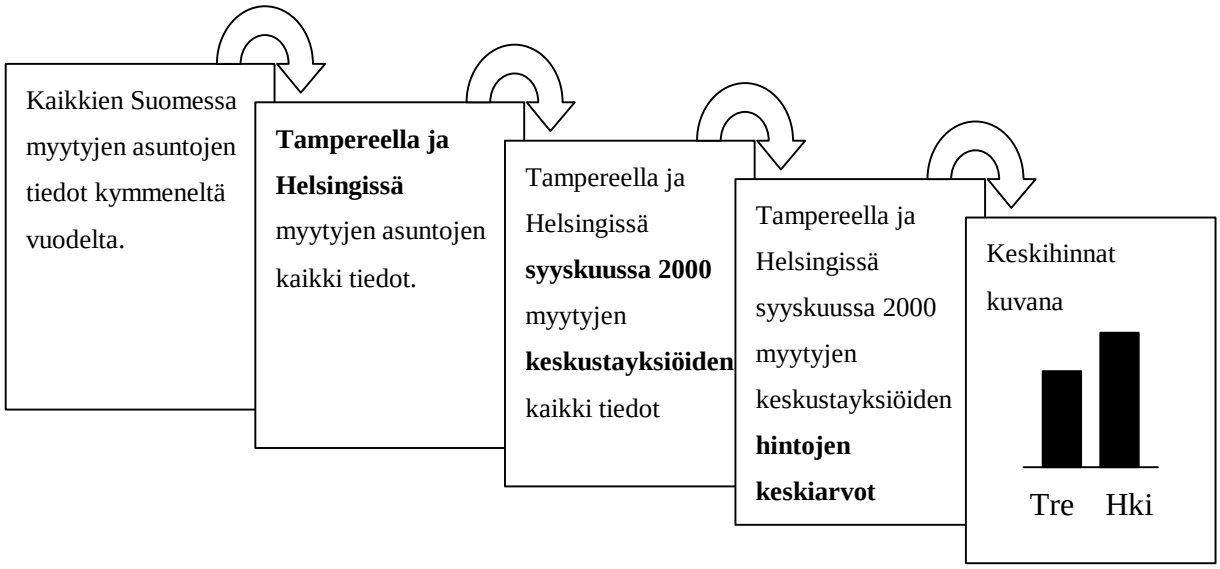
Tiedon visualisointi on tieteen ala, joka pyrkii löytämään yhä parempia tapoja havainnollistaa suuria tietomassoja kuvien avulla. Visualisointiin liittyy useita eri osa-alueita. Kaikissa ympäristöissä tapahtuvalle visualisoinnille yhteinen kiinnostava alue ovat erilaiset tiedon kuvaustavat. Joitain tietoja on luonnollista esittää pisteparvilla ja toisille tiedoille selkein esitystapa on kolmiulotteinen pinta. Tässä tutkimuksessa ei oteta kantaa näihin asioihin vaan keskitytään Internetissä tapahtuvan visualisoinnin ongelmakenttään.

1.1 Visualisointiprosessi

Visualisointi on prosessi, jossa lähtökohtana on suuri määrä tietoa. Prosessin aikana tiedosta poimitaan tutkijaa kiinnostava osuus ja muokataan se muotoon, jossa tiedon erityispiirteet ovat selvästi esillä. Tämän jälkeen valitaan sopiva tapa esittää tieto kuvana. Muokkaamalla visualisointiprosessin vaiheita, tutkija voi löytää uudenlaisia tapoja havainnollistaa alkuperäistä tietoa.

Tässä tutkimuksessa tullaan esittelemään malleja, joilla visualisointiprosessi voidaan jakaa useaan vaiheeseen. Vaihemallien perusteella tullaan esittämään erilaisia ratkaisuja, joilla vaiheisiin liittyvät tehtävät voidaan toteuttaa sovelluskomponentteina. Yhdenmukaisilla komponenteilla voidaan muodostaa visualisointiprosessia edistäviä ketjuja, jotka tuottavat tietomassasta useiden välivaiheiden kautta tietoa havainnollistavia kuvia (Kuva 1).

Kuva 1 - Visualisointiprosessi



Komponenttipohjaisen visualisointiprosessin toteuttamiseen Internet-ympäristössä liittyy monia huomioonotettavia asioita. Luvussa 6 tullaan esittelemään näitä asioita ja niihin sopivia ratkaisuvaihtoehtoja. Käsiteltäviä asioita ovat esimerkiksi toteutustekniikan valinta, tiedonsiirtomenetelmät sekä järjestelmän hajautus usealle eri tietokoneelle. Ratkaisuissa tukeudutaan yleisesti tunnettuihin tekniikoihin ja ohjelmistosuunnittelun menetelmiin.

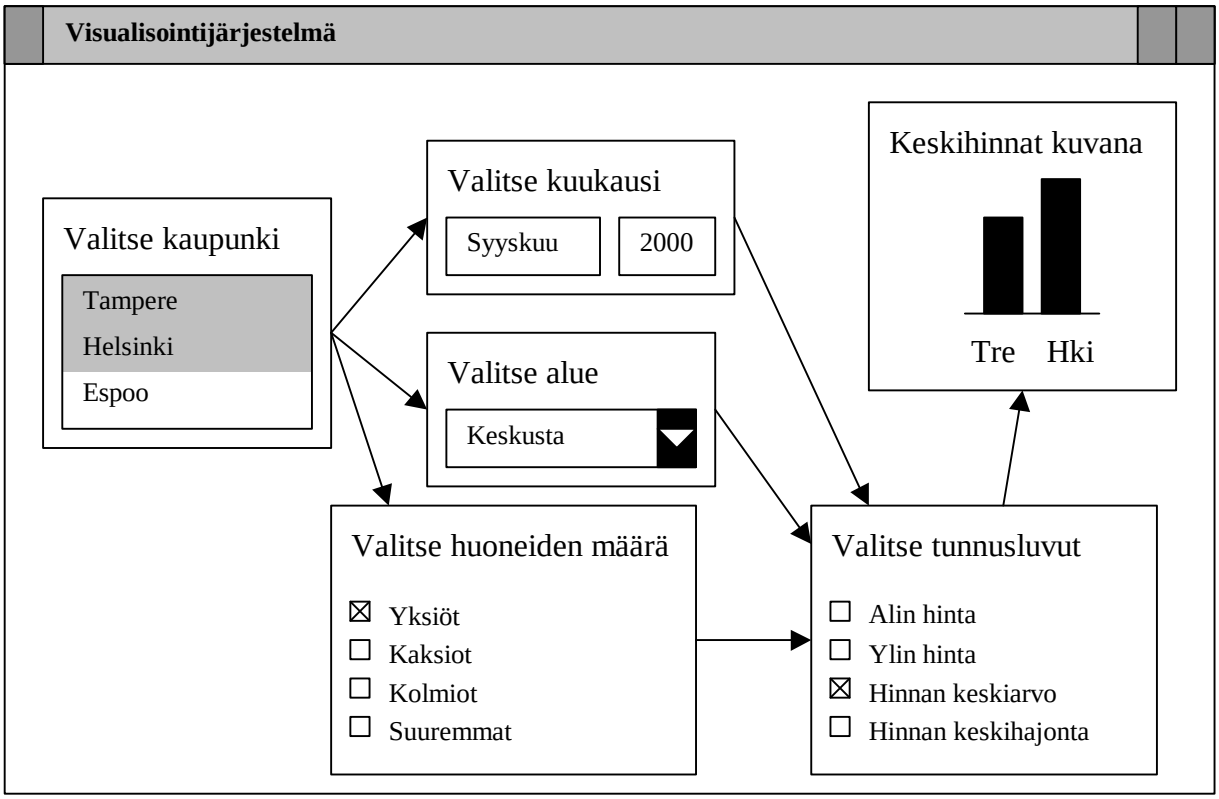
1.2 Visualisointijärjestelmä

Visualisointijärjestelmä on sovellus, jolla käyttäjä voi etsiä ja muokata tietoa sekä esittää sen kuvina (Kuva 2). Järjestelmä rakentuu visualisointikomponenteista. Komponenttien keskinäisiä sidoksia ja ominaisuuksia voidaan muokata graafisen käyttöliittymän avulla.

Internetin käyttö tiedon visualisoinnissa tarjoaa monia mahdollisuuksia. Moninaisten tiedon lähteiden lisäksi Internetille ominaista on käyttäjien välisten etäisyyksien katoaminen. Visualisointijärjestelmä voidaan nähdä laboratoriona, jossa eri puolilla maailmaa työskentelevät tutkijat voivat perehtyä yhteiseen tietomassaan ja jakaa havaintonsa toisten tutkijoiden kesken. Työskentely voi olla samanaikaista, jolloin kaikki tutkijat toimivat yhteisessä työtilassa nähden toistensa tekemän työn tosiaikaisena. Kun tutkijat tekevät työtä eri aikoihin, voidaan löytyneet visualisoinnit tallettaa tutkijoiden yhteiseen tiedostoon.

Tiedostosta on mahdollista selata eri tutkijoiden löytämiä kuvia yhteisestä tietomassasta [Isenhour et al.].

Kuva 2 - Visualisointijärjestelmä



Yliopistoissa on kehitetty komponenttipohjaisia visualisointijärjestelmiä, jotka toimivat Internet-ympäristössä. Tässä tutkimuksessa esitellään kolme erilaista visualisointijärjestelmää ja käydään läpi niiden ominaisuuksia aiemmin esiteltyjen toteutusvaihtoehtojen ja prosessimallien valossa. Valitut järjestelmät on tarkoitettu erilaisiin käyttötarkoituksiin ja erilaisille käyttäjäkunnille. Tuomalla esiin eri lähtökohdista tehtyjä visualisointijärjestelmiä, tutkimuksessa pyritään havainnollistamaan järjestelmälle asetettujen tarpeiden vaikutusta toteutustapojen valintaan. Kaikkiin käyttötarkoituksiin sopivia yleisratkaisuja ei siis välttämättä ole olemassa.

Lopuksi keskitytään tutkimaan yksittäisen visualisointikomponentin toimintaa ja kehitetään yleiskäyttöisen visualisointikomponentin runko.

1.3 Kaupallinen hyödyntäminen

Komponenttipohjaiset visualisointijärjestelmät eivät ole ainoastaan tieteellisten piirien konsepteja. Visualisointikomponenttien kaupallinen potentiaali on havaittu yrityksissä ja saatavissa on sekä visualisointijärjestelmien kehitysympäristöjä, että komponenttikirjastoja. Yleisesti hyväksytyt komponenttiteknologiat, kuten ActiveX ja JavaBeans, tuovat tavan, jolla ostettavissa olevat visualisointikomponentit ovat helposti liitettävissä ostajan omiin järjestelmiin.

Merkittävin osa tällä hetkellä myynnissä olevista visualisointikomponenteista on Microsoftin COM-tekniikkaan perustuvia ActiveX-komponentteja. Tämä johtuu siitä, että COM-komponentit ovat helposti liitettävissä Windows-sovelluksiin, kuten Microsoft Word:iin tai Excel:iin. Vähitellen visualisoinnissakin vahvistuva Internetin suosio lisää Java-pohjaisten visualisointikomponenttien suosiota. Monet aiemmin ActiveX-komponentteja toteuttaneet ohjelmistotalot ovat alkaneet kehittää myös JavaBean –visualisointikomponentteja.

1.4 Visualisointijärjestelmien tarve

Tietovarastojen suuret tietomassat luovat tarpeen järjestelmille, joilla tietoa voidaan havainnollistaa yksinkertaisella ja ymmärrettävällä tavalla. Koska käyttäjien tarpeet ovat hyvin moninaiset, ei tiedosta voida luoda kaikkien tarpeita täyttäviä valmiita esitystapoja. Visualisointijärjestelmien tarkoituksena on antaa käyttäjälle joustavat ja riittävän yksinkertaiset työkalut käyttäjän tietomassan muuttamiseksi havainnollisiksi kuviksi.

Visualisointijärjestelmille on olemassa paljon kaupallisia sovelluskohteita. Internet-kauppapaikasta voidaan vertailla tuotteiden ominaisuuksia ja hintoja. Tietoverkosta voidaan tutkia verkkoliikenteen jakautumista eri verkkoelementeille. Taloustiedoista voidaan etsiä säännönmukaisuuksia ja ennakoida tulevaa.

Tutkimuksessa ja opetuksessa jaetut visualisointiympäristöt edistävät kansainvälistä yhteistyötä. Internet poistaa etäisyyden tutkijoiden ja tutkittavan tietomassan väliltä. Tutkijat voivat jakaa havaintonsa ja tuoda näin esiin eri näkökulmia tutkittavasta kohteesta. Yhteisen havainnoinnin kautta tutkijoiden osaaminen voidaan koota yhteen ja jakaa kaikkien yhteiseksi hyödyksi.

Lukuisat mahdolliset käyttökohteet antavat uskoa, että Internet-pohjaisten visualisointijärjestelmien tulevaisuus on valoisa. Lähiaikoina visualisointijärjestelmien sovelluskohteet sijoittuvat ammattikäyttöön, opetukseen ja tutkimukseen. Tulevaisuudessa järjestelmät voivat löytää tiensä myös kotitalouksien työkaluksi.

2 Tiedon visualisointi

Tuhansien vuosien ajan ihmiset ovat ilmaisseet asioita kuvin ja merkein. Kartat ja tieteelliset piirrokset ovat helpottaneet ihmisten keskinäistä viestintää ja tiedonvälitystä. Kaikki tämä on ollut perinteisessä mielessä tiedon visualisointia – tiedon ilmaisua kuvien avulla [Owen]. Jatkossa tiedon visualisoinnista puhutaan lyhyemmin visualisointina.

Visualisoinnin käsitettä voidaan lähestyä tutkimalla englannin kielen sanan visualization merkityksiä. Termi ei ole tunnettu vain tietojenkäsittelyn piirissä, vaan sillä on analogisia merkityksiä muilla tieteenaloilla. Websterin sanakirja [Webster] antaa sanalle *visualization* seuraavat kolme merkitystä:

1. *formation of mental visual images*
2. *the act or process of interpreting in visual terms or of putting into visible form*
3. *the process of making an internal organ visible by the introduction (as by swallowing, by an injection, or by an enema) of a radiopaque substance followed by roentgenography”*

Ensimmäinen Websterin selityksistä tarkoittaa mielikuvien luomista, toinen tulkitsemista näköhavaintojen avulla ja kolmas varjoainekuvausta, jossa esimerkiksi ihmisen kehoon päästetyt aineet mahdollistavat tiettyjen ruumiinosien saamisen näkyviin röntgen-laitteilla. Pintapuolisesti erilaisissa merkityksissä on tarkemmin ajatellen paljonkin yhtäläistä. Kaikki edellä kuvatut merkitykset liittyvät yleisellä tasolla mielikuvien luomiseen, tiedon tulkitsemiseen sekä tavallisesti näkymättömissä olevan tuomista näkyviin.

Hieman erilainen määritelmä visualisoinnin käsitteestä on kirjassa ”Using Vision to Think” [Card et al., 6]. Kirjan määritelmässä on otettu tietojenkäsittelijän näkökulma visualisoinnin merkityksestä.

”The use of computer-supported, interactive, visual representations of data to amplify cognition.”

Tässä määritelmässä aikaisempiin visualisoinnin selityksiin on lisätty uusia vaatimuksia tietokoneavusteisuudesta ja interaktiivisuudesta. Määritelmän mukaan visualisointi on

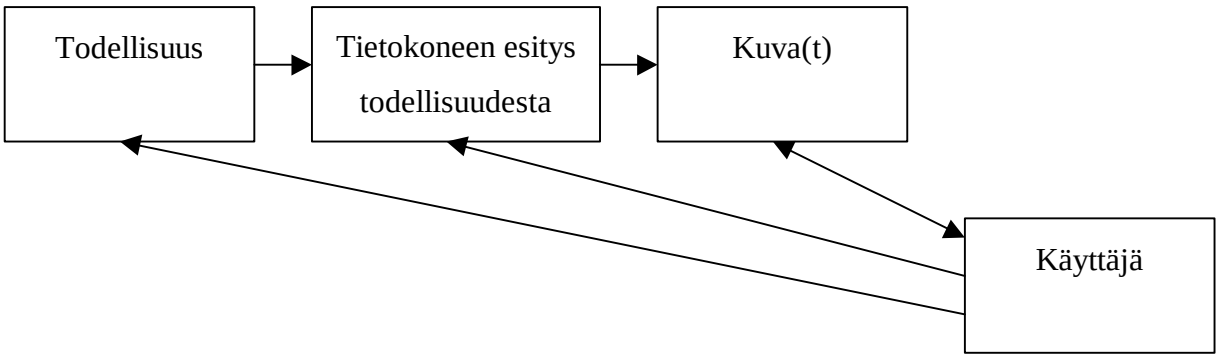
tietokoneavusteista, vuorovaikutteista ja kuvallista tiedon esitystä, jonka tarkoituksena on helpottaa ymmärtämistä.

Kolmannen niinikään tietojenkäsittelijöiden luoman määritelmän visualisoinnista antaa ACM:n visualisoinnin oppimateriaali, HyperVis [Owen]. Materiaalissa esitetään visualisoinnin klassinen määritelmä seuraavasti:

”The formation of mental visual images, the act or process of interpreting in visual terms or of putting into visual form.”

Visualisointiprosessia HyperVis havainnollistaa kaaviolla, jossa todellisuudesta tuotetaan tietokoneen avulla kuva, ja jossa käyttäjällä on mahdollisuus vaikuttaa todellisuuteen, tietokoneen prosessointiin sekä kuvaan. (Kuva 3)

Kuva 3



Yhteistä kaikille esitellyille määritelmille on visualisoinnin käsittäminen prosessina, joka tukee tutkittavan asian ymmärtämistä. Visualisoinnin käsite ei rajaa pois mitään sovellusaluetta vaan sen tarkoituksena on auttaa alasta riippumatta löytämään tutkittavasta uusia piirteitä ja vahvistaa havainnointia.

2.1 Visualisointitapoja

Tieto on mahdollista esittää kuvina lukemattomilla eri tavoilla. Visualisointina voidaan pitää niin tekstin korostamista lihavoinnilla kuin kuvia, animaatioita tai vuorovaikutteisia esityssovelluksiakin. Merkityksellistä on se, että visualisoinnilla on tietty sanoma, ja se tuodaan esiin graafisin menetelmin.

Tiedon visualisoinnin tutkituimmat osa-alueet painottuvat käyttöliittymärajapintaan. Tutkimuksissa yritetään löytää uusia tapoja esittää erilaisia tietoja yhä selkeämmillä kuvilla. Myös kuvien vuorovaikutteisuus on mielenkiintoinen tutkimuskohde. Uusilla työkaluilla yritetään tehdä visualisoitavan tiedon rajauksesta ja manipuloinnista yhä helpompaa ja joustavampaa.

2.2 Visualisoinnin vuorovaikutteisuus

Kuten aiemmin esitellyistä visualisointi-käsitteen tulkinnoista ilmeni, visualisointi voidaan eri yhteyksissä käsittää vuorovaikutteiseksi tai ei-vuorovaikutteiseksi toiminnaksi. Tämän mukaan erotetaan käsitteet dynaaminen ja staattinen visualisointi. Staattisella visualisoinnilla tarkoitetaan tietomassan sisällön kuvaamista kiinteällä kuvalla tai graafisella ilmauksella. Kun staattiseen visualisointiin liitetään uutena osana vuorovaikutteisuus käyttäjän kanssa, on kyseessä dynaaminen visualisointi.

2.2.1 Staattinen visualisointi

Staattisessa visualisoinnissa luodaan ennalta määrätyillä ehdoilla tietystä tietomassasta kiinteä näkymä. Tämän jälkeen näkymää ei voida muokata. Jos tietomassaan halutaan uusi näkymä, määritellään siihen liittyvät ehdot uudelleen ja luodaan uusi näkymä.

Staattisesta visualisoinnista voidaan pitää kuvaavana esimerkkinä vaikka sanomalehdissä esiintyvää uutisgrafiikkaa. Tällaisessa tapauksessa visualisoinnin sisällön päättää lehden toimittaja eikä lukija. Toimittaja on valinnut uutiseen liittyvästä tiedosta mielestään oleellisen ja yrittänyt löytää tavan, jolla tämän tiedon merkitys voidaan siirtää myös lukijalle.

Ongelmalliseksi staattinen visualisointi tulee kun tietomassa on laaja ja heterogeeninen. Chuah, Roth, Mattis ja Kolojejchick listaavat staattisen visualisoinnin ongelmia seuraavasti [Chuah et al.]:

1. Käyttäjä ei voi syventyä tiettyyn osaan näkymää ja katsoa sitä suuremmalla tarkkuudella yhtenäisessä kontekstissa kokonaisuuden kanssa. Tämä olisi hyödyllistä tietomassoissa, joissa on liian paljon tietoa yhdessä kuvassa näytettäväksi.
2. Visualisoinnissa, joissa katsekulmaa tai tietomassan rajauksia ei voida muuttaa, saattaa osa tietoalkioista jäädä toisten alkioiden taakse (occlusion).

3. Osa tietoalkioista voi olla täysin muista poikkeavia. Merkittävästi muista poikkeavien arvojen takia kuvan mittasuhteet saattavat olla sellaiset, että tietoalkioiden arvoja ei voi silmämääräisesti verrata keskenään.

4. Kiinnostavia tietomassan osia ei voida korostaa. Suuresta massasta olisi hyödyllistä voida esimerkiksi värein erottaa tiettyjä joukkoja.

5. Tietoalkioiden arvojen vertailu voi olla vaikeaa, koska kuvaan valittu katselukulma voi vääristää kuvan osien keskinäisiä suhteita.

2.2.2 Staattinen visualisointi Internetissä

Internet-ympäristössä käytetään paljon staattista visualisointia. Esimerkiksi useissa WWW-palveluissa esitetään graafisesti käyttäjämäärien kehitys ja käyttäjämäärän jakautuminen erilaisiin ryhmiin tulotason, ammatin tai iän mukaan. Usein ylläpitäjä päivittää nämä graafit käsin tai päivitykset tapahtuvat automaattisena eräajona esimerkiksi kerran kuukaudessa.

Staattinen visualisointi voi näkyä WWW-sivulla esimerkiksi GIF-kuvana. Tyypillisesti palvelimena toimii tavallinen WWW-palvelin, joka jakaa asiakkailleen levylle kuvatiedostoon talletettua visualisoinnin tulosta.

2.2.3 Dynaaminen visualisointi

Dynaamisessa visualisoinnissa voidaan suoravaikutteisesti muokata visualisointiin liittyviä määreitä (Dynamic Queries). Käyttäjällä on käyttöliittymän välityksellä mahdollisuus liikkua tietomassassa ja tehdä uusia rajauksia tai laajennuksia näkymiin. Tulokset ovat välittömästi nähtävissä visualisoinnin muuttumisena. Käyttäjän tekemät muutokset voivat liittyä joko esitettävän tiedon rajaamiseen tai sen esitystapaan.

Dynaamisen visualisoinnin avulla tietomassan tutkija voi etsiä alkioiden ominaisuuksien väliltä yhteyksiä tai säännönmukaisuuksia. Tutkija voi testata hypoteeseja, havaita poikkeuksia ja ryhmitellä tietoa. Havainnot voidaan liittää kuvina raportteihin. [Shneiderman]

Verrattuna staatiseen visualisointiin, dynaamisilla kyselyillä on myös omat ongelmansa. [Shneiderman]

1. Dynaamiset kyselyt asettavat suuria vaatimuksia laitteiston ja sovellusten suorituskyvylle. Suuren tietomassan käsittely on hidasta ja vasteajat ovat pitkät. Laitteiston käsittelykyvyn kasvaessa myös käsiteltävän tiedon määrä on kasvanut. Tämän vuoksi dynaamisissa kyselyissä voidaan harvoin puhua todellisesta suoravaikutteisuudesta.
2. Dynaamisia kyselyitä tukevan visualisointijärjestelmän luonti vaatii ohjelmointia, jotta sovellusalueen tarpeet voidaan ottaa hyvin huomioon.
3. Käyttöliittymät, jolla dynaamisia kyselyitä voidaan tehdä, mahdollistavat vain yksinkertaisia operaatioita, kuten rajauksia ja ryhmittelyjä.
4. Käyttöliittymät ovat vaikeita erityisesti heikkonäköisille tai huonosti visuaalisia rakenteita ymmärtäville käyttäjille.

Hyvä esimerkki dynaamisesta visualisoinnista on Dynamic HomeFinder [Williamson et al.]. Sovellus näyttää kartalla pisteinä myynnissä olevat asunnot. Kartan yhteydessä on valikoita, joilla voidaan rajata näytettävien asuntojen joukkoa. Käyttäjä voi esimerkiksi määritellä sopivan hintaluokan, neliömäärän ja huoneluvun. Rajausehtojen muuttuessa kartalle päivittyy jatkuvasti uusia ehtoja vastaavat asunnot. Amerikkalainen Spotfire Inc. on jatkokehittänyt Dynamic HomeFinderin perustekniikan kaupalliseksi tiedon rajaus- ja visualisointituotteeksi.

