Jani Makkonen

Grafiikan ohjaus Vizrt-järjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

21.3.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jani Makkonen Grafiikan ohjaus Vizrt-järjestelmässä 43 sivua 21.3.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Art Director Annette Stenroos

Ensin työssä esitellään Vizrt-järjestelmän eri osat eli Content Pilot, Viz Artist, Viz Template Wizards ja Viz Engine. Content Pilot on sapluunoiden käyttämiseen tarkoitettu sovellus. Viz Artist taas on grafiikan tekemiseen tarkoitettu sovellus. Template Wizardilla tehdään sapluunoita ja Viz Enginessä toistetaan valmiit scenet eli näytetään grafiikka.

Sitten käydään läpi mitä grafiikan ohjaaminen tarkoittaa, kerrotaan mitä scenet ja sapluunat ovat ja kerrotaan mitä ovat sapluunoiden variaatiot ja miten ohjausliitännäiset toimivat.

Sitten kerrotaan millaista ohjelmointia Vizrt järjestelmään voi tehdä. Ohjelmointia voi tehdä sapluunoihin, enginelle voi antaa komentokäskyjä ja Viz Artistiin voi ohjelmoida sen omalla skriptikielellä Viz Scriptillä.

Tämän jälkeen mennään Viz Script -kielen yksityiskohtiin. Käydään läpi kielen rakenne ja proseduurit mitä se sisältää. Lisäksi tutustutaan kosketusnäytön käyttöön Vizrt-maailmassa.

Lopuksi tutustutaan työn fyysiseen osaan eli modulaariseen malliin usein koodiesimerkein. Modulaarisen mallin idea on näyttää erilaista dataa graafisessa muodossa. Vaihtoehtoihin kuuluvat piirakka-, käyrä- ja pylväsgrafiikka sekä koko ruudun kuvat. Käyttäjä voi syöttää arvot ja jaksottaa grafiikoita suoraan sapluunasta käsin. Tämän mahdollistaa kahta kerrosta käyttävä malli, jossa keskimmäisellä graafisella kerroksella oleva master scene ohjaa etummaisella kerroksella olevia lapsi-scenejä.

Avainsanat	Vizrt, scene, sapluuna, Viz Script, modulaarinen malli

Author(s) Title Number of Pages Date	Jani Makkonen Controlling Graphics in Vizrt System 43 pages 21 March 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Annette Stenroos, Art Director

First the thesis introduces the different parts of the Viz system, Viz Artist, Content Pilot, Template Wizard and Viz Engine. Content Pilot is software used to manipulate and fill templates, Viz Artist is used to make the graphics whereas Template Wizards is for making the templates and Viz Engine is used to show the graphics.

Then, the thesis describes how to control graphics is the Viz system, including what are scenes and templates. Then it describes what are the variants of the templates and how the control plugins work.

Then the study explains what kind of programming it is possible to do in the Vizrt system. Programming is done in templates, so one can give commands straight to the Viz Engine and use Viz Script in the Viz Artist.

Then the specifics of the Viz Script-programming language are introduced, as well as the composition of the language and procedures which it includes.

Lastly the modular model with some code examples is introduced. The idea of the modular model is to show different data in graphical form. Options include pie, graph and column graphics as well as whole screen pictures. The user can input the data values and stack the graphics straight from the template. This is possible due to the model using two layers, in which the master scene on the middle graphical layer controls child scenes on the front layer.

Keywords	Vizrt, scene, template, Viz Script, modular model

Sisällys

1 Johdanto

Tämän työn tarkoituksena on käsitellä uutisgrafiikan ohjausta Vizrt-ohjelmistoperheellä. Ensin perehdytetään lukija Vizrt-järjestelmään ja sen jälkeen paneudutaan työn pääaiheeseen, eli grafiikan ohjaukseen. Vizrt sisältää valmiita komponentteja, joilla grafiikan ohjausta voi suorissa lähetyksissä tehdä, mutta lisäämällä omaa skriptikoodia on mahdollista rikastaa ja parantaa grafiikan ohjauksen mahdollisuuksia järjestelmässä.

Ohjelmointi mahdollistaa grafiikan dynaamisuuden suorissa lähetyksissä, sekä antaa henkilökunnalle (graafikot, toimittajat ja tekninen väki) mahdollisuuden vaikuttaa grafiikan sisältöön nopeasti ja mahdollisimman vaivattomasti.

Ensin käydään läpi eri ohjelmointikielet ja komennot, joita ohjaamiseen voi käyttää ja sen jälkeen käydään läpi muutamia sapluuna-scene-yhdistelmiä, joissa edellä mainittuja ratkaisuja on hyödynnetty.