2.2.4 Dynaaminen visualisointi Internetissä

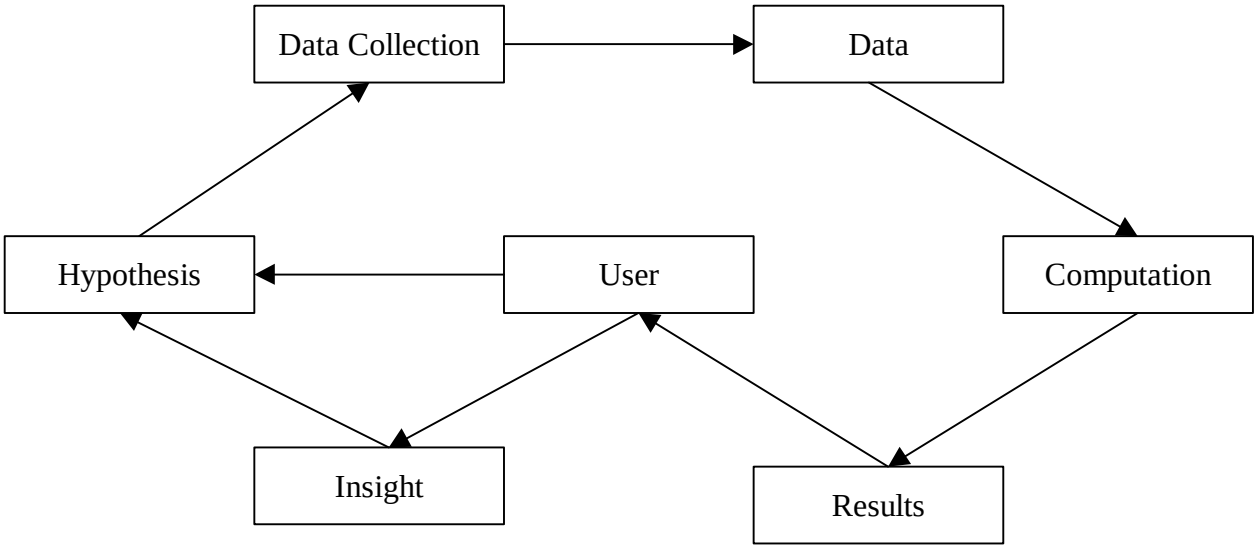
Dynaaminen visualisointi soveltuu hyvin Internet-käyttöön. Erääksi Internetin tärkeimmistä ominaisuuksista on katsottu ajasta ja paikasta riippumaton pääsy suuriin tietovarastoihin. Näiden tietovarastojen jäsentämiseen ja tiedon havainnollistamiseen dynaaminen visualisointi antaa hyvät mahdollisuudet.

Internet-ympäristön ongelmat tulevat teknisistä asioista. Suurien tietomassojen siirtäminen verkon sisällä on hidasta, kun taas dynaamisen visualisoinnin perusajatuksena on antaa käyttäjälle nopea tapa muokata visualisointia. [Shneiderman]

2.3 Visualisointiprosessi

Kuten aiemmin on todettu, visualisointi on käyttäjän ohjaama prosessi, jossa lähtökohtana oleva tieto pyritään muuttamaan ymmärrettävään kuvalliseen muotoon. ACM:n tuottama visualisoinnin oppimateriaali, HyperVis [Owen], esittää visualisointiprosessin eräänlaisena kiertokulkuna. Käyttäjä saa idean (Insight), jonka perusteella hän laatii hypoteesin (Hypothesis). Tämän jälkeen kerätään vaadittavat tiedot (Data Collection) ja suoritetaan kerätyn tiedon (Data) käsittely (Computation). Käsittelyn tulokset (Results) esitetään käyttäjälle (User), joka tulosten perusteella saa uuden idean tai muuttaa hypoteesiaan. (Kuva 4)

Kuva 4



Esitetyn mallin mukainen järjestelmä toimii käyttäjälle läpinäkyvästi, eli käyttäjä voi selvästi nähdä prosessin eri vaiheet ja vaikuttaa niiden toimintaan. Läpinäkyvyyden ansiosta tällainen järjestelmä mahdollistaa kokeilevan tutkimuksen. Tutkijaa tuetaan sekä ideoinnissa, että idean toimivuuden varmentamisessa [Owen].

3 Komponentit ja sovellusarkkitehtuurit

Nykyaikainen ohjelmistotuotanto pyrkii jakamaan sovellukset yhä selkeämpiin osakokonaisuuksiin. Olioperustaiset ohjelmointikielet ovat tuoneet luokan ja olion käsitteet, joilla ohjelmakoodi voidaan jakaa loogisiin kokonaisuuksiin. Olio-ohjelmien arkkitehtuuria selkeyttämään on kehitetty suunnittelumalleja, joilla tarkoitetaan tyypillisiin sovellusten toiminnallisuuksiin toteutettuja malliratkaisuja. Suunnittelumallit kokoavat usean olion joukon suorittamaan tietyn toiminnon.

Olioita korkeamman tason kokonaisuuksia ovat suunnittelumallien lisäksi komponentit. Komponentilla on olioiden kanssa monia yhteisiä ominaisuuksia, kuten toteutusratkaisuiden piilottaminen julkisen rajapinnan taakse. Komponentti on kuitenkin tyypillisesti suurempi kokonaisuus, jonka toimintaa ei ohjata ainoastaan ohjelmakoodin tasolta.

Sanojen merkitysten selittämiseen erikoistunut WhatIs.com määrittelee ohjelmakomponentin (component) seuraavasti [Whatis]:

In object-oriented programming and distributed object technology, a component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application.

Palvelun antamassa määritelmässä komponentin käsite yhdistetään olio-ohjelmointiin ja hajautettuihin oliotekniikoihin. Komponenttiin liitettäviä ominaisuuksia ovat uudelleenkäyttö, yhden komponentin kyky kytkeytyä toisiin komponentteihin samassa tietokoneessa tai hajautetussa ympäristössä toisissa tietokoneissa.

Sun Microsystems näkee komponentit rajapintojen ympäröiminä rakennuspalikoina, joista voidaan koota olioperustaisia sovelluksia [Sun]:

A piece of software with a special interfaces that enable it to plug into other software components as a building block. Components can be assembled into new object applications.

Komponentin käyttäjälle on oleellista tietää, miten komponenttia käytetään ja mitä se tekee. Sen sijaan merkitystä ei ole sillä, millaisen teknisen toteutuksen komponentti pitää sisällään. Hyvä komponentti on yleiskäyttöinen, eli samaa toiminnallisuutta voidaan käyttää moneen tarpeeseen. Komponenttien kyky kytkeytyä toisiinsa toteutetaan rajapintoina, joiden kautta

komponentti saa syötteensä, lähettää tuloksensa eteenpäin ja joiden kautta komponentin toimintaa voidaan ohjata.

Oliot täyttävät jo sinänsä monia komponentin edellytyksiä. Oliolla on tietyt metodit, joiden kautta sitä voidaan käyttää, eikä metodeja kutsuvien olioiden tarvitse tuntea niiden toteutustapoja. Olioiden käyttökelpoisuutta komponentteina vähentävät niiden sekalaiset käyttötavat. Käyttötapojen yhdenmukaistamiseksi voidaan määritellä rajapintoja, jotka toteutetaan kaikissa samantyyppisiin käyttötarkoituksiin tarkoitetuissa luokissa. Yhteisen rajapinnan avulla kaikkia luokkia voidaan käyttää samoilla menetelmillä. Rajapinnoilla ja vastaavilla perintätekniikoilla toimivia järjestelmiä ovat mm. Java-appletit ja JavaBean-komponentit.

Yleisesti oliokomponentit liittyvät kiinteästi tiettyyn ohjelmointikieleen. Java-olioita voidaan käsitellä Java-kielisissä sovelluksissa ja C++-olioita C++-kielisissä sovelluksissa. Komponentteja voidaan usein käyttää muutenkin kuin ohjelmakoodista käsin. Seuraavissa kappaleissa esitellään muutama tekniikka, jolla voidaan toteuttaa yleiskäyttöisiä komponentteja. Toteutettuja komponentteja voidaan käyttää WWW-sivulla tai osana Microsoft Windows -sovellusta.

3.1 Yleiset komponenttimallit

Yleiset komponenttimallit tarjoavat teknisen perustan komponenttipohjaiselle sovelluskehitykselle. Yleisesti tunnettujen mallien käyttö lisää komponenttien yleiskäyttöisyyttä ja pitkäikäisyyttä. Hyvin suunnitellut ja yleisesti tunnettuja komponenttimalleja käyttävät komponentit ovat hyödynnettävissä sellaisenaan useissa eri sovelluksissa. Komponenttimallit mahdollistavat uusien sovellusten laatimisen käyttämällä olemassaolevia ohjelmakomponentteja. Komponentteja voidaan toteuttaa esimerkiksi JavaBeans- ja ActiveX-tekniikoilla. [Sun]

Yleinen komponenttikehys määrittelee komponenteille yleensä seuraavat ominaisuudet [Whatis; Sun]:

- **Komponenttirajapinnat,** joiden kautta sekä saman kehittäjän tai vaikka toisen ohjelmistotalon kehittäjien on helppo käyttää komponentin palveluita,

- **Komponentin ominaisuudet,** joiden avulla komponentin piirteet voivat olla näkyvissä komponentin ulkopuolelle,
- **Tapahtumankäsittely,** jonka avulla komponentit voivat muodostaa keskinäisiä kontakteja ja reagoida toistensa tapahtumiin,
- **Pysyvyys,** jonka avulla komponentin tila on talletettavissa,
- **Tuki sovelluskehittimille,** mikä mahdollistaa uusien komponenttien helpon luomisen ja olemassaolevien komponenttien helpon analysoinnin, sekä
- **Komponenttien paketointi,** minkä avulla komponenttiin liittyvät tiedostot voidaan koota yhdeksi kokonaisuudeksi, joka on helposti asennettavissa, poistettavissa ja vaikka myytävissä.

Yleisen komponenttimallin pohjalta toteutetuista komponenteista on koottu komponenttikirjastoja. Komponenttikirjastoja valmistavat yritykset myyvät asiakkailleen valmiita käyttöliittymä- ja visualisointikomponentteja, joiden sovittaminen asiakkaiden järjestelmiin on yksinkertaista.

3.1.1 JavaBeans

JavaBeans-tekniikka on kehys Java-kielellä toteutetuille ohjelmakomponenteille. Tarkoituksena on, että eri ohjelmistoyritykset voivat toteuttaa keskenään yhteensopivia ohjelmakomponentteja, joista loppukäyttäjä voi koota omia sovelluksiaan [Hamilton].

JavaBean-komponentteja voidaan käyttää joko osana sovellusta tai osana dokumenttia. Sovelluksen osana komponentti voi olla esimerkiksi käyttöliittymän toiminnallinen osa, kuten valintalista tai liukusäädin. Dokumentissa JavaBean voi esiintyä esimerkiksi taulukkolaskentakomponenttina.

Sun Microsystems määrittelee JavaBean-komponentin seuraavasti [Sun]:

JavaBean on Java-kielellä toteutettu uudelleenkäytettävä komponentti, jota voidaan muokata graafisilla työkaluilla.

JavaBean-komponentit voivat olla toiminnaltaan yksinkertaisia tai monimutkaisia. Kaikki ne ovat kuitenkin yhdistettävissä toisiinsa tai muokattavissa graafisilla työkaluilla. Graafisella työkalulla voidaan tarkoittaa esimerkiksi sovelluskehittimen osaa tai tekstinkäsittely-

ohjelmaa. JavaBeans-tekniikka sopii hyvin käyttötarkoituksiin, joissa komponentteja halutaan käyttää graafisten työkalujen avulla. Valmiit sovelluksen osat, joita käytetään pääsääntöisesti ohjelmakoodista käsin, sopivat paremmin luokkakirjastoiksi. [Hamilton]

Normaalisti keskinäisessä yhteydessä toimivat JavaBean-komponentit toimivat saman virtuaalikoneen sisällä. Komponentti voi kuitenkin edustaa yhteyksiä toisiin palvelimiin tai toimintoja, jotka käytännössä suoritetaan toisella palvelimella. JavaBeans-tekniikka ei pidä sisällään omia hajautustekniikoitaan vaan hajautus tapahtuu tarvittaessa yleisillä tekniikoilla kuten CORBA:lla tai RMI:llä. Hajautustekniikoista kerrotaan lisää kappaleessa 3.3.

3.1.2 COM

Microsoftin Windows-ympäristössä voidaan rakentaa COM–mallin (The Component Object Model) mukaisesti yleiskäyttöisiä komponentteja. COM-komponentteja voidaan käyttää määrättyjen rajapintojen kautta tietämättä komponenttien sisäisiä toteutusratkaisuja. COM-tekniikka määrittelee ohjelmistoarkkitehtuurin, jolla eri ohjelmistotoimittajien sovelluskomponentit on kytkettävissä toisiinsa. Kun JavaBean-komponentteja voidaan tehdä ainoastaan Java-kielellä, on COM-komponentteja mahdollista toteuttaa useilla eri ohjelmointikielillä. COM-tekniikka on käytössä Microsoft Windows, Apple MacOS ja UNIX-ympäristöissä. [Williams et al.]

COM-tekniikka on Microsoftin vastaus kysymyksiin:

- Miten sovelluskehittäjät voivat tehdä keskenään yhteensopivia ohjelmakomponentteja?
- Miten sovelluksen yksi komponentti voidaan päivittää ilman koko sovelluksen päivittämistä?
- Miten eri ohjelmointikielillä toteutetut komponentit voivat keskustella keskenään?
- Miten samalla mallilla voidaan toteuttaa komponenttien yhteistyö samassa käyttöjärjestelmän prosessissa, eri prosesseissa tai jopa verkon eri tietokoneilla?

3.2 Verkkosovellusten komponenttimallit

Tässä dokumentissa ollaan erityisen kiinnostuneita komponenteista, joita voidaan julkaista WWW-sivuilla. Tällaisiin komponentteihin on kaksi kilpailevaa toteutusmallia; Microsoftin ActiveX ja Sun Microsystems:n Java Applet –tekniikka.

3.2.1 Java Applet

Java-appletit ovat Java-kielellä toteutettuja komponentteja, jotka sijaitsevat yleensä HTML-sivuilla. Verkkoselain käynnistää sivulla sijaitsevat appletit ja antaa niille syötteenä HTML-sivulla määritellyt syöteparametrit. Appletit voivat tutkia sivua, jolla sijaitsevat ja vaikuttaa verkkoselaimeen, joka appletit käynnisti. Samalla sivulla sijaitsevat appletit voivat kommunikoida keskenään.

Applettien toiminta on tietoturvasyistä rajoitettua Java-sovelluksiin verrattuna. Appletti ladataan palvelimelta ja käynnistetään HTML-sivua selaavan käyttäjän tietokoneella. Ilman turvallisuusrajoitteita applettien avulla olisi helppo toteuttaa viruksia, jotka väärinkäyttäisivät tietokoneen resursseja ja verkkoympäristöä. Rajoitteilla on estetty applettien yhteydenotto ajoympäristönä toimivaan tietokoneeseen ja sitä ympäröivään verkkoympäristöön. Appletti voi kuitenkin ottaa yhteyttä palvelimeen, josta se on selaimeen latautunut.

Koska appletti on usein turhan suuri kokonaisuus yhdeksi komponentiksi, sen toiminnallisuus voi olla järkevää jakaa osakokonaisuuksiksi. Hyvä tapa toteuttaa nämä osat on tehdä niistä JavaBean-komponentteja. Tällä tavoin yhdistetään Applet-komponentin mahdollisuudet JavaBeans:n tarjoamaan ympäristöriippumattomaan komponentti-arkkitehtuuriin.

3.2.2 ActiveX

ActiveX on Microsoftin toteuttama kokoelma olio-ohjelmointitekniikoita ja -työkaluja. Perustekniikkana ActiveX:ssä on COM- ja DCOM-komponenttimallit. ActiveX-komponentit voivat toimia WWW-sivulla, MS Office –sovelluksissa tai ohjelmointiympäristöissä sekä Microsoft Windows- että Apple MacOS -ympäristössä. [Whatis]

Komponentteja voidaan kirjoittaa usealla eri ohjelmointikielellä, mikä mahdollistaa sovellusten kokoamisen eri ohjelmointikielillä tehdyistä komponenteista. ActiveX-komponenttien tekoon on olemassa myös sovelluskehittimiä [Jern2,Whatis].

ActiveX-komponentit pystyvät keskustelemaan toimintaympäristönsä, kuten WWW-sivun, kanssa. WWW-sivulla komponentille voidaan välittää tietoa JavaScript:n avulla ja vastaavasti ActiveX-komponentista voidaan kutsua JavaScript-metodeita. Tämä ominaisuus

tekee komponenttien käytöstä helppoa ja vähentää tilanteita, joissa komponentin toiminnan muokkaaminen vaatii muutoksia komponentin ohjelmakoodiin. [Johns]

3.3 Hajautusta tukevat tekniikat

Hajautustekniikoiden käyttö visualisointiympäristön rakenteessa mahdollistaa raskasta laskentaa vaativien töiden jakamisen usealle eri tietokoneelle. Hajautus mahdollistaa myös yhteisölliset visualisointiympäristöt ja tietokoneavusteisen ryhmätyön (CSCW). Tällaisia monen samanaikaisen käyttäjän visualisointilaboratorioita voidaan käyttää useassa paikassa samanaikaisesti tapahtuvassa tutkimuksessa tai verkossa tapahtuvassa opetuksessa. Hyvä esimerkki ryhmätyöskentelyä tukevasta visualisointijärjestelmästä, on tässä tutkimuksessa myöhemmin esiteltävä Sluice [Isenhour et al.].

3.3.1 CORBA

CORBA (Common Object Request Broker Architecture) on arkkitehtuuri ja määrittely, jonka avulla voidaan luoda, jakaa ja hallita verkossa hajautettujen sovellusten olioita. Se mahdollistaa eri paikoissa sijaitsevien ja eri valmistajien toteuttamien sovellusten kommunikoinnin verkossa yleisesti tunnettujen rajapintojen välityksellä. CORBAn on kehittänyt yhteistyöorganisaatio OMG (Object Management Group), johon kuuluu yli 500 jäsenyritystä. ISO (International Organization for Standardization) on julistanut CORBAn yleiseksi arkkitehtuuriksi hajautettujen sovellusten toteuttamiseen [Whatis].

3.3.2 Java RMI

Sun Microsystems on kehittänyt oman tekniikkansa hajautettujen sovellusten toteuttamiseen. RMI (Remote Method Invocation) [RMI] on olioteknologia, joka mahdollistaa eri tietokoneilla sijaitsevien olioiden keskinäiset metodikutsut. RMI on kevyt sovelluskehys, jonka käytön oppiminen ei Java-kieltä osaavalle ole vaikeaa. Yksinkertaisuus johtuu siitä, että RMI-tekniikassa ei ole jouduttu tekemään ratkaisuja, jotka takaisivat yhdenmukaisen toiminnan eri ohjelmointikielillä toteutettujen olioiden välillä. [RMI2]

Java-sovelluksissa hajautus voidaan toteuttaa sekä RMI-, että CORBA-tekniikoilla. Monesti valinnassa päädytään CORBA:an johtuen sen suosiosta muilla ohjelmointikielillä toteutettujen järjestelmien hajautuksessa.

3.3.3 DCOM

DCOM (Distributed COM) on kokoelma Microsoftin malleja ja ohjelmointirajapintoja, joiden avulla asiakassovelluksen olio voi kutsua palveluita eri tietokoneilla toimivien sovellusten olioilta. DCOM perustuu COM-tekniikkaan, joka mahdollistaa olioiden keskinäisen yhteistyön yhden tietokoneen ympäristössä [Whatis].

DCOM kätkee hajautuksen siten, että käyttäjän näkökulmasta sovellus tuntuu toimivan samalla tavoin kuin paikallinenkin sovellus. Hajautuksen avulla useita eri tietokoneita voidaan asettaa ratkaisemaan samaa ongelmaa. [Jern3]

DCOM on perusajatukseltaan vastaavanlainen kuin CORBA. Molemmat tarjoavat joukon hajautetun sovelluksen toteuttamisessa tarvittavia työkaluja. DCOM on Microsoftin ratkaisu verkonlaajuiseen olioympäristöön. CORBAn ratkaisuja tukevat lähes kaikki muut alan yritykset. CORBAn vahvan aseman vuoksi Microsoft on joutunut toteuttamaan tekniikan, jolla DCOM- ja CORBA-sovellukset voivat keskustella keskenään. [Whatis]

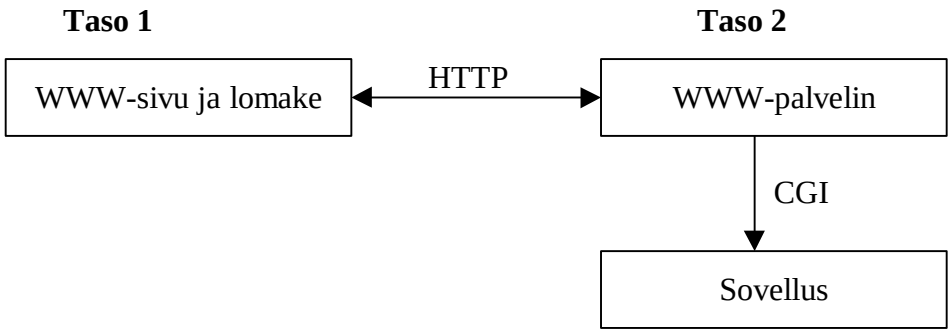
3.4 Kaksitasoiset sovellukset Internetissä

Verkkoympäristöön toteutetut sovellukset ovat usein hajautettuina usealle eri tietokoneelle. Kaksitasoisen arkkitehtuurin (client-server architecture) mukaisessa ratkaisussa sovellus jakautuu kahteen osaan, jotka ovat palvelin ja asiakas. Yhdellä palvelimella voi olla useita eri asiakkaita ja yksi asiakas voi keskustella usean eri palvelimen kanssa. Kaksitasoisen sovelluksen ensimmäinen taso edustaa käyttöliittymää ja toinen sovelluksen toimintalogiikkaa [Sun]. Tässä kohdassa kaksitasoista arkkitehtuuria tullaan kuvaamaan käyttämällä Java-teknologiaa. Valinta perustuu Javan avoimmuuteen verrattuna kilpailevaan ActiveX-teknologiaan.

HTTP-protokolla (Hyper Text Transfer Protocol) [HTTP] määrittelee WWW-selaimen ja palvelimen välisessä keskustelussa käytetyn kielen. HTTP-protokollan avulla verkkoselaimen käyttäjä voi pyytää palvelimilta HTML-sivuja ja saada ne vastauksena näytölleen. Protokollan ominaisuutena on rakenteellinen selkeys. Sekä pyyntö, että vastaus kulkee selkokielisenä tekstinä ja HTTP-paketin sisältöä kuvaavat tiedot ovat nimi=arvo -tyyppisinä pareina kunkin pyynnön ja vastauksen alussa. HTTP-soveltuukin nimenomaan palvelimen ja asiakkaan väliseen keskusteluun. Palvelinten keskinäinen vuorovaikutus tapahtuu usein tehokkaampia protokollia käyttämällä.

Yksinkertaisin tapa toteuttaa kaksitasoisella arkkitehtuurilla rakentuva sovellus Internetiin on käyttää CGI-tekniikkaa (Common Gateway Interface). CGI-sovellus toimii samassa ympäristössä WWW-palvelimen kanssa. Kun asiakassovellus, yleensä WWW-selain, haluaa käyttää CGI-tekniikalla toteutettua palvelinsovellusta, keskustelu tapahtuu WWW-palvelimen välityksellä. (Kuva 5) CGI-sovelluksia voidaan toteuttaa millä tahansa ohjelmointikielellä.

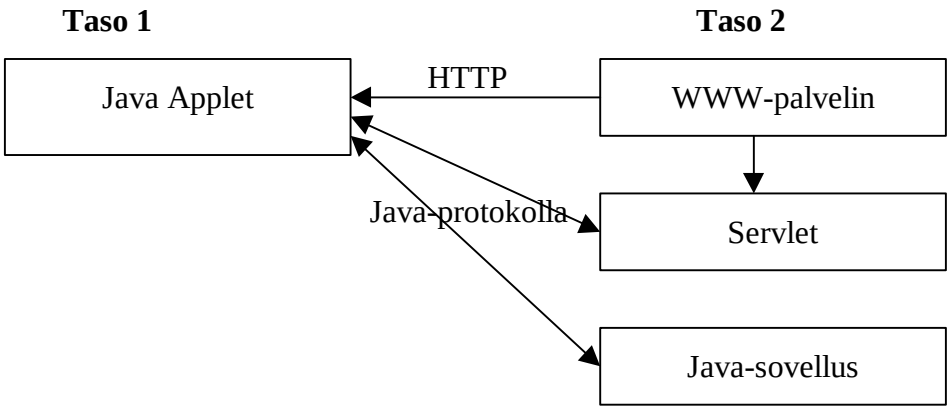
Kuva 5



Servlet-teknologia on Java-pohjainen vaihtoehto CGI:lle. Servletit ovat yksinkertaisen rajapinnan toteuttavia Java-komponentteja, jotka luovat käyttäjien pyynnöstä HTML-sivuja tai muita HTTP-vastauksia. Servletit toimivat täysin palvelimella eikä niillä ole omaa käyttöliittymää asiakkaan suuntaan. Servlet- ja CGI-sovellukset osaavat ainoastaan vastata HTTP-pyyntöihin ja luoda vastauksiksi esimerkiksi HTML-sivuja. Käyttäjän näkemän sivun sisältö muuttuu ainoastaan sivun latausten yhteydessä.

Java Applet –tekniikalla voidaan toteuttaa WWW-sivulle komponentteja, jotka mahdollistavat käyttäjän ja palvelun vuorovaikutuksen ilman jatkuvia sivun latauksia. Appletti näkyy WWW-sivulla kiinteän kokoisena kuvamaisena komponenttina. Sillä voi olla käyttöliittymäkomponentteja, joiden avulla ohjataan appletin toimintalogiikkaa. Appletti voi ottaa yhteyksiä palvelimella sijaitseviin sovelluksiin. Palvelimen kanssa keskustelu voi tapahtua esimerkiksi HTTP-protokollalla tai porttiyhteyksillä. Oliotason yhteys voidaan toteuttaa CORBA- tai Java RMI –tekniikalla. Oleellista on, että appletti voi keskustella vain niiden palvelinsovellusten kanssa, jotka toimivat palvelinkoneella, josta appletti on WWW-selaimeen ladattu. (Kuva 6)

Kuva 6



Java Applet-tekniikalla voidaan toteuttaa WWW-sivuille monimutkaisia käyttöliittymiä, jotka antavat käyttäjän tekemille valinnoille nopean vastauksen. Appleteilla on kuitenkin omat ongelmansakin. Kun HTML-sivu latautuu yleensä sekunnin murto-osassa, voi appletin latautumien kestää useita sekunteja. Käynnistyttyään appletti toimii nopeasti ja tarjoaa käyttäjälleen merkittävästi HTML-lomakkeita miellyttävämmän käyttöliittymän. Applettien ongelmana on WWW-selainten toisistaan poikkeavat Java-ajoympäristöt. Kaikki WWW-selaimet eivät tue Java-appletteja, osa tukee vain Javan vanhoja versioita ja kaikilla valmistajilla on Java-virtuaalikoneesta hieman erilaiset toteutukset.

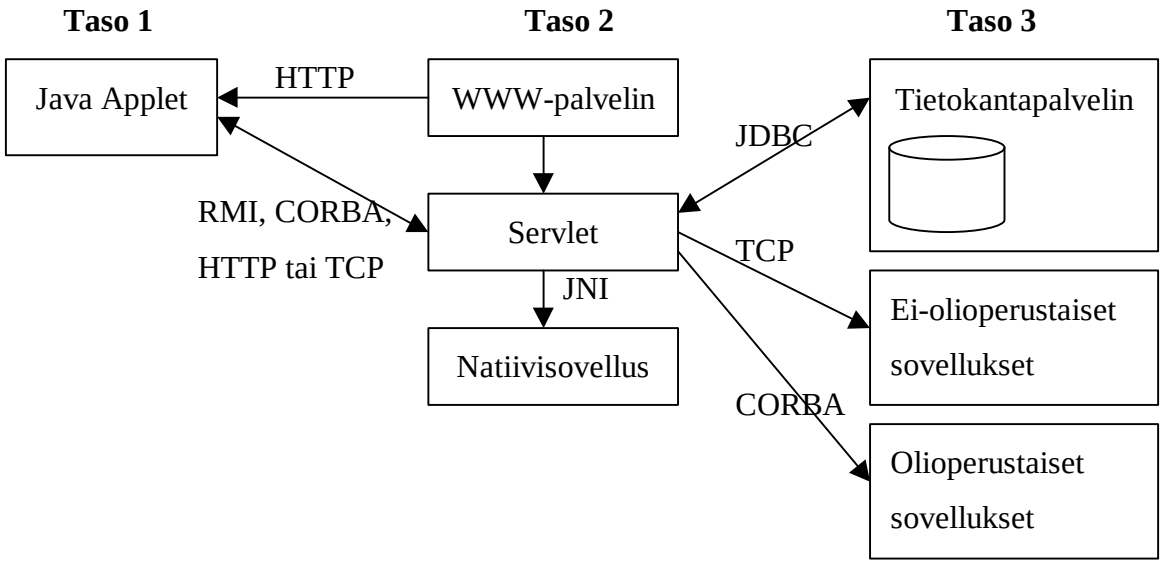
3.5 Kolmitasoiset sovellukset Internetissä

Kolmitasoisen arkkitehtuurin (three-tier architecture) ominaisuuksiin kuuluu selkeä looginen jaottelu eri tasojen välillä. Kaksitasoisen arkkitehtuurin tavoin ensimmäiseen tasoon kuuluu käyttöliittymä, syötteiden tarkistus sekä yksinkertaiset laskutehtävät. Toiselle tasolle on sijoitettu sovelluksen toimintalogiikka sekä käyttäjähallinta. Kolmannella tasolla sijaitsee tietokanta, muut tietovarastot sekä taustalla toimivat sovellukset [Sun] (Kuva 7). Tässä kohdassa kolmitasoinen arkkitehtuuri tullaan esittämään käyttämällä Java-teknologian ominaisuuksia.

Eri tasojen loogisella jaottelulla parannetaan järjestelmän modulaarisuutta eli sovelluksen jakautumista selkeisiin ja itsenäisiin osakokonaisuuksiin. Lähtökohtana on, että mikä tahansa

tasoista voidaan korvata uudella toteutuksella ilman, että muihin tasoihin täytyy tehdä muutoksia. Internet-pohjaisessa sovelluksessa tälle ominaisuudelle on monia käyttömahdollisuuksia. Järjestelmään voidaan esimerkiksi helposti luoda uusia käyttöliittymiä. Niinikään taustalla toimivat tietovarastot on helposti kytkettävissä eri asiakkaiden taustajärjestelmiin tai vanhoihin sovelluksiin.

Kuva 7



Ensimmäinen taso eli käyttöliittymä voidaan toteuttaa Java Appletilla, Java-sovelluksella tai HTML-sivuina. Java-toteutukset tarjoavat mahdollisuudet monipuolisempaan käyttöliittymäsuunnitteluun ja hajautustekniikoiden avulla käyttöliittymän sitominen toisen tason sovelluslogiikkaan on yksinkertaista.

Toinen taso voidaan toteuttaa WWW-palvelimella ja sen CGI- tai Servlet-laajennuksilla. Lisäksi toinen taso voi sisältää Java-sovelluksia tai ns. natiivisovelluksia, eli kyseiseen järjestelmään käännettyjä tietokoneohjelmia. Natiivisovellusten käyttöä varten Java-kielessä on määritelty JNI eli Java Native Interface. JNI:n avulla Java-sovelluksessa voidaan käsitellä esimerkiksi C++-ohjelmakoodia ja –olioita.

Kolmannelle tasolle kuuluu tietokannat ja tietovarastot. Tietokantayhteydet Java-kielessä toteutetaan JDBC:n (Java DataBase Connectivity) avulla. JDBC:llä voidaan käyttää kaikkia yleisimpiä tietokantoja.

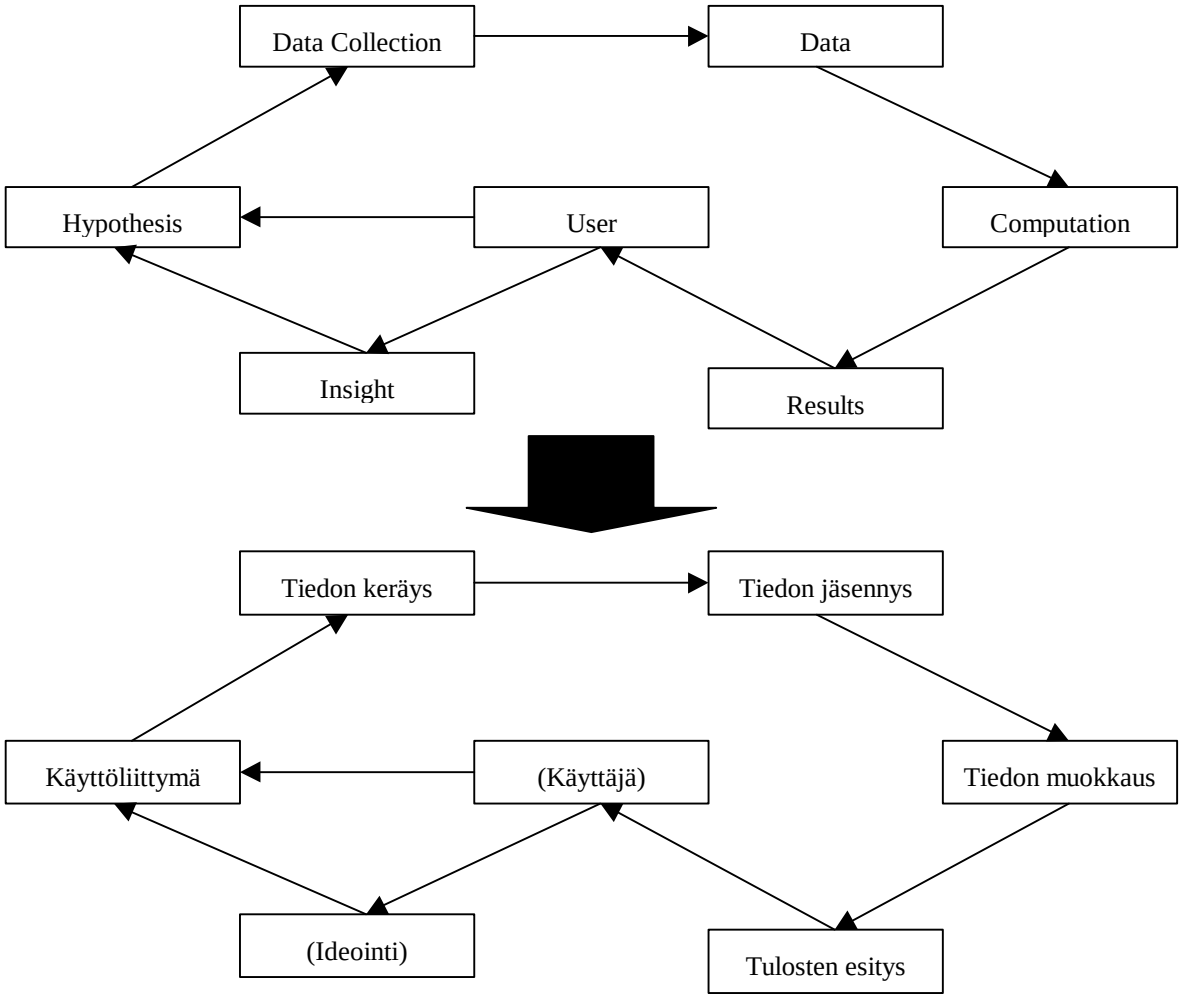
Kolmannella tasolla voi sijaita myös sovelluksia. Oliopohjaisiin sovelluksiin voidaan ottaa yhteys hajautettujen järjestelmien komponenttiteknologioilla, RMI:llä ja CORBA:lla. Vanhoihin sovelluksiin ja tietojärjestelmiin voidaan kytkeytyä TCP-porttiyhteyksillä. Kolmannen tason sovellukset ovat usein yrityksen vanhoja atk-järjestelmiä. Kolmitasoarkkitehtuuri antaa mahdollisuuden liittää samaan järjestelmään sekä uusia oliokomponentteja että vanhoja palvelinsovelluksiakin ilman että kokonaisjärjestelmän uudelleenkäyttöarvo heikkenee.

4 Komponenttipohjainen visualisointi

Komponenttipohjaista arkkitehtuuria on alettu käyttää myös visualisointijärjestelmissä. Visualisointi tapahtuu usein kokeellisesti tekemällä hypoteeseja ja testaamalla niitä. Tällainen ajatusmalli sopii hyvin komponenttiajatteluun. Visualisointiprosessi voidaan jakaa toiminnallisesti selväpiireisiin osaprosesseihin. Kukin osaprosessi on mahdollista toteuttaa omana komponenttinaan, joka sovitun komponenttirajapinnan kautta on kytkettävissä muuhun komponenttiketjuun.

Visualisointiprosessi voidaan jakaa osatehtäviksi useallakin eri tavalla. Seuraavassa esitellään tehtäväketju, joka perustuu kohdassa 2.3 esiteltyyn prosessikuvaukseen [Owen]. Visualisointiprosessia esittävästä kuvasta on johdettu tehtäväketju, joka alkaa käyttäjän syötteestä ja päättyy visualisoinnin tulosten esittämiseen (Kuva 8). Kuvassa kutakin prosessin vaihetta vastaa sovelluksen toiminnallinen osa.

Kuva 8

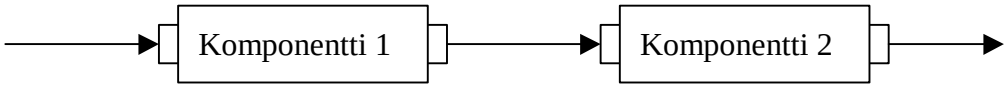


Käyttäjä syöttää visualisointijärjestelmälle hypoteesin käyttöliittymän kautta. Tästä seurauksena *tiedon keräys* -komponentti kokoaa tiedon tietokannasta tai muista tiedon lähteistä. Kerätty tieto jäsennetään visualisointijärjestelmän yleiseen tietomuotoon, jossa sitä voidaan käsitellä muokkauskomponenteilla. Muokkauskomponentit rajaavat tietoa tai luovat sinne uusia laskennallisia arvoja. *Tulosten esitys* –komponentti näyttää muokkauskomponenttien tuottaman arvojoukon käyttäjälle visuaalisesti.

Komponenttipohjainen visualisointi voi tapahtua edellä kuvatulla tavalla usean komponentin ketjuna, mutta visualisointikomponentti voi olla myös itsenäinen sovellus, joka tuottaa määrämuotoisesta syötteestä kuvan. Tällöin komponentti sisältää kaikki visualisointiprosessin vaiheet. Komponentti saa syötteenään visualisoitavan tiedon, muokkaa sen tämän jälkeen visuaalisiksi rakenteiksi ja esittää sen lopuksi kuvana. Tällöin sovelluksen ainoa komponenttimainen piirre on rajapinta, jonka kautta visualisoitava tietojoukko syötetään. Monessa käyttötarkoituksessa tämä on riittävä. Esimerkiksi amerikkalainen ohjelmistotalo AVS (http://www.avs.com) tarjoaa komponentteja, joilla voidaan visualisoida Excel-taulukosta rajatun alueen arvot. Tällainen komponentti ottaa syötteenään Excel-taulukon arvoja ja piirtää niistä kuvan.

Monimutkaisempi komponenttirakenne muodostuu, jos visualisointi tapahtuu usean moduulin yhteistyönä. Tällöin komponentille on määritelty rajapinta, jonka kautta se ottaa syötteet, ja toinen rajapinta, jonka kautta se lähettää tulokset eteenpäin. Tulosten täytyy olla moduulista tullessa joko valmiiksi toisen moduulin hyväksymässä syötemuodossa (Kuva 9) tai vähintään muutettavissa sellaiseksi (Kuva 10).

Kuva 9



Kuva 10

4.1 Visualisointikomponentit

HyperVis-oppimateriaali jakaa visualisointijärjestelmän osat kuuteen ryhmään: käyttäjä, käyttöliittymä, grafiikkajärjestelmä, visualisointitekniikka, tiedon muokkaus ja tiedon lähde. Käyttäjää lukuunottamatta ryhmät edustavat visualisointikomponentteja. Ne ovat järjestelmän osia, jotka saavat syötteen, käsittelevät sen omalla tavallaan ja keskustelevat muiden tuntemiensa komponenttien kanssa. Kutakin ryhmää kohti voi olla useita komponentteja. Visualisointijärjestelmän käyttäjällä tulisi olla mahdollisuus muokata olemassaolevia komponentteja ja luoda itse uusia. [Owen]

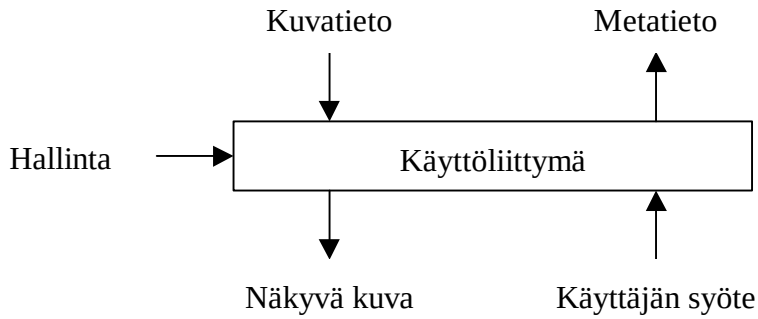
Käyttäjä (Kuva 11) näkee visualisointijärjestelmän käyttöliittymän ja mahdollisen kuvan. Käyttäjä voi muokata visualisointia antamalla järjestelmälle syötteitä käyttöliittymän välityksellä.

Kuva 11

Käyttöliittymä (Kuva 12) sijoittuu komponenttiketjussa käyttäjän ja grafiikkajärjestelmän väliin. Se ottaa vastaan käyttäjän syötteitä ja välittää niitä metatietona grafiikkajärjestelmälle ja sitä kautta muille komponentiketjun jäsenille. Tässä yhteydessä metatiedolla tarkoitetaan komponenttien keskinäistä tiedonvälitystä. Sen avulla komponentit voivat tietää esimerkiksi toistensa tilat ja tilojen muutokset.

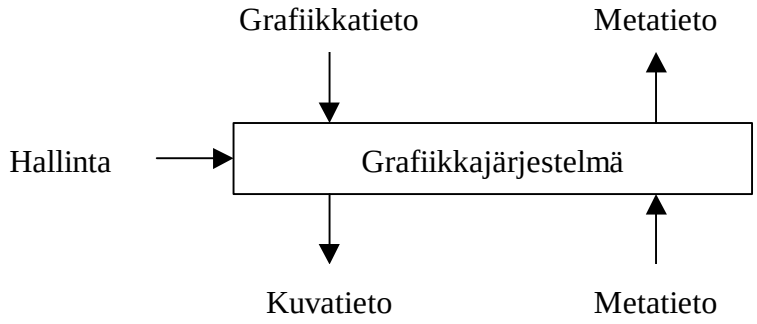
Käyttöliittymä esittää grafiikkajärjestelmältä syötteenä saamansa kuvan käyttäjän tietokoneen näytöllä. Hallinnalla tarkoitetaan sen tahon toimintaa, joka käynnistää komponentin ja ohjaa sen toimintaa.

Kuva 12



Grafiikkajärjestelmä (Kuva 13) sijoittuu komponenttiketjussa käyttöliittymän ja visualisointitekniikan väliin. Grafiikkajärjestelmä saa visualisointitekniikalta syötteenään tietoa, jossa kuvataan näytettävän kuvan geometria. Komponentti muokkaa tiedon käyttäjän ympäristössä esittämiskelpoiseksi kuvaksi esimerkiksi bittikarttamuotoon ja välittää sen eteenpäin. Käyttöliittymältä grafiikkajärjestelmä voi saada syötteenään metatietoa, jonka se välittää visualisointitekniikalle.

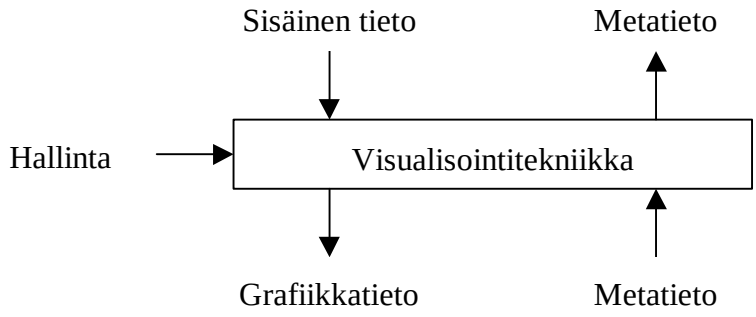
Kuva 13



Visualisointitekniikka (Kuva 14**)** saa tiedon muokkaus –komponentilta järjestelmän sisäisessä siirtomuodossa tietoa, jonka se muokkaa geometrisiksi määrityksiksi. Geometriatieto välitetään eteenpäin grafiikkajärjestelmälle. Geometriatieto voi olla esimerkiksi vektorimuotoista 2- tai 3-ulotteista grafiikkaa.

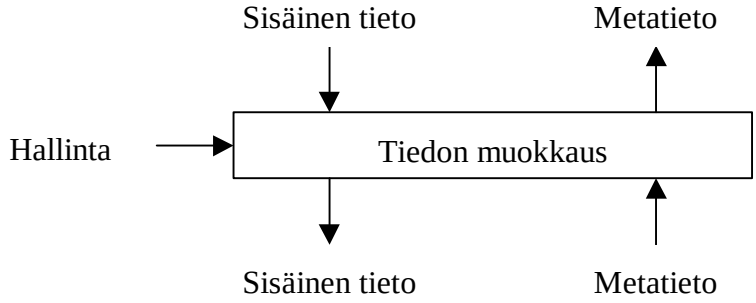
Tämän ryhmän komponentit ovat vastuussa syntyvän kuvan muodosta. Jokin komponentti voi muuttaa tietoa pisteparveksi ja joku toinen kolmiulotteiseksi pinnaksi.

Kuva 14



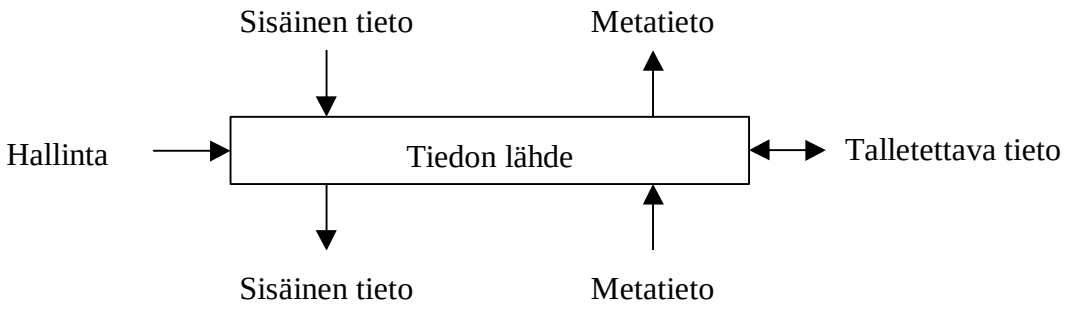
Tiedon muokkaus (Kuva 15) saa syötteenään tietoa sisäisessä siirtomuodossa, muokkaa tietoa ja lähettää sen samassa siirtomuodossa seuraavalle komponentille. Sisäinen siirtomuoto voi olla esimerkiksi lukuja sisältävä matriisi. Tiedon muokkauksella tarkoitetaan esimerkiksi tiedon rajausta tai matemaattisia muunnoksia.

Kuva 15



Tiedon lähde (Kuva 16) on komponenttityyppi, jolla voidaan hakea tietoa esimerkiksi tietokannasta tai tiedostoista. Tieto voi olla myös peräisin simulaattorista, mittalaitteesta tai se voidaan koota useasta lähteestä.

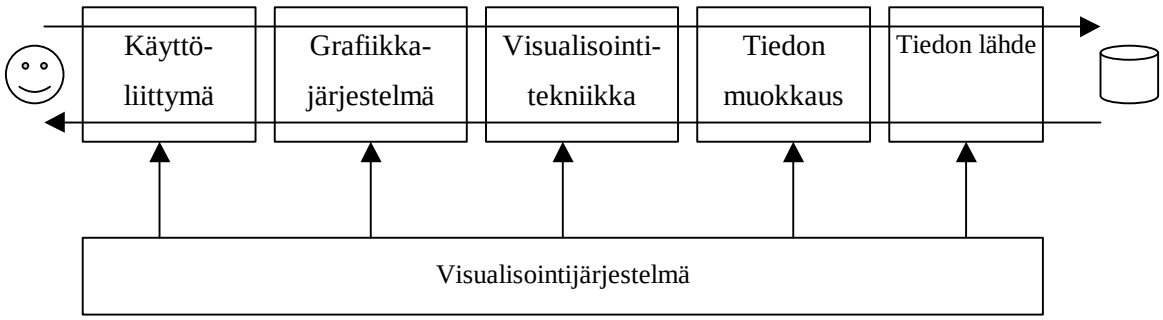
Kuva 16



Edellä esitellyistä komponenteista muodostuu ketju, joka kerää tiedon, muokkaa sen käyttäjän haluamaan muotoon, muuttaa sen geometriaksi ja lopulta näyttää sen kuvana käyttäjän tietokoneen näytöllä. (Kuva 17)

Käyttäjän keskusteluyhteys järjestelmän kanssa muodostetaan yhden komponentin kautta, mistä käyttäjän pyynnöt välitetään metatietona kaikille muille. Koko järjestelmän pohjalla on visualisoinnin sovelluskehys, jolla on hallinnollinen yhteys visualisointikomponentteihin.

Kuva 17

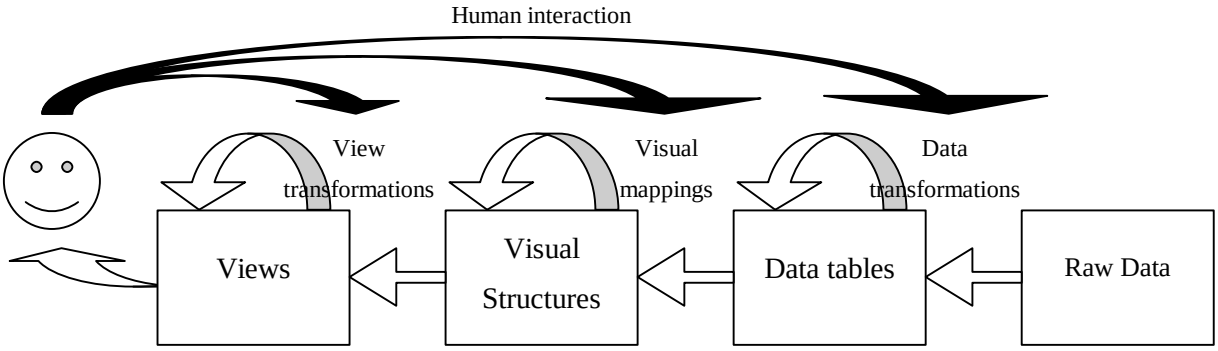


4.2 Visualisoinnin viitekehys

Visualisointikomponenteista koostuu järjestelmä, joka pystyy toteuttamaan visualisointiprosessin kaikki vaiheet. Aiemmin tutustuttiin erilaisiin komponenttityyppeihin ja koottiin niistä visualisoinnin komponenttiketju. Seuraavaksi tutustutaan toiseen malliin, joka pyrkii esittämään yleisellä tasolla komponenteista koostuvan visualisointijärjestelmän rakenteen.

Visualisoinnin viitekehys (reference model for visualization) [Card et al., 17] tarjoaa vaihejakoon ratkaisun, jossa tiedon voidaan käsittää olevan neljässä muodossa. Näiden muotojen välillä tapahtuu muunnoksia, joilla edistetään tiedon saamista lopulliseen tilaan ja visualisoinniksi. Yhden vaiheen sisällä voi tapahtua useita muunnoksia.

Aiemmin esitettyyn malliin (Kuva 17) verrattuna visualisoinnin viitekehys on yksinkertaistettu. Kehyksessä ei ole erillistä käyttöliittymäkomponenttia, vaan jokaisella komponentilla, joka haluaa olla vuorovaikutuksessa käyttäjän kanssa, on oma käyttöliittymä.



Kuva 18

Lähtökohtana kaikelle on visualisoitava tietomassa (raw data). Tietomassa voi sijaita hajautettuna useille palvelimille, monissa eri tietokannoissa tai se voi olla antureilla tosiaikaisesti kerättävää aineistoa.

4.2.1 Tietotaulukoiden luonti

Ensimmäisessä vaiheessa (data transformations) visualisoitavasta tietomassasta rajataan oleellinen osa ja muokataan se taulukkomuotoisiksi rakenteiksi (data tables). Taulurakenteet ovat luonnollinen ja helposti ymmärrettävä tapa jäsentää tietoa. Tämän vuoksi esimerkiksi relaatiotietokannat pohjautuvat taulurakenteisiin.

Tietoalkioihin liittyy varsinaisen arvon lisäksi tietoja alkion merkityksestä. Merkityksellä tarkoitetaan ominaisuuksia, joita alkion arvosta ei välttämättä ole pääteltävissä. Tietoalkiot jakautuvat nominaalisiin, ordinaalisiin ja kvantitatiivisiin tyyppeihin. Kvantitatiivisilla eli mitattavilla arvoilla voidaan tehdä laskutoimituksia ja vertailla alkioita. Kvantitatiivisen

arvon erikoismuoto, suhteellinen arvo, voi olla ainoastaan positiivinen tai nolla. Tähän ryhmään kuuluvat esimerkiksi henkilön ikä ja pituus. Ordinaalisilla arvoilla ryhmitellään alkioita. Esimerkiksi ihmisiä voidaan ryhmitellä sukupuolen perusteella. Nominaalisilla arvoilla määritellään alkiolle tietoja, joiden avulla ei voida tehdä laskutoimituksia eikä varrata alkioita keskenään. Henkilön nimi on tyypillinen nominaalinen arvo.

Seuraavassa taulukossa annetaan esimerkkinä henkilön tietoja, jotka kuuluvat erilaisiin muuttujatyyppeihin.

Kuva 19

Ominaisuus	**Arvo**	**Tyyppi**
Henkilön nimi	Kimmo Pulkkinen	Nominaalinen
Sukupuoli (mies/nainen)	M	Ordinaalinen
Ikä	52	Kvantitatiivinen

Tietotyyppejä on mahdollista muuttaa tietyissä rajoissa. Kvantitatiivinen tieto voidaan muuttaa ordinaaliseksi. Esimerkiksi edellisessä taulukossa henkilön ikä voidaan jakaa ordinaalisiin ryhmiin: nuori, keski-ikäinen, vanha. Myös nominaaliset tiedot ovat muutettavissa ordinaalisiksi. Esimerkiksi nimen perusteella on mahdollista järjestää kaikki henkilöt aakkosjärjestykseen.

Tyypin lisäksi alkioiden arvoilla voi olla muita ominaisuuksia. Kvantitatiivisella arvolla voi olla vaihteluväli ja tarkkuus. Ordinaalisella arvolla on joukko mahdollisia arvoja. Nominaalisella voi olla rajallinen pituus.

Komponentin, joka muuttaa raakatiedon taulukkorakenteiksi, pitäisi säilyttää myös eri tietoalkioihin liittyvä metatieto. Tämän avulla komponenttiketjussa seuraavat visualisointikomponentit osaavat käsitellä tietoalkioita mielekkäin menetelmin.

4.2.2 Tietotaulukoiden muokkaus

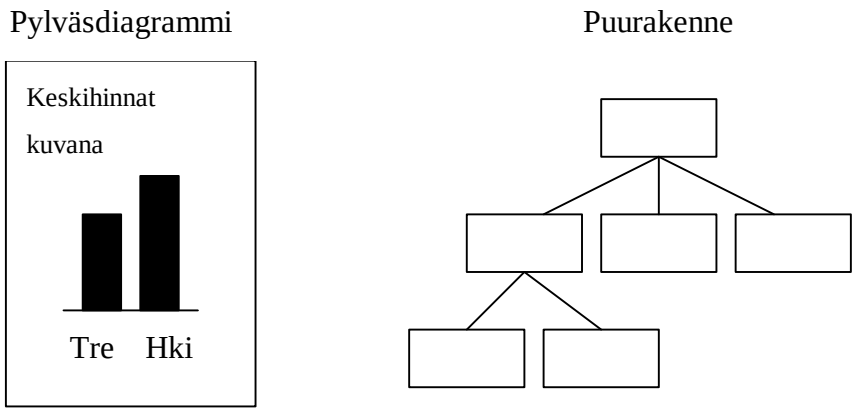
Data transformation –operaatioihin kuuluvat myös tietotaulukoiden muokkaustoiminnot. Muokkauskomponenteilla voidaan esimerkiksi rajata, järjestää tai jäsentää

taulukkomuotoista tietoa. Tietoalkioista voidaan laskea myös johdettuja arvoja. Muokkausoperaatioiden tulee olla sallittuja kohteena oleville tietoalkioille. Esimerkiksi laskutoimitukset sallitaan vain kvantitatiivisille alkioille.

4.2.3 Tiedon muovaus visuaalisiksi rakenteiksi

Toisessa vaiheessa (visual mappings) tietojoukosta muodostetaan visuaalisia rakenteita (visual structures). Prosessissa tieto siirretään muotoon, joka on sidoksissa lopulliseen esitystapaan, oli se sitten pylväsdiagrammi tai vaikka puurakenne (Kuva 20). Tässä vaiheessa on tärkeää tietää millä tavoin käytettävissä oleva tietojoukko on esitettävissä. Osa esitystavoista voi hukata oleellisia asioita tietojoukosta ja osa voi käsitellä tietojoukon alkiota väärillä tekniikoilla. Myös tässä vaiheessa on varmistettava, että oikeille tietotyypeille osataan valita oikeita esitystapoja.

Kuva 20



4.2.4 Näkymän määritykset

Viimeinen vaihe (view transformations) sisältää visuaalisten rakenteiden muunnokset ennen rakenteiden esittämistä kuvana. Yleisin muunnostyyppi on katselukulman säätö. Tähän ryhmään kuuluu rajaus, kohdennus, lähestyminen ja etääntyminen. Toinen muunnostyyppi on kuvan vääristäminen. Kuvan vääristämistä käyttävät hyväkseen linssinäkymät, joissa olennaisin tieto pyritään esittämään muuta tietoa suuremmassa koossa. Kolmanteen muunnostyyppiin kuuluu sijaintien käyttäminen visualisoinnissa, kuten pisteparvien esittäminen.

4.3 Visualisoinnin viitekehyksen soveltaminen komponenttiajatteluun

Kuten aiemmin todettiin, visualisointia voidaan soveltaa mitä moninaisimpiin käyttötarkoituksiin. Toimiakseen useissa eri käyttötarkoituksissa, visualisointijärjestelmältä vaaditaan joustavavuutta. Tämä tarkoittaa sitä, että samalla visualisointivälineellä voidaan tutkia moneen eri aihepiiriin liittyvää tietoa ja toisaalta alkuperäisen tiedon määrä voi olla laajuudeltaan mitä vain.

Monikäyttöisyyden vaatimukset ohjaavat visualisointijärjestelmän rakennetta modulaariseen suuntaan. Visualisointiprosessi voidaan jakaa aliprosesseiksi, joista kukin tapahtuu erillisen ohjelmistomoduulin suorittamana. Nämä moduulit käsittelevät tietojoukkoja ja lähettävät ne yleisessä muodossa seuraaville moduuleille käsiteltäväksi. Päätösvaiheessa tietojoukko on käynyt läpi muokkausvaiheet ja päätyy näkymäksi käyttäjän ruudulle. Tällainen useiden moduuleiden kautta tapahtuva visualisointiprosessi johtaa komponenttipohjaiseen visualisointiin. Komponenttikäsite vahventaa vielä moduulien keskinäistä riippumattomuutta.

Moduulit voivat noudattaa toiminnallisesti aiemmin esiteltyä visualisoinnin viitekehystä. Tätä kautta moduulien tehtävät voidaan jakaa kolmeen ryhmään:

- Data transformations. Komponentit, jotka muodostavat visualisoitavan tietomassan. Tähän liittyy tiedon etsiminen, muotoilu, rajaus ja johdannaistietojen muodostaminen.
- Visual mappings. Komponentit, jotka muodostavat lähtötiedoista tietorakenteita, jotka soveltuvat haluttuihin tiedonkuvaustapoihin.
- View transformations. Komponentit, jotka muokkaavat näkymän tietoja. Tähän liittyy kuvan suurennus, kutistus, venytys, rajaaminen sekä muut esittämiseen liittyvät parametrit.

5 Internet visualisointiympäristönä

Internet on mielenkiintoinen ympäristö visualisointijärjestelmälle. Verkosta on saatavissa suunnattomia määriä tietoa eikä tiedon saanti ole sidottu käyttäjän maantieteelliseen sijaintiin tai muihin ominaisuuksiin. Jokainen voi löytää verkosta tietoa itseään kiinnostavista aihepiireistä. Kiinnostava tieto voi olla saatavissa valmiina kokonaisuutena, mutta monesti sitä täytyy etsiä ja kerätä. Yleensä etsiminen on verkon selaajan tehtävä, mutta hakukoneet ja yhä yleistyvät verkkoagentit pyrkivät tekemään kiinnostavan tiedon hankkimisen käyttäjän puolesta.

Internet saa tiedon lähteenä yhä merkittävämmän aseman jos tämänhetkiset suuntaukset (Kuva 21) jatkuvat. Internetin käyttö lisääntyy jatkuvasti ja WWW-pohjaisten järjestelmien kehitys luo tekniset edellytykset yhä parempien tietovarastojen ja digitaalisten kirjastojen rakentamiselle [Card et al., 409].

Kuva 21

	Suuntaus	**Tulkinta**
1.	WWW	Kaikki tulee olemaan verkossa ja verkkoja on vain yksi.
2.	Digitaaliset kirjastot	Kaikki maailman tietämys tulee olemaan verkossa.
3.	Viestinnän saavutukset	Kehittynyt viestintä poistaa maantieteellisen etäisyyden käyttäjän ja tietolähteen väliltä.

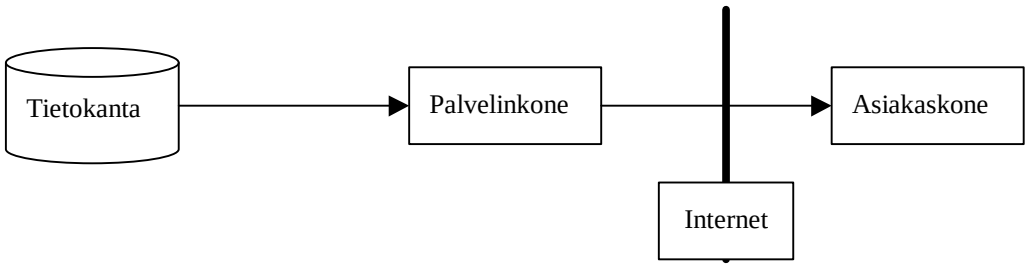
5.1 Visualisointisovelluksen hajautuksen perusteet

WWW-ympäristössä toimiva visualisointi tapahtuu usean tietokoneen yhteistyönä (Kuva 22). Palvelinkoneen tehtävänä on hakea tietovarastosta visualisointiin liittyvä tietomassa. Asiakaskoneen tehtävänä on esittää visualisointi käyttäjälle. Palvelinkoneen ja asiakaskoneen välillä tehtävät voidaan jakaa kahdella tavalla:

1. Palvelinkone tekee pääosan työstä ja asiakaskone vain näyttää visualisoinnin esimerkiksi kuvana. Tällöin asiakassovellus voi olla tavallinen WWW-selain.

2. Palvelinkone vain hakee tarvittavan tietomassan tietovarastosta ja asiakaskone suorittaa tietomassan rajauksen, visualisoinnin muodostamisen ja esittämiseen liittyvät tehtävät. Tällöin asiakassovellus voi olla esimerkiksi WWW-sivulle sijoitettu Java-appletti tai ActiveX-komponentti.

Kuva 22



Edellä kuvatun jaottelun mukaan asiakassovellukset jaetaan monimutkaisuutensa perusteella kahteen ryhmään [Jern1]:

1. Thin visualization clients, eli asiakassovellus, joka vain näyttää palvelimen tekemän visualisoinnin.
2. Fat visualization clients, eli asiakassovellus, joka luo palvelinsovellukselta saamastaan tietomassasta visualisoinnin ja näyttää sen.

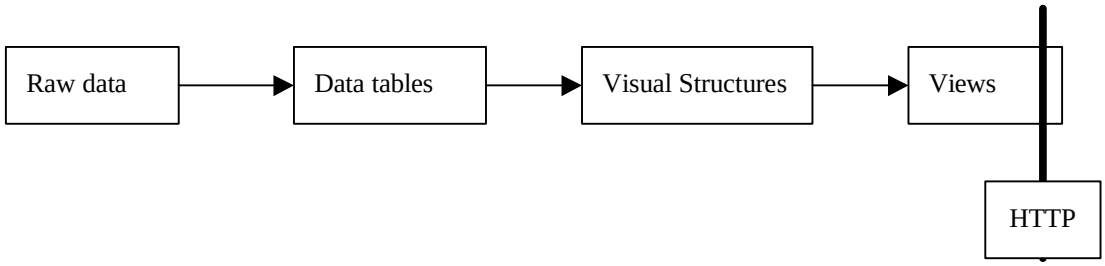
Koska ryhmille ei ole olemassa vakiintuneita suomenkielisiä nimiä, tässä paperissa tullaan käyttämään nimiä kevyt- ja raskas visualisointikomponentti. Kevyt- ja raskas-määreitä on suomen kielessä käytetty vastaavissa tilanteissa ryhmiteltäessä käyttöliittymäkomponentteja.

5.1.1 Kevyet visualisointikomponentit

Kuten aiemmin todettiin kevyellä visualisointikomponentilla tarkoitetaan yksinkertaista sovellusta, joka vain näyttää visualisoinnin tuloksen eli kuvan. WWW-ympäristössä tämä voi tarkoittaa sitä, että visualisointi tapahtuu täysin palvelimella ja käyttäjälle visualisoinnin tulos näkyy kuvana HTML-sivulla. Tällainen ratkaisu tarkoittaa sovelluksen toiminnan hajauttamisen kannalta sitä, että käyttäjä voi keskustella visualisointijärjestelmän kanssa vain WWW-selaimen välityksellä. Tämän seurauksena vuorovaikutteisuus visualisoinnin kanssa ei ole välitöntä vaan jokainen käyttäjän toivoma muutos näkymässä edellyttää selaimelta sivun lataamista uudelleen. [Jern1]

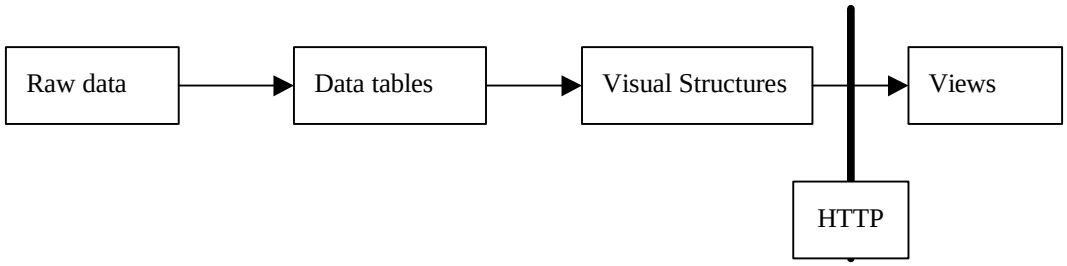
Visualisoinnin viitekehyksessä HTTP-yhteys sijoittuu joko ennen Views-vaihetta tai sen sisälle. Tämä riippuu kyseisen vaiheen toteutuksesta. Mikäli palvelin käsittelee koko visualisointiprosessin ja lähettää asiakkaalle selaimelle vain HTML-sivulla näkyvän kuvan, voidaan katsoa, että HTTP-yhteys tapahtuu Views-vaiheen sisällä (Kuva 23)

Kuva 23



Jos taas palvelimella luodaan HTML-sivu, joka sisältää visualisoitavan tiedon HTML-määrityksinä, kuten taulukoina tai VRML-maailmana, voidaan katsoa, että koko Views-vaihe tapahtuu asiakassovelluksessa. (Kuva 24)

Kuva 24



Palvelimen puolella suoritetaan tiedon kerääminen, muotoilu tietojoukoiksi sekä tietojen siirto visuaalisiksi rakenteiksi. Asiakassovellus tarjoaa vain tavan näyttää visualisoinnissa syntynyt kuva sekä tavan tehdä visualisointipalvelimelle kyselyitä. [Jern1]

Kevyiden visualisointikomponenttien etu on niiden pohjalta rakennettujen järjestelmien helppo ylläpidettävyys. Visualisointi tapahtuu keskitetysti palvelimella ja asiakassovelluksen ominaisuudet ovat hyvin rajatut. Kun visualisointijärjestelmään halutaan lisätä uusia ominaisuuksia, voidaan kaikki muutokset tehdä visualisointipalvelimelle ja asiakassovellusten päivittämiseltä säästytään. Kevyillä visualisointikomponenteilla säästytään myös laitteisto- ja käyttöjärjestelmäriippuvuuksista. Kun visualisoinnin

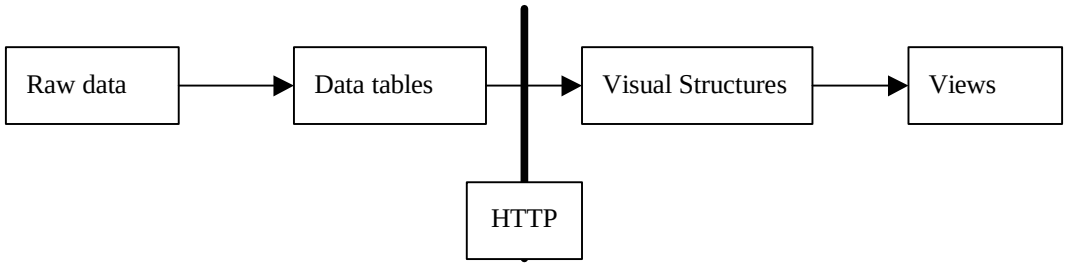
esittäminen ja palvelinkeskustelu hoidetaan tavallisella WWW-selaimella, ei asiakassovellusten toiminta ja kehittäminen ole lainkaan visualisointijärjestelmän tekijän tai ylläpitäjän vastuulla. [Jern1]

5.1.2 Raskaat visualisointikomponentit

Raskaissa visualisointikomponenteissa asiakassovellukselle on annettu merkittävä rooli visualisoinnin muodostamisessa. Asiakassovellus saa palvelimelta esitystavasta riippumattoman tietomassan, käsittelee sen mieleiseensä muotoon visuaalisiksi rakenteiksi ja esittää sen käyttäjälle omalla tavallaan. Raskas visualisointikomponentti toimii vuorovaikutuksessa käyttäjän kanssa. Muutokset visualisoinnin esitystapaan eivät aiheuta yhteydenottoa palvelimeen vaan kaikki yksinkertaisimmat käsittelyt visualisointiin voidaan tehdä asiakassovelluksessa. [Jern1]

Visualisoinnin viitekehyksessä HTTP-yhteys voi sijoittua periaatteessa minkä tahansa vaiheiden väliin. Voi olla, että visualisoinnin asiakassovellus hakee itse raakatietoa eri lähteistä, muodostaa siitä tietorakenteita ja tuottaa lopulta niistä kuvan. Toisaalta raskas visualisointikomponentti voi toteuttaa vain näkymätason ja tarjota sinne yksinkertaisia kuvan käsittelyyn liittyviä mahdollisuuksia. Tyypillinen rajanveto palvelimelle sijoitetun toiminnallisuuden ja asiakassovelluksen välille voisi olla taulurakenteiden ja visuaalisten rakenteiden välissä (Kuva 25). Tällöin palvelimella voidaan rajata visualisoitava tieto ja lähettää asiakassovellukselle vain tarpeellista tietoa. Tiedon määrä on hyvä rajata jo palvelinpuolella, koska tiedon siirto Internetissä on paikalliseen tiedon käsittelyyn verrattuna hidasta ja kallista.

Kuva 25



Raskaita visualisointikomponentteja voi käytännössä toteuttaa Java Applet-tekniikalla, WWW-selaimen plug-in –laajennuksina tai Microsoftin ActiveX-komponenteilla [Jern1, Jern2].

Merkittävimpiä syitä visualisointijärjestelmän toteuttamiseen raskaana visualisointi-komponenttina ovat [Jern1]

- välitön vuorovaikutus käyttäjän kanssa,
- visualisoitavan tiedon käsittely asiakassovelluksessa, sekä
- raskaan visualisointikomponentin kyky esittää näyttävää grafiikkaa.

5.2 Kolmitasoarkkitehtuurin mukainen hajautus

Rakennettaessa visualisointijärjestelmiä, joissa tiedon keruu ja koostaminen käsiteltävään muotoon on monimutkaista, on järkevää soveltaa arkkitehtuurissa kolmitasomallia (Kohta 3.5). Kolmitasoratkaisussa järjestelmä jakautuu osiin käyttöliittymä, järjestelmän toimintalogiikka sekä tietovarastot. Eri osien tehtäväjako voi vaihdella eri visualisointijärjestelmissä. Esimerkiksi raskaalla visualisointikomponentilla käyttöliittymä-tasolle on tuotu toimintoja, jotka kevyissä visualisointikomponenteissa sijaitsevat toimintalogiikka-tasolla.

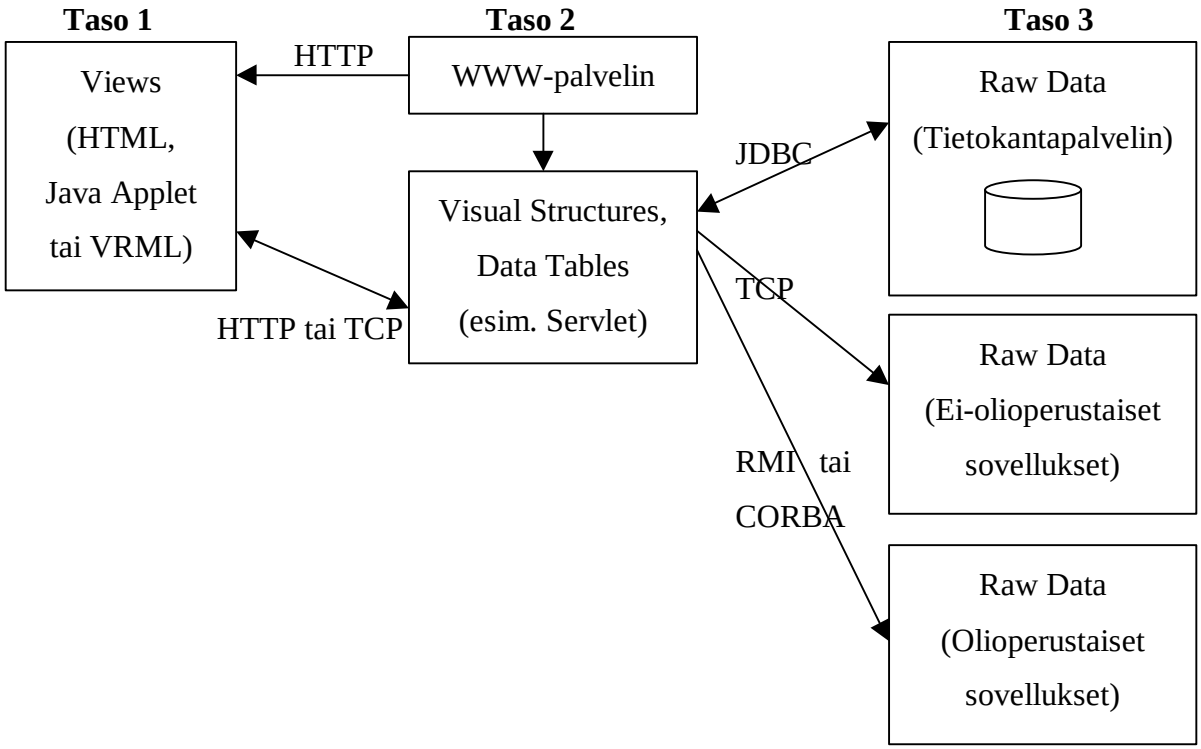
Yksittäisellä arkkitehtuurin tasolla voi olla useita visualisointikomponentteja. Osa niistä huolehtii liikenteestä tasojen välillä ja osa muokkaa visualisoitavaa tietoa kohti esitettävää muotoa. Java-maailmassa luonteva tapa rakentaa komponentteja eri tasoille on JavaBeans-tekniikka.

5.2.1 Kevyet visualisointikomponentit

Käyttöliittymä on visualisointijärjestelmän ikkuna käyttäjälleen. Se esittää visualisoinnin tulokset käyttäjälle ja antaa työkalut visualisointiprosessin ohjaamiseen. Käyttöliittymä saa arkkitehtuurin toiselta tasolta visualisoitavan aineiston. Kun kyseessä on kevyet visualisointikomponentit, aineisto on visuaalisia rakenteita, jotka ovat yksinkertaisia esittää näytöllä. Käyttäjällä voi olla mahdollisuus muuttaa kuvan esittämiseen liittyviä asetuksia suoravaikutteisesti, mutta kaikki visualisoitavan tiedon keräämiseen ja visuaalisten

rakenteiden tuottamiseen liittyvät asetukset välitetään arkkitehtuurin toiselle tasolle. Käyttöliittymän perusta tulee WWW-palvelimelta HTML-dokumenttina. (Kuva 26)

Kuva 26



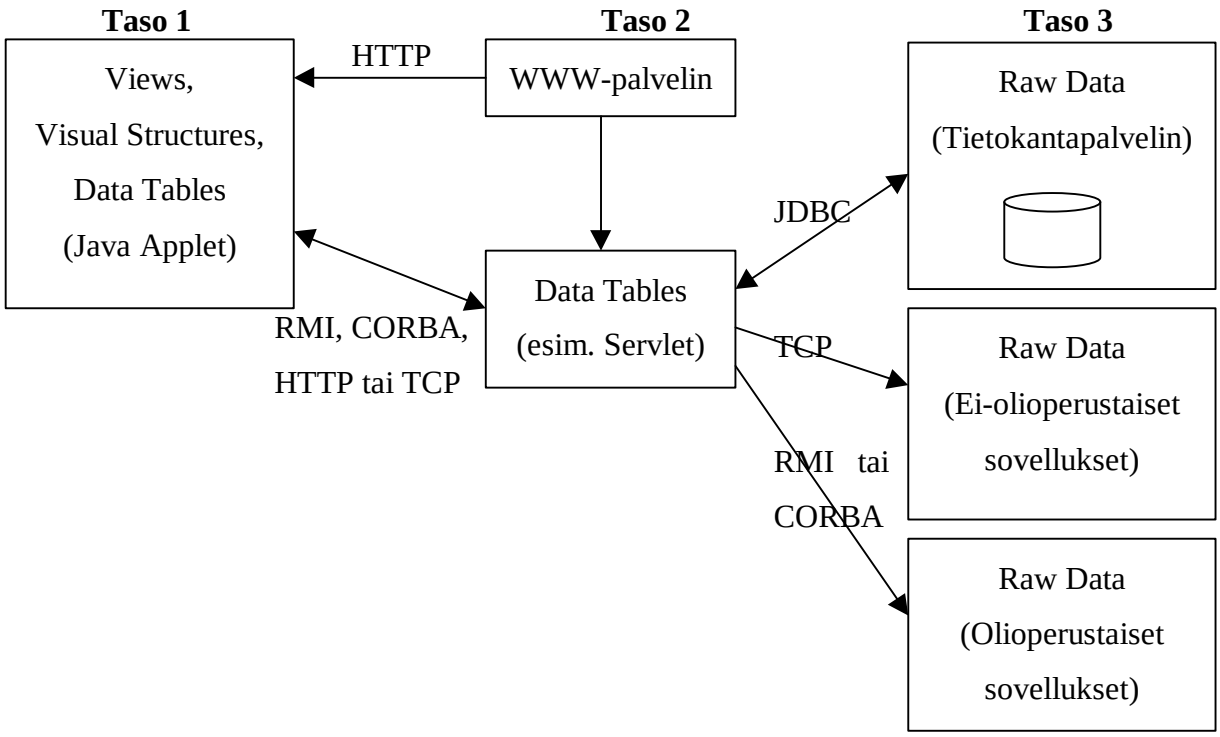
Toisen tason tehtävänä on kerätä kolmannelta tasolta saamaansa tietomassa yhteen ja muuttaa se tietotauluiksi. Tietotauluja voidaan muokata tai rajata. Käsittelyn tuloksena syntyvät tietotaulut muutetaan visuaalisiksi rakenteiksi ja lähetetään ensimmäiselle tasolle esitettäviksi käyttäjälle. Ensimmäisen ja toisen tason välinen yhteys voi olla toteutettu yhteyspohjaisesti HTTP-protokollalla tai TCP-porttiyhteyksillä.

Tiedon keräys tapahtuu kolmannella tasolla. Alkuperäinen tieto voi olla peräisin esimerkiksi tietokannoista, eri sovelluksista tai vaikka antureista. Kolmas taso välittää toiselle tasolle eri lähteistä kerätyn tiedon. Tieto voi tulla eri lähteistä erilaisissa muodoissa. Toisella tasolla tiedon muoto yhdenmukaistetaan tietotauluiksi. Tasojen välinen yhteys voidaan toteuttaa tarpeen mukaan JDBC-tietokantayhteytenä, TCP-porttiyhteytenä tai oliotasolla käyttäen RMI- tai CORBA-tekniikoita.

5.2.2 Raskaat visualisointikomponentit

Raskaissa visualisointikomponenteissa käyttöliittymätasolle on tuotu lisää ominaisuuksia ja toimintalogiikkatasoa on ohennettu. Käyttöliittymätaso saa toiselta tasolta tietorakenteita, joita voidaan vielä rajata tai muokata ennen niiden muuttamista visuaalisiksi rakenteiksi ja käyttäjälle näkyväksi kuvaksi. (Kuva 27)

Kuva 27



Kun kevyiden visualisointikomponenttien kohdalla ensimmäisen ja toisen tason väli tapahtui HTTP- tai TCP-yhteyksillä, on raskaiden visualisointikomponenttien kohdalla monesti järkevää käyttää oliotason yhteyttä. RMI- ja CORBA-tekniikoiden käyttöä hankaloittaa niiden vähäinen tuki selainympäristöissä. Jos visualisointijärjestelmän käyttäjäkunta on rajallinen, voidaan käyttäjiltä vaatia valittua tekniikkaa tukeva selain. Julkisissa palveluissa tällaisia vaateita ei ole mielekästä esittää.

Kun asiakasrajapinta on toteutettu raskaana visualisointikomponenttina on toimintalogiikkakerrosta vastaavasti kutistettu. Tämän tason tehtäväksi jää lähinnä tiedon kerääminen kolmannen tason komponenteilta, tiedon yhdenmukaistaminen tietotauluiksi

sekä alustavien rajausten tekeminen. Vaikka tiedon rajausta tapahtuu myös käyttöliittymätasolla, on alustava rajaus järkevää tehdä toimintalogiikka-tasolla. Tällä vältetään turhaa verkkoliikennettä asiakkaan ja palvelinsovellusten välillä.

Kolmannen tason tehtävä on samanlainen kuin kevyen visualisointikomponentin tapauksessa.

5.3 Yhteenveto

Tässä luvussa tutkittiin Internetiä visualisointiympäristönä ja sovellettiin visualisoinnin viitekehystä hajautettuihin järjestelmiin. Viitekehys oli sovellettavissa hyvin sekä kaksitasoiseen, että kolmitasoiseen arkkitehtuuriin. Visualisointijärjestelmän käyttöliittymä on yleensä vain pinta, joka peittää monelle eri palvelimelle hajautetun taustajärjestelmän. Taustajärjestelmänä toimivat tietolähteet, niistä tietoa kokoavat sovelluksen osat sekä kerätyn tiedon käsittelijät. Visualisointijärjestelmän näkeminen visualisoinnin viitekehyksen toteutuksena helpottaa kokonaisuuden hahmottamista. Erilaiset hajautustavat saattavat jakaa järjestelmän useille eri tyyppisille ja eri paikoissa sijaitseville palvelimille. Järjestelmän osat muodostavat kuitenkin viitekehyksen mukaisen kokonaisuuden.

6 WWW-sivun visualisointikomponenttien ominaisuuksia

Tässä luvussa tullaan käymään läpi WWW-sivulla sijaitsevien visualisointikomponenttien ominaisuuksia. Ominaisuuksia ovat esimerkiksi valitut toteutustekniikat sekä rakenteelliset ratkaisut. Visualisointikomponenttien ominaisuuksia pyritään esittelemään sitoutumatta tiettyihin teknisiin ratkaisuihin. Osassa kohdista esitetyt ominaisuudet liittyvät Java-teknologialla toteutettuihin visualisointikomponentteihin. Tämä siksi, että kyseisten ominaisuuksien ratkaisumallit ovat riippuvaisia toteutustekniikasta.

6.1 Toteutustekniikka

Verkkoselainympäristössä visualisointikomponentin toteuttamiseen voidaan käyttää Java-appletteja, selainlaajennuksia (plug-in) tai ActiveX-komponentteja.

Selainlaajennusten ja ActiveX-komponenttejen etuna on nopeus. Tekniikoiden kesken nopeuseroja on sekä sovelluksen käynnistymisessä, että sovelluksen suorituksessa. Koska selainlaajennukset asennetaan käyttäjän koneelle, niiden käynnistyminen on nopeaa. Appletit ja ActiveX-komponentit sen sijaan eivät asennu käyttäjän koneelle vaan ne joudutaan käynnistyksen yhteydessä lataamaan palvelimelta. Suoritusnopeudessa Java on vaihtoehdoista hitain, koska Java-sovelluksia ei ajeta käyttöjärjestelmän natiiviympäristössä vaan sen päälle rakennetussa virtuaalikoneessa. Javan etuna muihin vaihtoehtoihin nähden on Java-applettien yhteensopivuus eri käyttöjärjestelmien ja laiteympäristöjen kanssa.

6.1.1 Java Applet

Applet-tekniikkaa on tuettu selaimissa jo usean vuoden ajan. Appletteina voidaan toteuttaa WWW-sivulle suuriakin sovelluksia. Sovellukset voivat rakentua yleiskäyttöisten komponenttimallien, kuten JavaBeans, mukaisesti.

Aina kun visualisointiappletin sisältävä WWW-sivu aukeaa, selain lataa appletin Java-luokat palvelimelta. Jos appletti on kookas tai verkkoyhteys on heikko, voi sovelluksen käynnistuminen kestää kauan. Latautumisen hitautta voidaan pitää applettien ongelmana [Jern1, Jern2]. Toisaalta asiakassovelluksen päivittäminen on helppoa, kun sovellusta ei asenneta kiinteästi asiakaspäätteille.

6.1.2 Selainlaajennus

Selainlaajennukset (plug-in) ovat asiakkaan laitteisto- ja ohjelmistoympäristössä toimivia natiivisovelluksia. Selainlaajennukset asennetaan osaksi selainsovellusta. Verkkoselainten mukana toimitetaan nykyään yleisimpiä laajennuksia, kuten 3-ulotteisia maailmoja mallintava VRML-selain.

Selainlaajennusten ongelmana on se, että niiden käyttäminen vaatii erillisten laajennusmoduulien asentamista WWW-selaimeen. Nykyaikainen WWW-selain pystyy automaattisesti asentamaan puuttuvat selainlaajennukset. Jos esimerkiksi tullaan sivulle, joka sisältää Shockwave Flash –animaatioita, osaa selain asentaa itseensä laajennuksen, joka pystyy esittämään animaatiot. Ei kuitenkaan voida olettaa, että kaikilla käyttäjillä on vaadittavat selainlaajennukset asennettuina. Selainlaajennukset ovat järjestelmä- ja selainriippuvaisia.

6.1.3 ActiveX

ActiveX-komponentit ovat Microsoftin määrittelemän komponenttimallin mukaisia sovelluksia, joita voidaan ajaa paitsi WWW-ympäristössä, myös MS Office -sovelluksissa. ActiveX toimii Windows-käyttöjärjestelmässä sekä Macintosh-tietokoneissa.

ActiveX-tekniikan varjopuolena Java-appletteihin verrattuna on tekniikan sitoutuminen Microsoftin tukemiin ympäristöihin. Kun Java-appletit perustuvat avoimiin tekniikoihin ja yksinkertaisiin rajapintoihin, löytyvät ActiveX:n taustalta Microsoft Windowsista lähtöisin olevat suljetut komponenttimallit ja käyttöliittymäkirjastot.

6.2 Komponenttien järjestäytyminen

Komponenttipohjainen visualisointijärjestelmä voi olla rakenteeltaan joko dynaaminen tai staattinen:

1. Rakenteeltaan staattinen järjestelmä koostuu tietystä joukosta keskenään yhteensopivia visualisointikomponentteja, joiden keskinäiset suhteet on ennalta määritelty.
2. Rakenteeltaan dynaaminen järjestelmä koostuu sekalaisesta joukosta keskenään yhteensopivia visualisointikomponentteja, joiden keskinäiset suhteet eivät ole kiinteästi

määritellyt. Käyttäjällä voi olla mahdollisuus määritellä komponenttien suhteet. Joissain järjestelmissä komponentit osaavat järjestäytyä itse.

Monesti visualisointikomponentteja ympäröi konkreettinen visualisointijärjestelmä, joka mahdollistaa tietovirtojen ja niihin liittyvien komponenttien määrittelyn. Visualisointijärjestelmä tarjoaa myös yhdenmukaisen käyttöliittymän, jolla voidaan hallita järjestelmän komponentteja.

Yhden tason sovelluksissa komponenttien järjestäminen on suhteellisen yksinkertaista. Esimerkiksi järjestelmässä, jossa kaikki komponentit ovat Java-appletteja, voidaan appletin käynnistysparametreissä määrittää komponenttien keskinäisiä riippuvuuksia. Järjestäytymisen kannalta ongelmallisempia ovat monitasoarkkitehtuurilla toteutetut järjestelmät. Tällaisessa järjestelmässä voi olla visualisointikomponentteja useilla eri palvelimilla. Hajautettujen järjestelmien hallinta onnistuu parhaiten yhtenäisen käyttöliittymän kautta. Käyttöliittymän pitää mahdollistaa tietovirtojen ja komponenttien järjestämisen kaikilla arkkitehtuurin tasoilla sekä komponenttien asetusten muuttamisen.

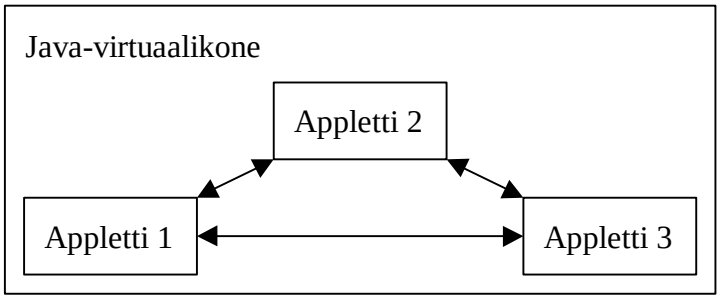
6.3 Komponenttien tutustuminen

Rakenteeltaan dynaamisessa järjestelmässä saattaa komponenteilla olla tarve ilmoittaa toisille komponenteille olemassaolostaan. Toteutuksessa, jossa komponentteja ympäröi konkreettinen visualisointisovellus, komponenttien ilmoittautumista ei tarvita. Sen sijaan samassa virtuaalikoneessa ilman yhteistä kehystä sijaitsevien visualisointikomponenttien täytyy löytää toisensa. Tyypillinen ympäristö, jossa visualisointikomponentteja voi toimia ilman yhteistä visualisointisovellusta, on WWW-sivu. Seuraavassa esitellään kolme erilaista tapaa muodostaa yhteys komponenttien välille.

6.3.1 AppletContext

Samalla sivulla toimivat appletit voivat keskustella keskenään (Kuva 28). Yhteys tapahtuu AppletContext-luokan kautta. Kun halutaan tutkia kaikki sivulla toimivat visualisointiappletit, voidaan pyytää AppletContext-oliolta lista sivulla toimivista appleteista ja testata ovatko ne visualisointiappletteja. Kaikki visualisointiappletit periytetään yhteisestä yläluokasta.

Kuva 28

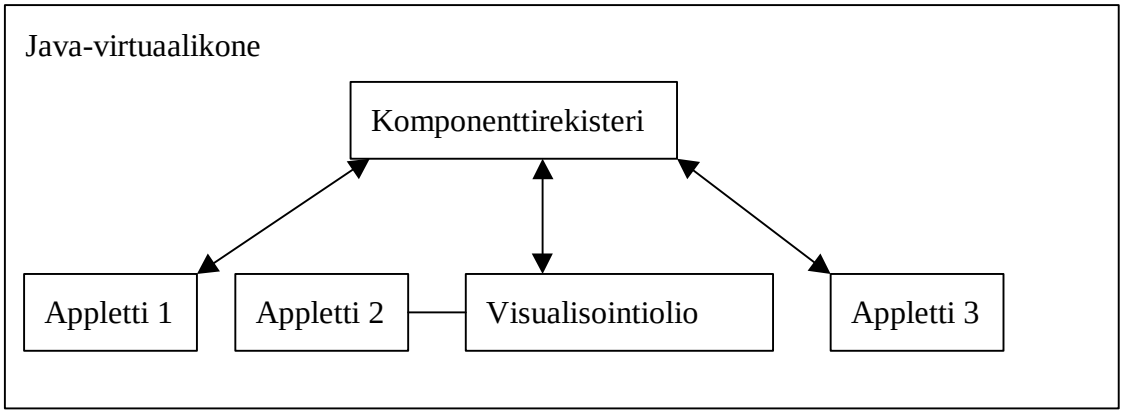


Tässä ratkaisussa kaikkien visualisointikomponenttien pitää olla samalla HTML-sivulla ja kaikkien komponenttien tulee olla Java-appletteja, joilla on visualisointimetodit tarjoava yläluokka. Tällä tavoin on toteutettu myöhemmin esiteltävän Live Documents –järjestelmän komponenttien välinen tutustuminen (Kohta 7.1).

6.3.2 Ainokainen virtuaalikoneessa

Toinen vaihtoehto muodostaa yhteys komponenttien välille on luoda rekisteri, johon jokainen visualisointikomponentti käynnistyessään ilmoittautuu (Kuva 29). Kuollessaan visualisointikomponenttien pitää niinikään poistua rekisteristä. Rekistereitä on kerrallaan olemassa korkeintaan yksi, minkä vuoksi rakenne on toteutus suunnittelumallille ainokainen (singleton) [Gamma et al., 127-134; Koskimies, 31-32]. Tämän tyyppinen keskusteluyhteys on käytössä kohdassa 7.3 esiteltävässä Dove-järjestelmässä.

Kuva 29



Rekisteri voidaan toteuttaa staattisena luokkana, jolla on metodit ilmoittautumista, poistumista ja ilmoittautuneiden visualisointikomponenttien listaamista varten. Kaikki visualisointikomponentit periytetään yhteisestä yläluokasta.

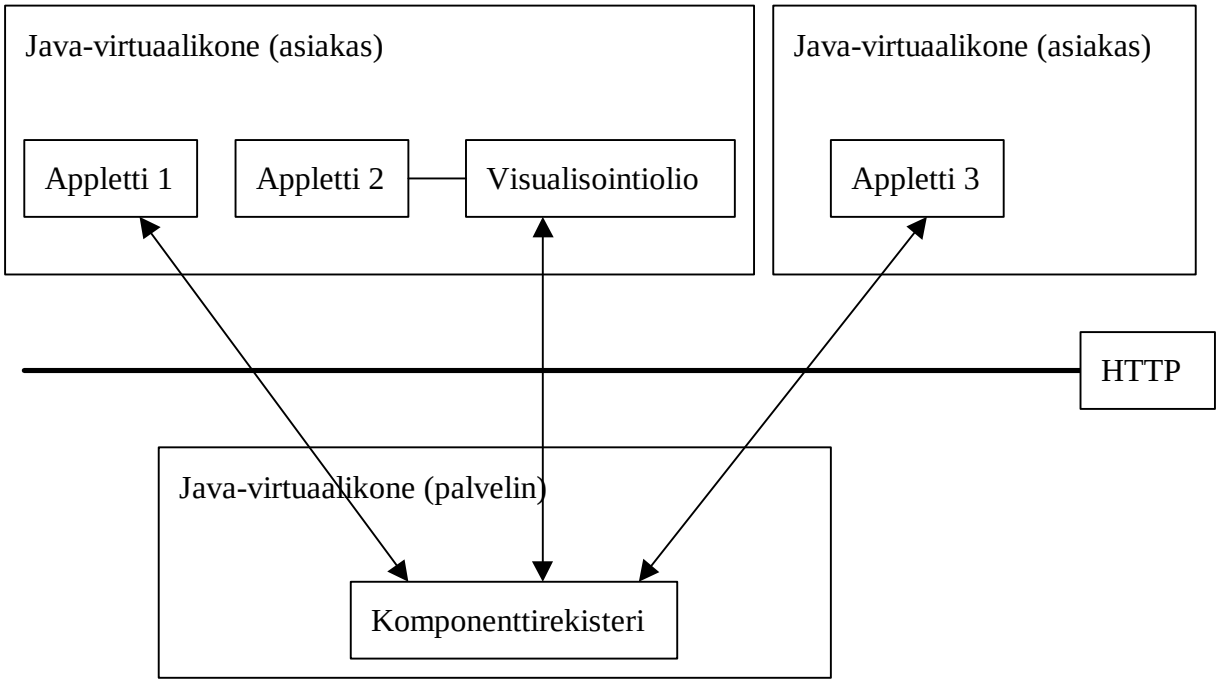
Tässä ratkaisusta kaikkien visualisointikomponenttien pitää olla samassa virtuaalikoneessa. Visualisointikomponentit voivat olla mitä tahansa luokkia, kunhan niillä on visualisointimetodit tarjoava yläluokka. Tekniikka toimii siis muissakin kuin Applet-ympäristöissä.

6.3.3 Ilmoittautuminen palvelimelle

Kolmas ratkaisu on toteuttaa palvelimena rekisteri, johon jokainen visualisointikomponentti ilmoittautuu (Kuva 30). Kuollessaan komponentin täytyy poistua rekisteristä. Palvelin toteuttaa saman toiminnallisuuden kuin edellisessä kohdassa kuvattu staattinen luokka. Palvelinratkaisussa on mahdollista, että verkko-ongelmien vuoksi ilmoittautumis- ja poistumisilmoitukset eivät pääse aina perille. Tämä on toteutuksessa otettava huomioon.

Asiakas- ja palvelinsovellusten välisessä keskustelussa voidaan käyttää HTTP- tai TCP-porttiyhteyksiä. Vaihtoehtoisesti yhteys voidaan muodostaa yleisillä hajautustekniikoilla, kuten RMI:llä tai CORBA:lla.

Kuva 30



Tämän ratkaisun etuna on, että visualisointikomponentit voidaan hajauttaa useisiin virtuaalikoneisiin ja useisiin tietokoneisiin. Tekniikalla voidaan myös luoda yhteisiä näkymiä usealle samanaikaisesti työskentelevälle tutkijalle ja visualisointikomponenttien tilat ovat helposti tallennettavissa. Visualisointikomponentit voivat olla mitä tahansa visualisointimetodit tarjoavasta yläluokasta perittyjä olioita. Ratkaisu ei sovellu tilanteisiin, joissa samassa virtuaalikoneessa toimivat komponentit tarvitsevat nopean keskusteluyhteyden.

Tässä kuvatulla tavalla on toteutettu Sluice-visualisointijärjestelmien keskinäinen viestintä, järjestelmän tilan tallennus ja tosiaikaiset työryhmäominaisuudet. (Kohta 7.2)

6.3.4 Staattinen rakenteen määrittely

Applettiympäristössä luonnollinen tapa määritellä visualisointikomponenttien keskinäiset suhteet on käyttää appletin HTML-parametreja. Appletti saa HTML-parametrinsä ”nimi=arvo”–muodossa.

Parametreissä voidaan määritellä appleteille sekä oma nimi, että niiden applettien nimet, joiden käskyjä kuunnellaan. Toinen tapa on määrittää käskytyypit joita kuunnellaan. Tätä tapaa käytetään Live Documents -järjestelmässä, joka on esitelty kohdassa 7.1.

Edellä esitetty tapa toimii silloin kun kaikki visualisointikomponentit ovat appletteja.

6.4 Komponenttien keskustelu

Tutustumisen jälkeen järjestelmän kaikki komponentit ovat toistensa tiedossa. Seuraavassa vaiheessa alkaa komponenttien yhteistoiminta, joka perustuu komponenttien väliseen keskusteluun. Keskusteluyhteys on olemassa niin pitkään kuin yksikin visualisointikomponentti on toiminnassa.

Kun komponentille tulee tarve ottaa yhteys toisiin komponentteihin, yhteydenmuodostus tapahtuu AppletContext-olion, staattisen luokan tai palvelimen kautta, riippuen siitä, mikä toteutustapa yhteyden hallintaan on kohdan 6.3 vaihtoehdoista valittu. Yhteydenotto voi olla joko ilmoitus komponentissa syntyneestä tapahtumasta tai kysely mahdollisista tapahtumista muissa järjestelmän komponenteissa.

Vaihtoehtoja tapahtumanvälitykseen on siis kaksi:

- Komponentti ilmoittaa tapahtumista tai visualisoitavan tiedon muutoksista aina kun niitä tulee (tosiaikainen).
- Komponentti kysyy tietyin väliajoin onko muissa komponenteissa syntynyt tapahtumia tai onko visualisoitava tieto muuttunut (ajastettu).

Ensimmäinen vaihtoehto sopii hyvin järjestelmään, jossa kaikki komponentit toimivat samassa virtuaalikoneessa. Muutos yhdessä komponentissa vaikuttaa välittömästi kaikkiin muihinkin komponentteihin. Mikäli komponenttien välinen keskustelu tapahtuu palvelimen välityksellä, ensimmäinen tapa ei ole mahdollinen. Tämä johtuu siitä, että palvelin ei voi ottaa yhteyttä applettiin, vaan yhteydenoton täytyy aina tapahtua appletilta palvelimeen päin.

6.5 Tiedonsiirtomuoto

Tiedonsiirtomuodon valinnalla otetaan kantaa siihen, minkä tyyppistä syötettä komponentit saavat. Syöte voi olla sisällöltään kahta tyyppiä:

1. visualisoitava data, ja
2. visualisointikomponenttiin liittyvät ohjauskäskyt.

Edellisessä kohdassa kerrottiin kuinka komponentit voivat välittää tietoa toisilleen. Tässä kohdassa esitellään muotoja, joissa tieto voidaan välittää. Siirtomuotoja ovat esimerkiksi Java-oliot, XML-tietueet ja erotinmerkeillä jäsennetyt tekstitiedostot.

Java-olioiden siirtäminen yhden Java-virtuaalikoneen sisällä on yksinkertaista. Sen sijaan virtuaalikoneiden välinen tiedonsiirto täytyy toteuttaa yleisillä hajautustekniikoilla kuten RMI:llä tai CORBA:lla. Muut seuraavissa kohdissa esiteltävät tiedonsiirtomuodot voidaan välittää myös HTTP-protokollalla tai TCP-porttiyhteyksillä.

6.5.1 ASCII-taulukointi

ASCII-pohjaisella taulukolla voidaan esittää matriisimuotoinen tietomassa. Taulukolle on määritelty sarakkeita ja rivejä erottavat erotinmerkit. Sarakkeiden ja rivien otsikot voidaan esittää osana taulukkoa.

Esimerkki 1 – ASCII-taulukko, jossa sarakkeita erottaa puolipiste (;) ja rivejä ruutu (#):

Kuukausi;Kuukausittainen sademäärä #Tammikuu ;10 mm #Helmikuu; 20 mm

ASCII-taulukoinnin etuna on tiedon tiiviys. Siirrossa välitetään ainoastaan visualisoitavaa tietoa ja erotinmerkkejä. Tämän siirtomuodon etuna on myös siirrettävän tiedon selväkielisyys ja riippumattomuus ohjelmointikielistä ja komponenttien toteutustavoista.

Tällä esitystavalla voidaan esittää taulukkomuotoista tietoa ja taulukon sarakkeisiin liittyvää metatietoa, kuten solujen tietotyypit tai mahdolliset arvojoukot. Visualisointikomponenteille tarkoitetut ohjauskomennot sopivat heikosti tähän esitystapaan.

6.5.2 Pakattu binäärimuoto

Pakatulla binäärisiirrolla voidaan välittää ASCII-taulukko vielä huomattavasti tiiviimmässä muodossa. Tällöin tiedon lähettäjä pakkaa taulukon määritellyllä algoritmilla ja tiedon vastaanottaja vastaavasti purkaa sen alkuperäiseen muotoonsa.

Pakatu binäärisiirron suurimpana etuna on esitetyistä vaihtoehdoista paras tiedon tiiviys. Tätä kautta tiedon siirto on erittäin nopeaa. Sen sijaan aikaa kuluttavaa on tiedon pakkaaminen ja purkaminen. Tämä siirtomuoto on siis verkkoliikenteen kannalta nopea, mutta tiedon prosessoinnin kannalta hidas. Binäärimuodossakin säilytetään riippumattomuus ohjelmointikielistä ja komponenttien toteutustavoista. Verrattuna ASCII-taulukointiin pakatussa binäärimuodossa menetetään tiedon selväkielisyys.

6.5.3 Nimi-arvo –parit

Tässä esitystavassa siirrettävä tieto käsitetään joukkona tietoalkioita, joista jokaiselle on olemassa yksikäsitteinen nimi.

Esimerkki 2:

Graafi1_otsikko = ”Kuukausittainen sademäärä”
Graafi1_piste1_otsikko = ”Tammikuu”
Graafi1_piste1_arvo = ”10 mm”
Graafi1_piste2_otsikko = ”Helmikuu”
Graafi1_piste2_arvo = ”20 mm”

Tämän siirtomuodon suurin etu on siirrettävän tiedon selkeys. Jokaisen arvon merkitys on helposti ymmärrettävissä ja visualisoitavaa tietoa ja komponentin ohjauskomentoja on helppo kirjoittaa vaikka käsin. Nimi-arvo –parien käyttö ei myöskään aseta vaatimuksia ohjelmointikielelle tai komponenttien toteutustavoille. Tällä esitystavalla voidaan esittää luontevasti taulukkoon liittyvää metatietoa, kuten solujen tietotyyppejä tai mahdollisia arvojoukkoja. Myös visualisointikomponenteille tarkoitetut ohjauskomennot sopivat tähän esitystapaan.

Nimi-arvo –parien suurin ongelma on tiedon väljyys. Jokaista tietoalkiota kohti on olemassa yksikäsitteinen otsikko, joka voi muodostua hyvinkin pitkäksi. Siirrettävästä tietomassasta alle puolet saattaa olla visualisoitavaa tietoa.

6.5.4 Applet-parametrit

Applet-parametreillä tarkoitetaan HTML-kielessä määriteltyä tapaa välittää Java-appleteille parametrejä. Parametrit luetellaan <APPLET>-määreiden välissä nimi-arvo –pareina. Parametrivälityksen perusajatus on sama kuin edellisessä kohdassa, mutta esitystapa on vielä väljempi.

Esimerkki 3:

```
<applet code=”Appletti.class” width=100 height=100>
<param name=”Graafi1_otsikko” value=”Kuukausittainen sademäärä”>
<param name=”Graafi1_piste1_otsikko” value=”Tammikuu”
<param name=”Graafi1_piste1_arvo” value=”10 mm”
<param name=”Graafi1_piste2_otsikko” value=”Helmikuu”
<param name=”Graafi1_piste2_arvo” value=”20 mm”
</applet>
```

Tämä siirtomuoto on ominaisuuksiltaan samanlainen kuin edellinenkin.

Siirtomuodon väljyys on tässä esiteltävistä muodoista väljin. Jokaista tietoalkiota kohti on olemassa yksikäsitteinen otsikko, joka voi muodostua hyvinkin pitkäksi. Lisäksi siirtomuodossa käytettävä <PARAM>-määre on esitystavaltaan hyvin lavea. Kuten esimerkistä ilmenee, siirrettävästä tietomassasta vain pieni osa on visualisoitavaa tietoa.

6.5.5 XML

XML-siirtomuodolla voidaan välittää monimutkaisempiakin tietorakenteita. XML:n ominaisuuksiin kuuluu tiedon eheyden varmistaminen, mikä pitää sisällään tietotyypit ja mahdolliset arvojoukot. Kaikki komponentin hyväksymät parametrit ja niiden mahdolliset arvojoukot voidaan määritellä ns. DTD-tiedostoon (Document Type Definition). XML-tiedostoja käsittelevät sovellukset vertaavat saamaansa XML-syötettä DTD:ssä määriteltyyn kehykseen ja jos syöte on kehyksen mukainen, syöte on käsiteltävissä.

Esimerkki 4 - XML-tietue:

```
<graafi>
<riviotsikot>
<rivi numero=”1”>Tammikuu</piste>
<rivi numero=”2”>Helmikuu</piste>
</riviotsikot>
<sarake numero=”1”>
<otsikko>Kuukausittainen sademäärä</otsikko>
<rivi numero=”1”>10 mm</piste>
<rivi numero=”2”>20 mm</piste>
</graafi>
```

Myös XML-siirtomuodossa tieto on helposti ymmärrettävää. Tieto on jäsennetty selkeisiin kokonaisuuksiin ja komponentin ymmärtämä kielioppi on dokumentoituna DTD-tiedostoon. XML-tiedostojen välittämiseen, manipulointiin ja käyttöön on olemassa runsaasti valmiita ohjelmakomponentteja ja tekniikoita. XML ei sido toteutusta mihinkään ohjelmointikieleen. Tällä esitystavalla voidaan esittää luontevasti taulukkoon liittyvää metatietoa, kuten solujen tietotyyppejä tai mahdollisia arvojoukkoja. Myös visualisointikomponenteille tarkoitetut ohjauskomennot sopivat tähän esitystapaan.

XML-tiedostojen käsittely on suhteellisen hidasta ja kuluttaa runsaasti sekä prosessoriaikaa, että muistia. Tämä on merkityksellinen seikka lähinnä ympäristössä, jossa kuljetetaan suuria määriä visualisoitavaa dataa, ja siirtoa tapahtuu jatkuvasti.

6.5.6 Oliot

Olioiden siirtäminen on nopea, mutta hyvin ohjelmointiläheinen ratkaisutapa. Olioiden siirto onnistuu joko yhdellä ohjelmointikielellä toteutettujen komponenttien välillä tai käyttämällä yleistä hajautustekniikkaa, kuten CORBA, useiden eri kielien välillä. Siirtoon käytetään rakenteeltaan tietyn muotoista oliota. Rakenne määritellään rajapinnaksi, joka on kaikkien sovelluskomponenttien tiedossa.

Tiedon siirto olioina on tyypillistä eri palvelinten välisissä yhteyksissä. Esimerkiksi kolmitasoarkkitehtuurissa tasojen kaksi ja kolme välinen yhteys toteutetaan yleensä RMI- tai CORBA-tekniikalla. Mikäli asiakassovellus toteutetaan Java-applettina, RMI:n ja CORBA:n käyttö ei ole suositeltavaa eikä aina mahdollistakaan. Tämä johtuu siitä, että kaikkien selainten Java-laajennukset eivät mahdollista näiden hajautustekniikoiden käyttöä. Java-sovelluksien kohdalla tällaisia ongelmia ei ole.

6.6 Komponentin lähtötiedot

Java Applet -tekniikalla toteutettu komponentti saa HTML-sivulta käynnistysparametrinsa, kuten kohdassa 6.5.4 on kuvattu. Parametreissä komponentille voidaan kertoa joko kaikki appletin tarvitsemat lähtötiedot tai kohde, josta tiedot on haettavissa. Kun komponentin tarvitsemat lähtötiedot ovat vähäiset, on järkevää sijoittaa kaikki käynnistysparametrit HTML-sivulle. Tällä tavoin joudutaan sivunlatauksen yhteydessä tekemään ainoastaan yksi HTTP-pyyntö.

Kun komponentille halutaan välittää paljon tietoa, on mielekästä käyttää HTML-parametrejä tiiviimpää esitystapaa ja hakea parametrit erillisellä pyynnöllä. Tällöin on mahdollista käyttää mitä tahansa kohdassa 6.5 esiteltyä tietonsiirtomuotoa. Tieto voidaan hakea joko palvelimella sijaitsevasta tiedostosta tai halutusta URL-osoitteesta. Yhteyksiä voidaan ottaa kuitenkin vain samaan palvelimeen, josta appletti on ladattu. Tästä rajoituksesta on mahdollisuus poiketa, jos annetaan appletille poikkeuksellisia valtuuksia tietoturvan ohittamiseksi.

Monesti visualisointikomponenteilla esitettäviä tietoja halutaan hakea verkkopalveluista kuten tietokannasta tai agenteilta. Tällöin komponentille määritellään URL-osoite, josta näihin lähteisiin on mahdollista päästä.

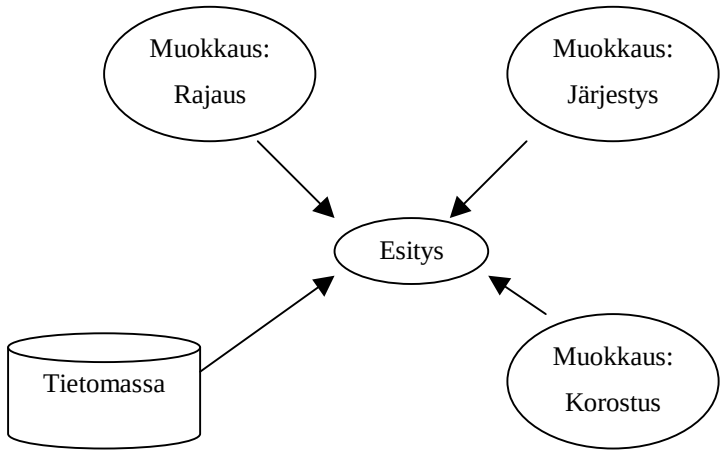
6.7 Komponenttien keskustelusuunta

Usean visualisointikomponentin muodostama visualisointijärjestelmä voi muodostua joko nauhamaisesti visualisointiprosessia edistävästä komponenttiketjusta tai yhdestä keskuskomponentista, jota ympäröi tähtimäisesti usean visualisointia tukevan ja vuorovaikutusta lisäävän komponentin joukko.

6.7.1 Tähtimäinen järjestäytyminen

Tähtimäisessä muodostelmassa visualisointijärjestelmän perustoiminnallisuus on kerätty yhteen komponenttiin. Komponentti saa syötteenään visualisoitavan tiedon, käsittelee tiedon ja muokkaa sen esityskelpoiseen muotoon. Muiden järjestelmään kuuluvien komponenttien tehtävänä on keskuskomponentin rajapintoja käyttämällä muovata visualisoitavaa tietoa käyttäjän haluamilla tavoilla. (Kuva 31)

Kuva 31 – Tähtimäinen visualisointijärjestelmä



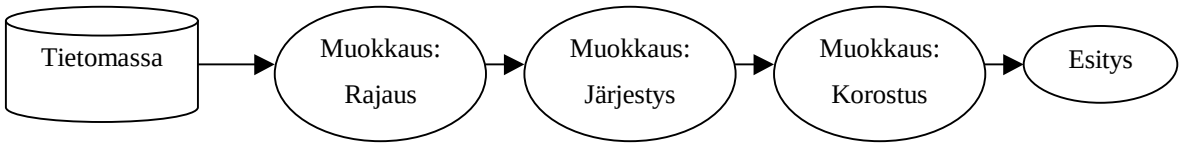
Rajapintamielessä tähtimäinen järjestäytymistapa on hankala, koska avustavien visualisointikomponenttien täytyy pystyä muokkaamaan keskuskomponentin toimintaa monipuolisesti. Mikäli avustavilla komponenteilla halutaan toteuttaa älykkäitä toimintoja, täytyy keskuskomponentin pystyä myös kertomaan tilastaan ja tilansa muutoksista avustaville komponenteille.

Tässä dokumentissa esitellyistä visualisointijärjestelmistä Live Documents (7.1) [Mockus et al.] rakentuu tähtimäisen mallin mukaisesti.

6.7.2 Nauhamainen järjestäytyminen

Tähtimäistä järjestäytymistä yleisempi tapa koota visualisointikomponentit yhteen on nauhamainen järjestäytyminen. Tämä malli (Dataflow) [Card et al., 233] jakaa visualisointiprosessin osatehtäviin, joista jokaista vastaa yksi visualisointikomponentti. Kukin komponentti saa ketjussa edelliseltä komponentilta syötteenään visualisoitavaa tietoa. Yksittäisen komponentin tehtävänä on jalostaa tietoa kohti lopullista tietojoukkoa ja viimeinen komponentti esittää sen käyttäjälle (Kuva 32).

Kuva 32 – Nauhamainen visualisointijärjestelmä



Visualisoinnin viitekehys (4.1) [Card et al., 17] voi olla lähtökohtana nauhamaisen järjestelmän komponentteja määriteltäessä. Kaikkia viitekehyksen vaiheita ei välttämättä toteuteta omina komponentteinaan, ja toisaalta yksi vaihe voidaan jakaa monen komponentin tehtäväksi.

Nauhamaisessa mallissa visualisointiprosessi on helposti nähtävissä ja kukin komponentti toimii ikäänkuin suodattimena. Muutos yhden komponentin toiminnassa vaikuttaa ainoastaan sitä seuraaviin komponentteihin. Yksittäisen komponentin ei siis tarvitse olla perillä muiden kuin sitä edeltävän komponentin tilasta tai sen muutoksista.

Nauhamaisessa komponenttimuodostelmassa tietomassa siirtyy aina edeltävältä komponentilta seuraavalle. Tästä voi seurata prosessoriteho- tai muistivaatimuksia. Jos tiedon siirtomuoto on esimerkiksi XML, joutuu jokainen komponentti ensin lukemaan XML-syötteen tietorakenteeseen, käsittelemään tietoa ja palauttamaan sen taas seuraavaa komponenttia varten XML-muotoiseksi. Tiedon siirto komponentilta toiselle tapahtuu yksinkertaisimmin olioina. Näin rakentuu myös Sluice-visualisointiympäristö [Isenhour et al.], joka esitellään tarkemmin kohdassa 7.2.

Kuten aikaisemmissa luvuissa on nähty, nauhamaisen visualisointiketjun jakaminen monitasoiselle arkkitehtuurille on luontevaa (Kohta 5.2). Komponenttiketjun osat voivat olla sidoksissa toisiinsa niin paikallisen yhteyden kuin hajautustekniikalla toteutetun yhteydenkin kautta.

7 Tutkimuksia visualisointikomponenteista Internet-ympäristössä

WWW-pohjaiseen visualisointiin ja visualisointikomponentteihin liittyviä tutkimuksia on olemassa suuri joukko (Esimerkiksi Dincer et al., Isenhour et al., Jern3, Mockus et al. ja Schmidt). Pääosa tutkimuksista on pyrkinyt kehittämään tiettyä tarkoitusta ja sovellusaluetta tukevaa visualisointijärjestelmää. Sovellusaloista riippumattomista tutkimuksista osa on painottanut komponenttiperustaisten ratkaisujen kehittämistä, ja osa on paneutunut WWW-ympäristön ominaisuuksiin ja alkanut siitä lähtökohdasta miettiä komponenttien yhteyksiä ympäristöönsä ja muihin komponentteihin. WWW-ympäristön ja komponenttiajattelun yhdistävää tutkimusta ei ole juuri tehty.

Seuraavissa kohdissa tutustutaan muutamaan tutkimukseen, jotka ovat syventyneet WWW-ympäristössä tapahtuvaan visualisointiin ja visualisointikomponentteihin. Valituissa tutkimuksissa järjestelmien lähtökohdat poikkeavat hyvin paljon toisistaan. Tämän vuoksi myös esimerkkitutkimuksissa tuotetut ratkaisut ovat hyvin erilaisia.

7.1 Live Document

Live Document –käsitteellä tarkoitetaan dokumenttia, jonka sisältöön lukija voi vaikuttaa dokumentin sisältämillä vuorovaikutteisilla työvälineillä [Mockus et al.]. Kun tavallisesti lukija voi vain selata dokumenttia, antaa Live Document lukijalle mahdollisuuden muokata dokumentissa esitettyjä visualisointeja ja löytää niistä uusia merkityksiä. Dokumentin tuottaja voi antaa lukijalle tarkoituksella tietyn joukon valmiita visualisointeja, mutta myös mahdollisuuden tehdä omia havaintoja ja löytää tiedosta uusia sisältöjä, joita välttämättä edes dokumentin tuottaja ei ole löytänyt. Live Documentin lukija voi siis itse osallistua tutustumansa aiheen tutkimukseen dokumentin tuottajan määrittelemistä lähtökohdista.

Mockus, et al. Tutkivat Live Document –ajatuksen soveltamista WWW-ympäristöön. Lisäksi artikkelissa pyritään ratkaisemaan vuorovaikutteisiin dokumentteihin liittyviä käytettävyysongelmia. Käytettävyyden kannalta ongelmallista on Live Documentin ero tavalliseen kiinteäsisältöiseen artikkeliin. Tavallisesta artikkelista lukijan on helppo selata sisältö ja huomata kiinnostavat, artikkelin kirjoittajan valitsemat näkökulmat. Live Documentissa käyttäjältä vaaditaan kykyä ja kiinnostusta tutkia syvällisemmin artikkeliin

liittyviä visualisointeja sekä käyttää omaa tietämystään ja havainnointikykään saadakseen artikkelista kaiken irti.

Ratkaisumalliksi Mockus, et al. Ovat valinneet HTML-pohjaiset WWW-sivut, joiden sisällä dynaamiset osuudet on toteutettu Java-appleteilla. Live Documentin tekijälle annetaan

- mahdollisuus määritellä visualisointinäkymille tietosisältö,
- työkalut, joilla lukija voi visualisointia muokata,
- esitysvälineet, joilla visualisointi näytetään, sekä
- mekanismi, jolla kaikki edelliset voidaan helposti kytkeä toisiinsa.

Edellisen listan kolme ensimmäistä kohtaa ovat käytännössä Java-appletteja ja neljännellä kohdalla tarkoitetaan applettien keskinäistä kommunikointia. Kommunikointikäytännöt määritellään appletin parametreilla komponentteja ympäröivällä HTML-sivulla. Koska applettien keskinäiset yhteydet on erotettu niiden varsinaisesta toiminnasta, on komponenteissa päästy uudelleenkäytettävyyteen ja yksinkertaisuuteen.

7.1.1 Kommunikointi

Visualisointikomponenttien keskinäiset yhteydet on toteutettu Observer-suunnittelumallin [Gamma et al., 293-303] tyyppisesti määrittelemällä komponenteille toimija-kuuntelija –suhteita. Sivulla sijaitseva appletti voi saada alustustietonaan viestityypin, johon kuuluvia viestejä appletti ottaa vastaan. Alustustiedoissa voidaan määritellä myös viestityyppi, johon kuuluvia viestejä appletti itse lähettää. Vastaanotettavat viestityypit määritellään ”Listen”-parametrilla ja lähetettävät viestityypit ”Inform”-parametrilla. Lähettäjinä voivat toimia vain muokkauskomponentit ja vastaanottajina vain esityskomponentit.

Seuraavassa (Esimerkki 5) esitetään HTML-koodilla kaksi applettia, joista ensimmäinen lähettää viestejä tyypiltään ”Lähettäjän viestit” ja toinen vastaanottaa samantyyppisiä viestejä.

Esimerkki 5

```
<applet code=”Lahettaja.class” width=100 height=100>
<param name=”Inform” value=”Lähettäjän viestit”>
</applet>
```

<applet code=” Vastaanottaja.class width=100 height=100>
<param name=”Listen” value=”Lähettäjän viestit”>
</applet>

Esityskomponentit voidaan yhdistää toisiinsa ”LinkOn”-parametrilla. Jos kahdella esityskomponentilla on sama ”LinkOn”-parametri, näille pätevät aina samat rajaus- ja käsittelyvalinnat.

7.1.2 Tietosisältö

Esityskomponentit hakevat tietosisällön halutulta palvelimelta URL-yhteydellä. Komponentti voi saada parametrin ”url”, jonka arvo kertoo osoitteen, josta tietosisältö on haettavissa. Tiedonsiirtomuoto täytyy olla kaikille tietolähteille ja visualisointikomponenteille yhteinen.

7.1.3 Esitysvälineet

Esityskomponentit voivat toimia joko täysin itsenäisesti tai kuuntelijoina muokkaustyökaluille. Kullakin esityskomponentilla on oma tapansa esittää tietonsa käyttäjälle. Esitettävää tietoa voidaan rajata joko määrittämällä esityskomponentille alustustietoina rajaustapa tai kuuntelemalla toisten sivulla sijaitsevien komponenttien rajauskomentoja. Esitysvälineiden toimintaa voidaan ohjata kuvassa esitetyillä parametreilla (Kuva 33).

Kuva 33

Komento	**Merkitys**
url	Osoite, josta esitettävä tieto haetaan
Select	Valitsee esitettävät osan tietojoukkoa
Variable	Määrittelee näytettävät tietokentät
SortBy	Esityksen järjestystapa
Highlight	Valitsee tietojoukon osan, joka esitetään korostettuna

Transform	Määrittelee visualisoinnin esitysmuodon
Listen	Määrittelee kuunneltavan viestityypin
LinkOn	Määrittelee esityskomponenttien keskinäiset yhteydet

Esityskomponentti voi jo sinänsä sisältää mahdollisuuksia visualisoinnin muokkaamiseen ja esitettävän tiedon rajaamiseen. Näitä muutoksia ei kuitenkaan voida välittää toisille komponenteille. Esimerkiksi esityskomponentissa tehdyt valinnat eivät näy muutoksina muokkauskomponenteissa.

7.1.4 Muokkaustyökalut

Muokkauskomponenttien tehtävänä on vaikuttaa esityskomponenttien tilaan. Tilanmuutokset tapahtuvat muokkauskomponenttien esityskomponenteille lähettämien viestien perusteella. Muokkauskomponenteista järjestelmään on toteutettu yleisimmät valikkotyypit kuten pudotusvalikot ja listavalikot. Seuraavassa esimerkissä [Esimerkki 6] on muokkauskomponentti, joka näkyy käyttäjälle järjestystapojen pudotusvalikkona. Pudotusvalikko on tyypiltään ”Choice”. Kun listasta tehdään valinta, komponentti lähettää kuuntelijoilleen käskyn muuttaa järjestys uuden valinnan mukaiseksi. Oletuksena on valittuna ”Increasing order” eli nouseva järjestys.

Esimerkissä siis ”Choice” määrittelee, että valikon ulkoasu on pudotusvalikko. ”Order” puolestaan kertoo, että valikon vaihtoehdot ovat ennalta määritellyt tiedon järjestämiseen liittyvät vaihtoehdot. ”Increasing order” valitsee nousevan järjestyksen oletusvalinnaksi.

Esimerkki 6

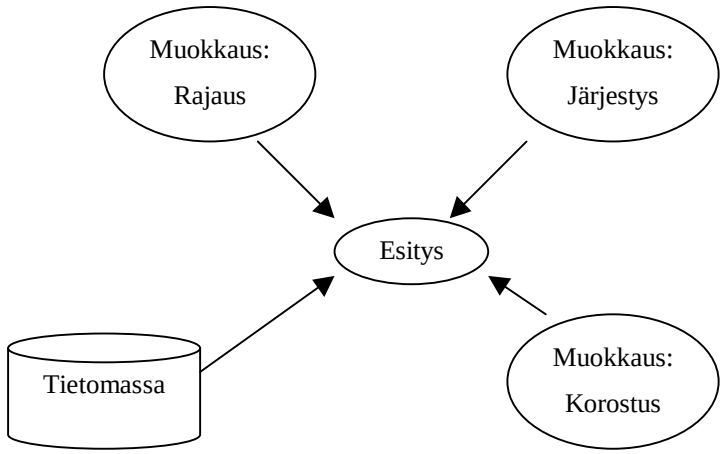
```
<applet code=Lahettaja.class width=100 height=100>
<param name=”Choice” value=”Order: Increasing order”>
<param name=”Inform” value=”Lähettäjän viestit”>
</applet>
```

7.1.5 Yhteenveto

Live Documents esittää toimivan ratkaisun komponenttipohjaiselle visualisoinnille HTML-sivuilla. Java-tekniikan käyttö antaa mahdollisuuden monipuolisten

visualisointikomponenttien tuottamiseen ja HTML-sivuilla määritellyt kommunikointisuhteet tekevät helpoksi komponenttien yhdistämisen toisiinsa. Live Document –järjestelmän komponentit muodostavat tähtimäisiä rakenteita, joiden keskuksina ovat esityskomponentit ja niiden ympärillä muokkauskomponentit.

Kuva 34



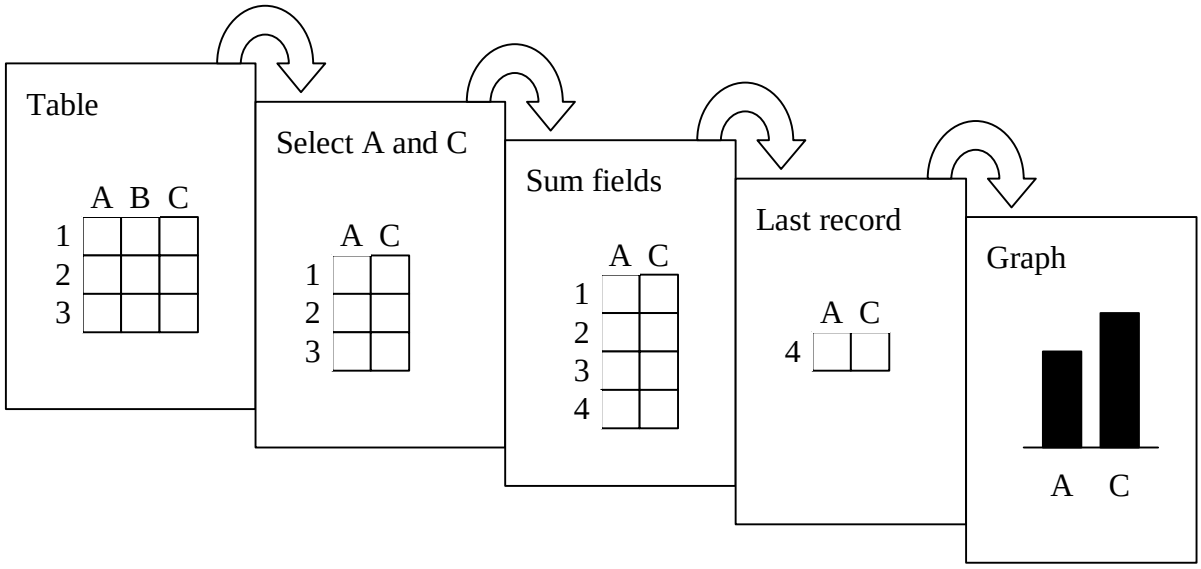
7.2 Sluice

Sluice on Virginian teknisessä yliopistossa kehitetty sovelluskehys, jolla voidaan luoda laboratoriomaisia visualisointiympäristöjä WWW-ympäristöön [Isenhour et al.]. Sluice ei itsessään ole visualisointiympäristö. Sluicella tuotettu visualisointiympäristö koostuu Javalla toteutetuista komponenteista, joilla voidaan käsitellä tietojoukkoja ja luoda niistä erilaisia visualisointeja. Kaikki järjestelmän visualisointikomponentit toteuttavat saman yksinkertaisen rajapinnan. Sluice on kehys, jossa on toteutettu komponenttipohjaisen visualisointiympäristön perustoiminnallisuus, ja jossa on määritelty rajapinnat omien visualisointikomponenttien toteuttamiseksi. Järjestelmä ei ole suunniteltu vain yhden käyttäjän ympäristöksi, vaan perustekniikassa on huomioitu visualisointilaboratorion hajautus usealle asiakasjärjestelmälle ja eri asiakkaiden yhteistoiminta.

7.2.1 Sluicen komponentit

Visualisointijärjestelmän komponentit järjestäytyvät nauhamaisesti (Kohta 6.7.2). Kukin nauhan visualisointikomponentti voi rajata, laajentaa tai muuten muokata visualisoitavaa tietoa tai näyttää sen käyttäjälle kuvana (Kuva 35). Komponentit kertovat tilassaan tapahtuvista muutoksista seuraaville komponenttiketjun komponenteille. Tapahtumanvälitys on toteutettu Observer-suunnittelumallin mukaisesti [Isenhour et al.; Gamma et al., 293-303].

Kuva 35 – Komponenttien tekemiä muunnoksia



Sluicen komponenttien tehtäväjako noudattaa visualisoinnin viitekehystä (4.1) [Card et al., 17]. Käsitteellisesti Sluice näkee komponentit kolmessa eri kategoriassa:

1. Lähdemoduulit, jotka luovat tai keräävät tietoa taulumuotoisiin tietorakenteisiin. Tieto voi olla peräisin tietokannasta tai tiedostosta. Tiedon lähde voidaan toteuttaa myös toiselle tietokoneelle hajautettuna sovelluksen osana käyttämällä Corba- tai RMI-tekniikoita. Visualisoinnin viitekehyksessä tämä vastaa muunnosta tietomassasta taulurakenteiksi (raw data -> data tables).

2. Käsittelijämoduulit, jotka muokkaavat taulumuotoisia rakenteita rajaamalla, laajentamalla tai muokkaamalla. Käsittelijämoduulin toimintaa voidaan ohjata käyttöliittymän tai parametritiedostojen avulla. Viitekehyksessä tätä vastaa muunnos taulurakenteista visuaalisiksi rakenteiksi (data tables -> visualization structures).

3. Visualisointimoduulit, jotka esittävät taulumuotoiset tietorakenteet käyttäjälle näkyvinä kuvina ja tarjoavat käyttäjälle vuorovaikutusvälineitä. Viitekehyksessä tämä vastaa muunnosta visuaalisista rakenteista näkymiksi (visual structures -> views).

Kaikki komponentit toteuttavat saman rajapinnan riippumatta tehtävästään tai sijainnistaan visualisointiketjussa. Ratkaisu yksinkertaistaa komponenttien tekoa, mutta korostaa myös sitä, että yksi visualisointi voi tarjota helppotajuisen mahdollisuuden rajata seuraavan visualisoinnin tietojoukkoa. Esimerkiksi pisteparvesta voidaan valita yksi tai useampi piste tarkempaan tarkasteluun. Tällaisessa tapauksessa on vaikea erottaa toisistaan käsittelijämoduulia ja visualisointimoduulia [Isenhour et al.].

Käyttäjällä on mahdollisuus koota moduuleista komponenttiketjuja, ja näin rakentaa omien toiveidensa mukainen visualisointi.

7.2.2 Ryhmätyön tukeminen

Sluice tukee sekä samanaikaista, että eriaikaista ryhmätyöskentelyä Internetin välityksellä. Samanaikaisessa työskentelyssä kaikilla työryhmän jäsenillä on käynnissä visualisointiympäristö, jossa on käytössä sama tietomassa. Siirtonopeutta säästääkseen Sluice säilyttää ryhmän jäsenten ympäristöissä yhteisen tilan välittämällä muutosviestejä. Kun yksi käyttäjä muuttaa visualisointinsa asetuksia, muutosviestin välityksellä muutos siirtyy kaikille muillekin ryhmän jäsenille. Muutokset voivat tapahtua sekä visualisointiketjun rakenteessa että yksittäisen visualisointikomponentin asetuksissa.

Sluicen perustekniikka mahdollistaa komponenttien tilojen tallentamisen palvelimelle tai tilojen välittämisen toiselle käyttäjälle. Tekniikka perustuu Javan yleiseen sarjallistamismekanismiin (serialization). Kukin komponentti osaa kertoa tilansa tekstuaalisesti ja visualisointijärjestelmä osaa siirtää tiedot tiloista palvelimelle. Tekstuaalinen tilatieto on selkokielinen, eli käyttäjän ymmärrettävissä jo sellaisenaan.

7.2.3 Yhteenveto

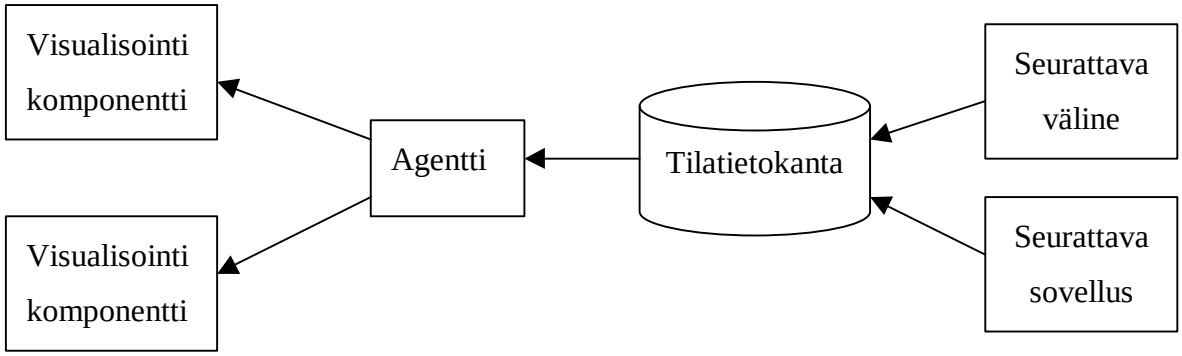
Sluice tarjoaa perustekniikan laboratoriomaisen visualisointiympäristön luontiin. Yksittäinen visualisointiympäristö on aina Java-kielellä toteutettu kokonaisuus. Toisin kuin Live Documents, Sluice ei siis ole visualisointiominaisuuksia lisäävä laajennus hypertekstisivustoon. Merkittävin Internetin tuoma hyöty Sluicen ominaisuuksissa on

visualisointijärjestelmän hajautuksessa. Hajautusta voidaan hyödyntää joko tiedon haussa tai usean käyttäjän ryhmätyöominaisuuksissa.

7.3 Dove

Dove (Distributed Object Visualization Environment) on Washingtonin yliopistossa kehitetty konsepti verkonhallinta- ja visualisointijärjestelmäksi WWW-käyttöliittymällä. Verkon tilan seuranta on toteutettu agenttiteknologialla. [Schmidt]

Kuva 36



7.3.1 Tiedon keräys tilatietokantaan

Dove-järjestelmä kerää seurattavien laitteiden tai sovellusten informaatiota tilatietokantaan (management information base). Tilatietokannan päivityksestä huolehtii kutakin seurattavaa kohdetta varten toteutettu sovellus. Yhdessä Dove-järjestelmässä on kerrallaan vain yksi tilatietokanta, johon tallennetaan kaikkien seurattavien kohteiden tilat ja niihin tapahtuvat muutokset. [Schmidt]

7.3.2 Tietojen visualisointi

Tilatietokannan tietojen välitys visualisointikomponenteille tapahtuu agenttien välityksellä. Agentit on toteutettu Ainokainen-suunnittelumallin (Singleton) mukaisesti siten, että kustakin agenttiluokasta on olemassa kerrallaan korkeintaan yksi agenttiolio.

Agentti seuraa tilatietokannan muutoksia joko tosiaikaisesti tai ajastetusti. Ajastetussa mallissa agentti tarkkailee tilatietokantaa tietyin väliajoin ja huomaa siellä tapahtuneet muutokset. Tosiaikaisessa mallissa tilatietokanta kertoo agentille muutoksistaan itse.

Visualisointikomponentit ilmoittautuvat agenttiolioille ja alkavat tämän jälkeen saada agentilta tietoja seurattavista kohteista. Visualisointikomponentti voi ilmoittautuessaan kertoa, minkä laitteiden tapahtumia haluaa seurata, jolloin agentti osaa lähettää ainoastaan komponenttia kiinnostavia tietoja. Jos agentti ei voi ottaa esimerkiksi palomuurin takia suoraa yhteyttä visualisointikomponenttiin, komponentti kysyy agentilta lyhyin väliajoin mahdolliset muutokset tilatietokannassa. Mikäli visualisointikomponentti on toteutettu Java-applettina, joudutaan muutokset kysymään ajastetusti.

7.3.3 Yhteenveto

Dove-tekniikkaa voidaan käyttää joko Java-appletteina tai erillisinä sovelluksina. Taustalla on kuitenkin aina palvelin, jossa sijaitsee tilatietokanta ja mahdolliset agentit. Visualisointiprosessiin osallistuvat komponentit järjestyvät nauhamaisesti siten, että tiedon lähdettä edustaa palvelinympäristössä toimiva agentti ja esitysosuutta visualisointikomponentti. Doven mukaisella arkkitehtuurilla tiedon keräys, käsittely ja esitys on eritelty tiukasti erillisiksi kokonaisuuksiksi, kun Sluicen mallissa komponenttityyppien rajat olivat hyvin hämärät.

8 Yleiskäyttöinen visualisointikomponentin runko

Visualisointijärjestelmän rakenteesta voidaan esittää useita erilaisia malleja, jotka soveltuvat määrättyihin sovelluskohteisiin. Tässä tutkimuksessa on esitelty tieteen piirissä tehtyjä visualisointijärjestelmiä, joissa järjestelmä oli suunniteltu reaalimaailman tutkimus-ympäristöjä mallintamalla. Sluice [Isenhour et al.] käytti lähtökohtana laboratoriometaforaa, kun taas LiveDocs [Mockus et al.] yhdisti visualisoinnin tarpeen kirjan lukemiseen. Erilaiset tarpeet ja lähestymistavat tekevät yleispätevän visualisointijärjestelmän teon vaikeaksi ellei jopa mahdottomaksi. Kunkin visualisointijärjestelmän ratkaisu tukeutuu johonkin ajatusmalliin.

Jos visualisointijärjestelmän rakenne on eri järjestelmissä merkittävästi toisistaan poikkeava, ovat visualisointikomponentit sen sijaan periaatteiltaan hyvin samankaltaisia. Tästä lähtökohdasta ajatellen onkin mielekästä kehittää visualisointikomponentteja, jotka eivät sitoudu yksittäiseen visualisointijärjestelmään vaan voivat toimia monenlaisissa ympäristöissä.

Tässä luvussa tullaan määrittelemään vaatimuksia, joita yleiskäyttöiselle visualisointikomponentille asetetaan. Vaatimuksiin perustuen kuvataan ratkaisu, jolla komponentti on toteutettavissa. Tässä ratkaisumallissa käytetään toteutuskielenä Javaa. Monet esitetyistä ratkaisuista ovat kuitenkin toteutettavissa myös muilla ohjelmointikielillä. Ratkaisun tarkkuudessa pysytellään rakenteellisella tasolla. Päämääränä on esitys komponentin osista ja rajapinnoista.

8.1 Vaatimukset

Tämä tutkimus on esitellyt erilaisia komponenttipohjaisen visualisoinnin arkkitehtuuri-ratkaisuja, toteutustekniikoita ja visualisointiprosessin vaiheita. Keräämällä erilaiset vaihtoehdot yhteen saadaan kattava lista vaatimuksia, joita yleiskäyttöiselle visualisointikomponentille voidaan asettaa.

1. **Vuorovaikutteisuus.** Järjestelmän tulee mahdollistaa erilaiset käyttäjän vuorovaikutustavat. Komponenteilla voidaan toteuttaa sekä staattisia (Kohta 2.2.1) että

dynaamisia (Kohta 2.2.3) visualisointijärjestelmiä. Jokaisella komponentilla, myös palvelimilla sijaitsevilla, on mahdollisuus vuorovaikutukseen käyttäjän kanssa.

2. **Hajautustekniikat.** Järjestelmän tulee olla hajautettavissa usealle eri tietokoneelle. Hajautustekniikka voidaan valita tarpeen mukaan. (Kohta 3.3)

3. **Arkkitehtuuritasot.** Komponenttien täytyy pystyä toimimaan kaikilla monitaso-arkkitehtuurin tasoilla. (Kohdat 3.4 ja 3.5)

4. **Komponenttien erityispiirteet**. Saman mallin perusteella on voitava rakentaa kaikentyyppisiä visualisointikomponentteja (Kohta 4.1). Kokonaisuus voi muodostaa visualisoinnin viitekehyksen (Kohta 4.2) mukaisen järjestelmän.

5. **Riippumattomuus ympäristöstä**. Komponenttimallin tulee olla riippumaton erilaisista visualisointiympäristöistä. Komponentit voivat sijaita kiinteän visualisointijärjestelmän sisällä tai vaikka WWW-sivulla. Mallissa tulee mahdollistaa erilaiset tavat komponenttien järjestäytymiseen (Kohta 6.2), komponenttien tutustumiseen (Kohta 6.3), komponenttien keskusteluun (Kohta 6.4) ja komponenttien alustamiseen (Kohta 6.6).

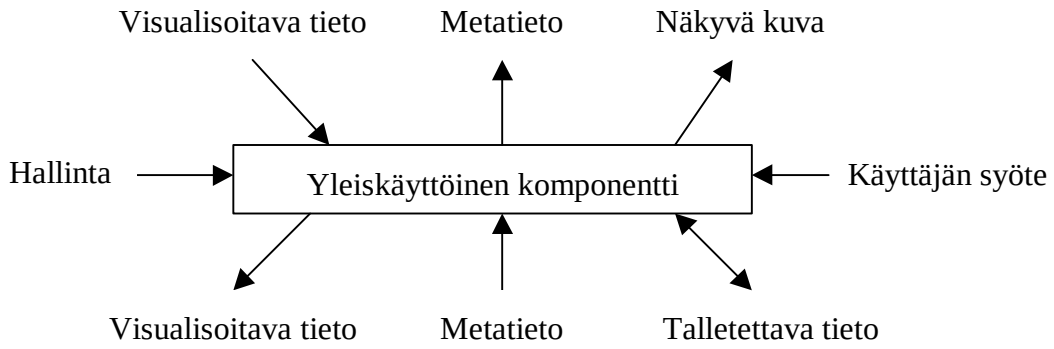
6. **Tiedonsiirtomuoto.** Komponenttimalli ei saa ottaa kantaa komponenttien keskinäiseen tiedonsiirtomuotoon (Kohta 6.5).

7. **Komponenttien keskustelusuunta.** Mallilla tulee voida toteuttaa sekä tähtimäisiä, että nauhamaisia visualisointijärjestelmiä (Kohta 6.7).

8.2 Rakennemalli

Yleiskäyttöisen visualisointikomponentin runkoa on luontevaa lähteä suunnittelemaan erilaisten visualisointikomponenttien tarpeita ja visualisoinnin viitekehystä tarkastelemalla. Kohdassa 4.1 esiteltiin HyperVis-oppimateriaalissa [Owen] listatut visualisointiprosessiin kuuluvat osat tietovirtoineen. Suunniteltaessa yleiskäyttöistä mallia visualisointikomponentille joudutaan ottamaan huomioon kaikkien prosessin vaiheiden vaatimukset. Yhdistämällä kaikkien osien tietovirrat saadaan kuva yleisen visualisointikomponentin mahdollisista tietovirroista (Kuva 37).

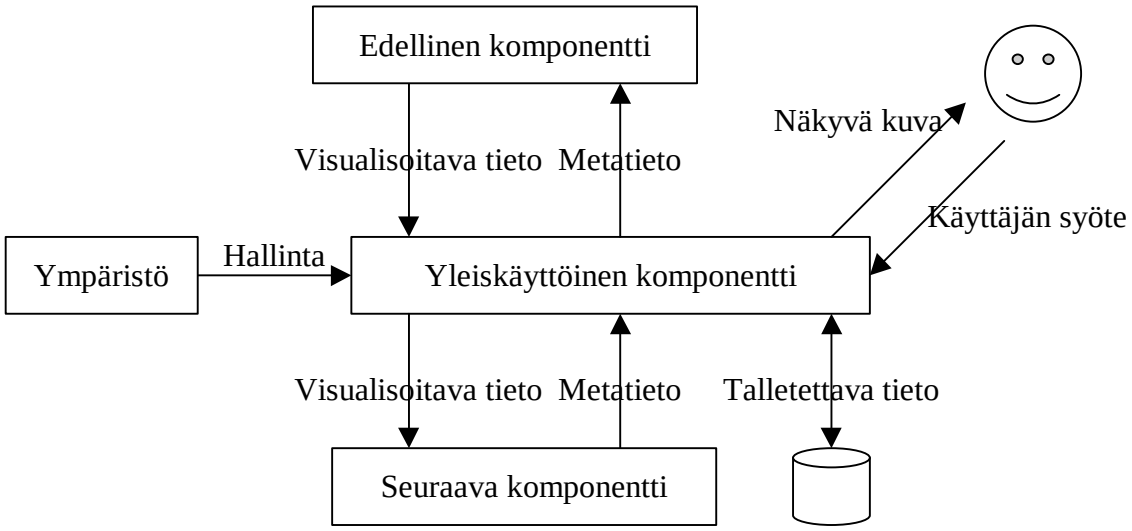
Kuva 37



Kuvasta huomataan, että komponenttiin kohdistuu hyvin monenlaisia tietovirtoja. Osa tietovirroista ohjaa komponentin toimintaa ja osa vie visualisoitavaa tietoa eri suuntiin. Vertaamalla kuvaa visualisoinnin viitekehyksen määrittelemiin tietovirtoihin (Kuva 18) huomataan, että esitettyjä tietovirtoja käyttämällä on mahdollista toteuttaa myös viitekehyksen mukaisia visualisointikomponentteja.

Komponentin tietovirtakaaviota voidaan laajentaa lisäämällä siihen eri virtojen keskustelukumppanit. Tällä tavoin muodostuneesta kuvasta voidaan löytää komponenttiin liittyvät sidosryhmät ja niiden vuorovaikutussuhteet komponentin kanssa (Kuva 38).

Kuva 38



Komponentilla on nähtävissä viisi mahdollista keskustelukumppania. Käyttäjä voi antaa komponentille syötteitä ja komponentti voi näyttää käyttäjälle käyttöliittymän tai kuvan visualisoitavasta tiedosta. Komponentti voi hakea tietoa tietokannasta sekä tallettaa sinne esimerkiksi tietoja visualisointijärjestelmän tilasta. Ympäristö käynnistää komponentin, antaa sille lähtötiedot ja ohjaa sen toimintaa. Komponentti muodostaa edellisen ja seuraavan komponentin kanssa ketjun, jossa visualisoitava tieto kulkee ketjussa eteenpäin ja ketjun toimintaa ohjaava metatieto taaksepäin.

8.3 Komponentin luokkakaavio

Komponentin tietovirtojen ja keskustelukumppanien kartoitus on luonteva lähtökohta komponentin luokkakaavion suunnitteluun. Koska jokainen keskustelukumpani voi olla teoriassa mikä tahansa komponentin kanssa yhteensopiva olio, on järkevää erottaa jokainen keskusteluyhteys omaksi rajapinnakseen. Seuraavassa kuvataan komponentin ja sen sidosryhmien väliset yhteydet. (Kuva 39)

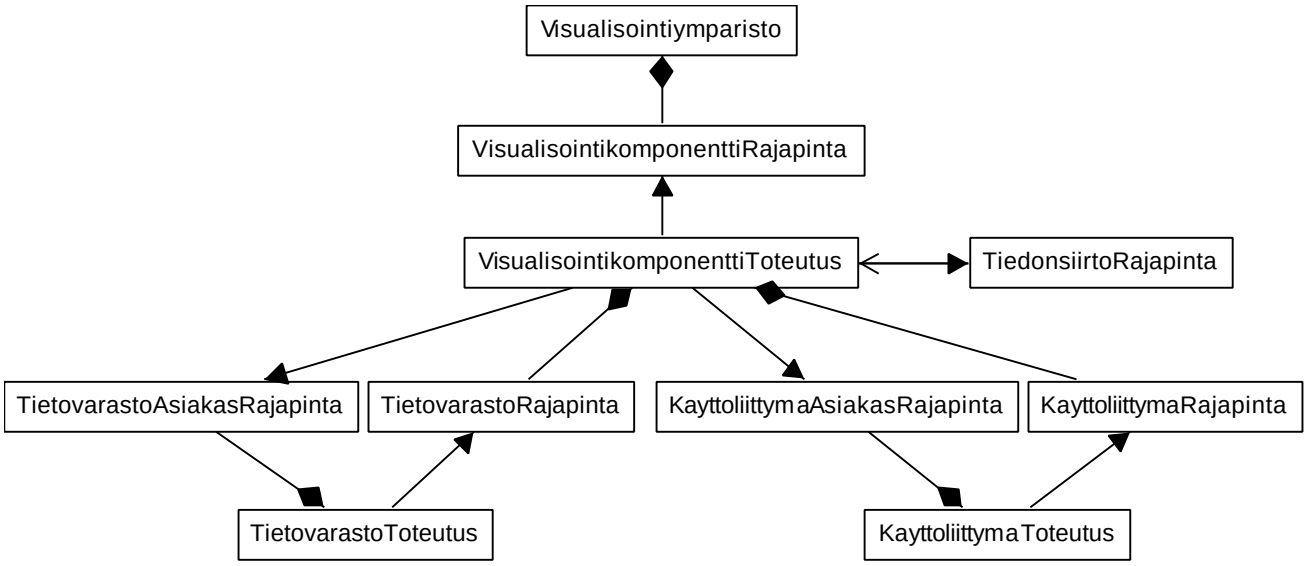
Kuva 39

Lähde	**Kohde**	**Kuvaus**
Käyttäjä	Komponentti	Käyttäjä voi vaikuttaa komponentin toimintaan.
Komponentti	Komponentti	Komponentti voi saada visualisoitavaa tietoa edelliseltä komponentilta ja lähettää sitä eteenpäin seuraavalle komponentille.
Komponentti	Tietovarasto	Komponentti voi tallentaa tietoa tietovarastoon.
Komponentti	Käyttäjä	Komponentti voi olla yhteydessä käyttäjään.
Tietovarasto	Komponentti	Tietovarasto voi toimittaa komponentille tietoa.
Ympäristö	Komponentti	Ympäristö luo komponentin ja hallitsee sen toimintaa.

Taulukkoesityksestä voidaan nähdä mitä yhteyksiä yksittäiseen komponenttiin liittyy. Kutakin yhteyttä varten luodaan oma rajapinta. Komponentti, jolla on olemassa kaikki

mahdolliset yhteydet, toteuttaa rajapinnan käyttäjää, komponenttia, tietovarastoa ja ympäristöä varten. Käyttämällä rajapintoja annetaan komponentin toteuttajalle täysi valta päättää toteutustavasta. Tästä seuraa, että esimerkiksi komponenttiketjun hajautukselle ei anneta mitään rajoituksia. Lisäämällä rajapinnat kuvaan komponenteista sidosryhmineen (Kuva 38) saadaan komponentin ja sen ympäristön luokkakaavio. Seuraavassa esityksessä on käytetty UML-notaatiota (Kuva 40).

Kuva 40



Visualisointiympäristö sisältää joukon visualisointikomponentteja (VisualisointikomponenttiToteutus). Kukin komponentti on toteutus yleiselle visualisointikomponentin rajapinnalle (VisualisointikomponenttiRajapinta).

Komponenttien keskinäinen tiedonsiirto tapahtuu TiedonsiirtoRajapinnan välityksellä. Jos visualisointikomponentilla on yhteys tietovarastoon, tietovaraston (TietovarastoToteutus) täytyy toteuttaa TietovarastoRajapinta. Jotta tietovarasto voisi lähettää tietoa visualisointikomponentille, komponentin on toteutettava TietovarastoAsiakasRajapinta.

Halutessaan vastaanottaa tietoa käyttöliittymältä (KäyttöliittymäToteutus), visualisointikomponentin täytyy toteuttaa KäyttöliittymäAsiakasRajapinta. Jos visualisointi-

komponentti haluaa lähettää tietoa käyttäjälle, käyttöliittymän täytyy toteuttaa rajapinta KäyttöliittymäRajapinta.

Visualisointikomponentti (VisualisointikomponenttiToteutus) on yhteydessä ympärillä oleviin järjestelmän osiin ainoastaan rajapintojen kautta. Itse komponentin tehtäväksi jää ottaa yhteydet rajapintojen toteutuksiin ja toteuttaa itse komponentin yksilöllinen toiminnallisuus. Esitellyssä mallissa on siis komponentin toiminnallisuus täysin erillään visualisointiympäristöön sidotuista toiminnallisuuksista.

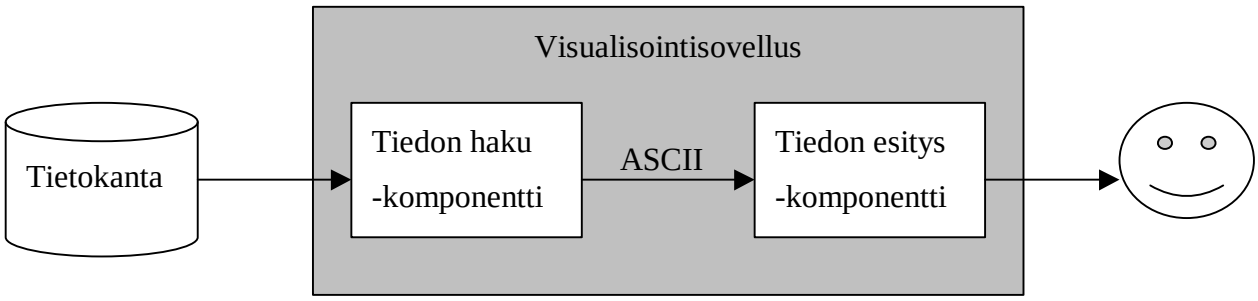
8.4 Yksinkertainen visualisointijärjestelmä

Tässä kappaleessa esitellään yksinkertainen komponenttipohjainen visualisointijärjestelmä ja toteutetaan se käyttämällä edellisessä kappaleessa kuvattua yleisen visualisointikomponentin mallia. Esimerkkijärjestelmä esitetään ensin toiminnallisesti ja sen jälkeen karkean tason luokkakaaviona.

8.4.1 Toiminnallinen malli

Järjestelmä on nimeltään *Visualisointisovellus*. Sovellus koostuu kahdesta komponentista. *Tiedon haku* –komponentti hakee visualisoitavan tiedon tietokannasta ja lähettää sen ASCII-muodossa *Tiedon esitys* –komponentille. Tämä puolestaan esittää visualisoitavan tiedon kuvana käyttäjälle. Yksinkertaisuuden vuoksi käyttäjällä ei ole mahdollisuutta muokata visualisointia. (Kuva 41)

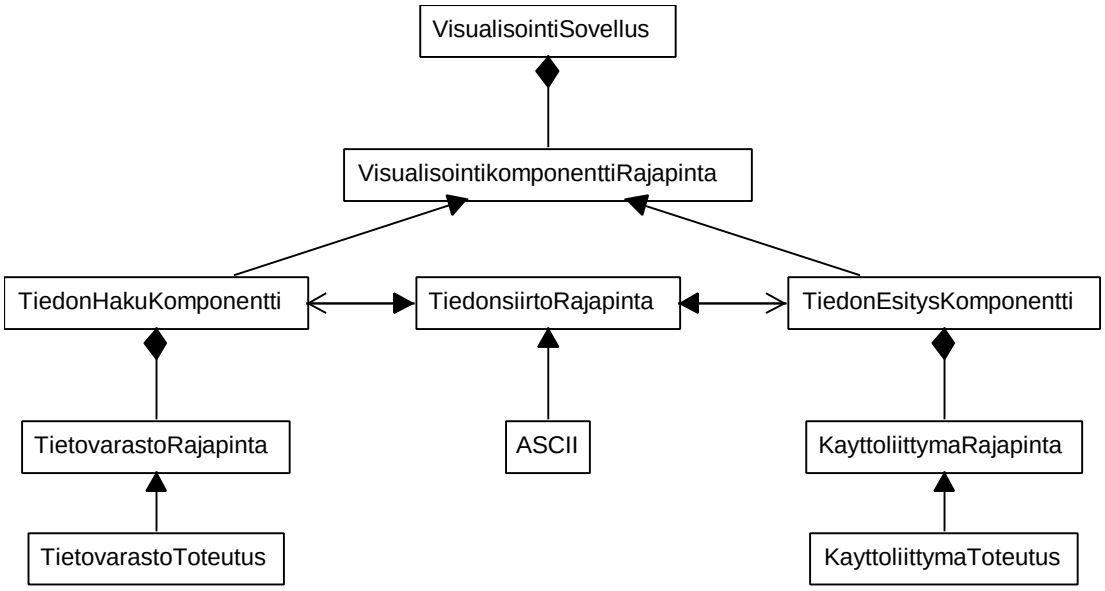
Kuva 41



8.4.2 Luokkakaavio

Luokkakaaviossa toiminnallisen kuvan visualisointisovellusta vastaa luokka VisualisointiSovellus, johon kuuluu kaksi VisualisointikomponenttiRajapinnan toteuttavaa komponenttia. TiedonHakuKomponentilla on yhteys tietokantaan TietovarastoRajapinnan kautta. Yhteys TiedonEsitysKomponenttiin tapahtuu TiedonsiirtoRajapinnan kautta. Koska tiedon siirto on tarkoitus tapahtua ASCII-muodossa, TiedonsiirtoRajapinnasta käytetään ASCII-pohjaista toteutusta. TiedonEsitysKomponentti pitää sisällään yhteyden käyttöliittymään. Keskustelu komponentista käyttöliittymään tapahtuu KäyttöliittymäRajapinnan kautta. (Kuva 42)

Kuva 42



8.5 Yhteenveto ja tulosten arviointi

Edellisissä kohdissa luotiin HyperVis-oppimateriaalissa [Owen] esittettyjen visualisointi-komponentin toiminnallisten rakenteiden perusteella yleiskäyttöisen komponentin rakenne. Rakennemallissa eriteltiin tietovirrat ja näin muodostuneeseen kuvaan lisättiin tietovirtojen kautta komponentin kanssa yhteydessä olevat keskustelukumppanit. Tämän jälkeen listattiin eri tietovirrat päätepisteineen ja havaittiin komponentin ympäristöön tarvittavat rajapinnat.

Lopuksi rajapinnat sijoitettiin komponenttien ja sen keskustelukumppanien välille ja luotiin kokonaisuudesta luokkakaavio.

Kun verrataan toteutunutta luokkakaaviota kohdassa 8.1 esitettyihin yleiskäyttöisen visualisointikomponentin vaatimuksiin huomataan, että muodostunut komponentin rakenne täyttää asetetut vaatimukset. Kaikki komponentin ympäristötekijät on kartoitettu ja erotettu rajapinnoilla. Toteuttamalla rajapintoja on mahdollista kattaa kaikki vaatimukset.

Yleiskäyttöisen visualisointikomponentin rakennemalli mahdollistaa komponentin toiminnallisen osuuden erottamisen osista, jotka kiinnittävät komponentin ympäristöönsä. Rajapintojen avulla voidaan toiminnallisuuden ympärille kiinnittää tarpeen mukaan liittymiä toisiin komponentteihin, tietovarastoihin tai käyttäjärajapintaan. Rakennemallissa mikä tahansa komponenttiin kuuluva osa voidaan vaihtaa toiseen ilman, että komponentin toiminnalliseen osuuteen täytyy koskea. Komponentin osista voidaan tehdä eri tilanteisiin sopivia valmiita toteutuksia. Esimerkiksi tiedonsiirtoon käytetystä rajapinnasta voidaan tehdä visualisointikomponentteja varten toteutukset niin ASCII- kuin XML-siirtoakin varten.

Koska komponentin toiminnallisuus on erotettu ympäristöstä riippumattomaksi, voidaan samaa komponentin ydintä käyttää kaikissa edellä kuvattua komponenttimallia tukevissa visualisointijärjestelmissä riippumatta niiden järjestäytymis- tai hajautustavoista. Ydintä ympäröivät rajapinnat kytkevät toiminnallisuuden komponenttia käyttävään visualisointiympäristöön.

9 Yhteenveto

Tässä tutkimuksessa on tutustuttu Internetin, nykyaikaisen ohjelmistosuunnittelun ja tiedon visualisoinnin yhdistämiseen. Jokaisesta näistä aiheista on tehty suuret määrät tieteellistä tutkimusta ja uusia tutkimuksia tehdään jatkuvasti. Sen sijaan tutkimus näiden asioiden yhdistämiseksi on jäänyt vähemmälle.

Perustellun lähtökohdan näiden asioiden yhdistämiseksi tarjoavat erilaiset kuvaukset visualisointiprosessista. Tämä tutkimus liitti uutena asiana visualisointiprosessin komponenttipohjaiseen sovellusarkkitehtuuriin sekä usealle tietokoneelle hajautettuihin palvelinratkaisuihin. Eri arkkitehtuuriratkaisujen vaikutukset visualisointiprosessiin esitettiin visualisoinnin viitekehyksen vaihejakomallia soveltaen. Lisäksi tutkittiin visualisointikomponenttien käyttäjärajapinnan toteutusmahdollisuuksia WWW-sivuilla. Tätä aihetta valaistiin käyttäjärajapinnan komponenteista tehtyjen tutkimusten avulla. Myös käyttäjärajapinnan komponenttiratkaisuja sovitettiin visualisoinnin viitekehykseen.

Esiteltyjä malleja ja visualisointijärjestelmän toteutustekniikoita verrattiin kolmeen eri tyyppiseen visualisointiympäristöön. Hyvin erilaiset esimerkkiympäristöt toivat esiin vaihtoehtoisten ratkaisumallien käyttökohteita ja osoittivat visualisointijärjestelmien monimuotoisuuden. Visualisointijärjestelmät pyrittiin rakentamaan luonnollisten vertauskuvien pohjalta esimerkiksi virtuaaliseksi laboratorioksi tai vuorovaikutteiseksi kirjaksi.

Visualisointijärjestelmien monimuotoisuutta voidaan pitää rikkautena. Kukin erilainen lähestymistapa tarjoaa omalle kohderyhmälleen sopivimman tavan päästä käsiksi visualisointityökaluihin. Internetissä tapahtuvan tiedon visualisoinnin lisääntyminen muuttaa järjestelmien käyttäjäryhmiä ja tutkimuskohteita yhä moninaisemmiksi. Tämän vuoksi on järkevää myös kehittää erilaisia visualisointijärjestelmiä eri tarpeisiin sen sijaan, että tutkittaisiin järjestelmää, joka sopisi kaikkiin sovelluskohteisiin.

Tutkimuksen päätteeksi todettiin, että mielekkäämpää kuin visualisointijärjestelmien yleistäminen, on kehittää visualisointikomponentteja, jotka voivat toimia eri järjestelmissä. Yleiskäyttöisten visualisointikomponenttien rakenteelle kerättiin vaatimuslistaa ja

visualisointiprosessin vaiheita tutkimalla johdettiin eräs ratkaisu yleiskäyttöisen visualisointikomponentin luokkarakenteeksi.

Yleiskäyttöiset visualisointikomponentit antavat mielestäni kiehtovan lähtökohdan jatkotutkimuksille. Tuloksena syntyvät komponentit eriyttäisivät komponentin loogisen toiminnan visualisointijärjestelmän rakenteellisista toiminnoista.

Kiinnostavia tutkimusaiheita tarjoavat myös hajautetut visualisointijärjestelmät. Hajautuksen avulla visualisoinnissa voidaan hyödyntää Internetin laajat tietolähteet ja nopeat kansainväliset yhteydet. Hajautetuilla ratkaisuilla voidaan toteuttaa usean käyttäjän virtuaalisia tiedeyhteisöjä, joissa eri maissa työskentelevät tutkijat voivat paneutua yhteisiin tutkimuskohteisiin.

Viiteluettelo

[Card et al.] Card, Mackinlay and Shneiderman, *Readings in Information Visualization – Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[Chuah et al.] Mei C. Chuah, Steven F. Roth, Joe Mattis and John Kolojejchick, SDM: Selective Dynamic Manipulation of Visualizations. In: *Proceedings of UIST'95, ACM Symposium on User Interface Software and Technology* (1995), Pittsburgh, pp. 61-70

[Dincer et al.] Kivanc Dincer and Geoffrey C. Fox, Using Java and JavaScript in the Virtual Programming Laboratory: A Web-Based Parallel Programming Environment. Available as *http://www.npac.syr.edu/users/dincer/papers/SCCS-788-conc/Conc-vpl.html*

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

[Gosling et al.] James Gosling, Bill Joy, Guy Steele and Gilad Bracha, The Java Language Specification (Second Edition). 2000. Available as *http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html.*

[Hamilton] Graham Hamilton (ed), JavaBeans Specification 1.01. 1997. Available as *http://java.sun.com/products/javabeans/docs/beans.101.pdf.*

[HTTP] HTTP 1.0 Specification (RFC 1945). Available as *http://www.cis.ohio-state.edu/htbin/rfc/rfc1945.html.*

HTTP 1.1 Specification (RFC 2068). Available as *http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html.*

[Isenhour et al.] Philip Isenhour, Clifford A. Shaffer, James "Bo" Begole, Jeff Nielsen and Marc Abrams, Sluice: A Java-Based Framework for Collaborative

Interactive Modular Visualization Environments. Available as *http://simon.cs.vt.edu/linc/public/Sluice/sluice.html.*

[Jern1] Mikael Jern, ”Thin” vs. ”Fat” Visualization Clients. In: *AVI’98, Advanced Visual Interfaces,* May 22-27, 1998, pp. 270-273

[Jern2] Mikael Jern, Active X Visualization Components. 1998. Available as *http://www.uni-ulm.de/www.avs.com/solution/success/papers/activex.htm.*

[Jern3] Mikael Jern, Information Visualization on the Web. In: CODATA Euro-American Workshop, Visualization of Information and Data: Where We Are and Where Do We Go From Here, June 24-25, 1997

[Johns] Paul Johns, The ABC of MFC ActiveX Controls. 1996. Available as *http://msdn.microsoft.com/library/techart/msdn_abcsmfc.htm*

[Koskimies] Kai Koskimies, *Pieni oliokirja*. Suomen ATK-kustannus Oy, 1998

[Mockus et al.] Audris Mockus, Stacie Hibino and Todd Graves, Flexible Information Visualization Components for Authoring WWW Live Documents. Available as *http://www.niss.org/~graves/papers/webvis/webvis.html*.

[Owen] G. Scott Owen (ed.), HyperVis – Teaching Scientific Visualization Using Hypermedia. Available as *http://www.dcc.ufba.br/mat056/hypervis/hypervis.htm.*

[RMI] Java Remote Method Invocation Specification. Available as *http://java.sun.com/products/jdk/rmi/index.html.*

[RMI2] Java Remote Method Invocation – Distributed Computing for Java (White Paper). 1999. Available as *http://java.sun.com/marketing/collateral/javarmi.html.*

[Schmidt] Douglas C. Schmidt, The Distributed Object Visualization Environment. Available as *http://www.cs.wustl.edu/~schmidt/dove.html.*

[Shneiderman] Ben Shneiderman, Dynamic queries for visual information seeking, *IEEE Software*, **11**, 6, (1994), 70-77

[Sun] *Java Technology Architecture Planning and Design*. Sun Microsystems Inc, 1998

[Webster] Merriam-Webster Online. http://www.merriamwebster.com

[Whatis] WhatIs.com. http://www.whatis.com

[Williams et al.] Sara Williams and Charlie Kindel, The Component Object Model: A Technical Overview. 1994. Available as *http://msdn.microsoft.com/library/techart/msdn_comppr.htm.*

[Williamson et al.] C. Williamson and B. Shneiderman, The Dynamic Home Finder: Evaluating Dynamic Queries in a Real-Estate Information Exploration System. In: *Proceedings of SIGIR'83, ACM Conference on Research and Development in Information Retrieval*, 1983, New York, 339-346