Työn ymmärtäminen vaatii lukijalta perustietoja olio-ohjelmoinnista ja it-tekniikasta. Työn on myös tarkoitus toimia apuvälineenä työn teettäjän Yleisradio Oy:n työntekijöille.

2 Vizrt-ohjelmistoperheen esittely

2.1 Vizrt yleisesti

Vizrt on norjalainen yritys, jonka nimi muodostuu sanoista VizuliZation (in) Real Time. Se on kehittänyt norjalaisen tv-kanava TV2:n kanssa yhteistyössä samannimisen ohjelmistoperheen, jonka tarkoituksena on mahdollistaa kolmiulotteisten grafiikoiden reaaliaikainen ulosajo sekä helpottaa työnkulkua onnistuneen uutislähetyksen aikaansaamiseksi kaikkien työalueiden osalta. Vizrt tarjoaa työkaluja median säilyttämiseen, grafiikan tekemiseen, uutistapahtumien tallentamiseen jne.

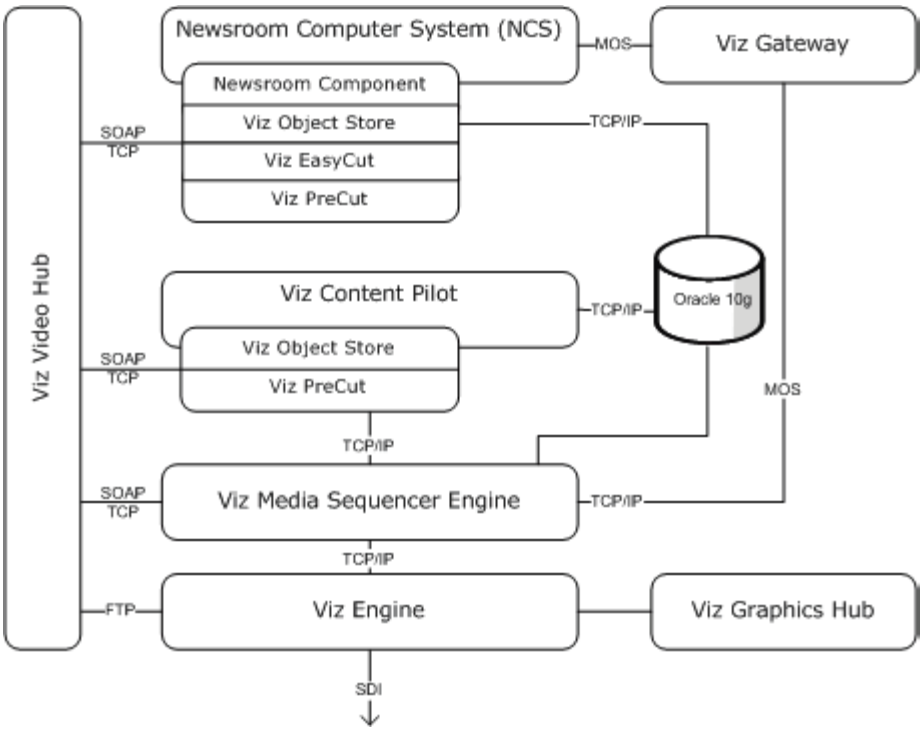
Vizrt:in ohjelmistot ovat käytössä useissa tv-yhtiöissä ympäri maailman (mm. CNN, CBS, Fox) ja sillä on käytännössä monopoli alalla. Suomessa sitä käytetään lähes kaikilla kanavilla (mm. YLE, Mtv3 ja Nelonen).

2.2 Vizrt:in osat

Vizrtin keskeisin osa on Viz Engine, joka on tarkoitettu grafiikan 2d- ja 3d-grafiikan reaaliaikaiseen renderöimiseen. Se käyttää OpenGL-kirjastoa, joka mahdollistaa grafiikan ajamisen useilla erilaisilla raudan ja käyttöjärjestelmien yhdistelmillä.

Grafiikan kehitystä varten Viz Engineen on liitetty Viz Artist -niminen visuaalinen käyttöliittymä ja grafiikan rakennustyökalu. Artist mahdollistaa grafiikan tekemisen erilaisista graafisista elementeistä animaatioineen ja ääniefekteineen. Tämä työ keskittyy eniten Viz Artistin ja Viz-sapluunoiden ohjelmointimahdollisuuksiin ja kykyyn kehittää dynaamista grafiikkaa.

Viz-sapluunoja tehdään Viz Template Wizard -nimisellä ohjelmalla. Ne ovat suoraan kytköksissä Viz Artistilla tehtyyn grafiikkaan ja mahdollistavat esimerkiksi grafiikan tekstuaalisen sisällön muutoksen juuri ennen suoraa lähetystä tai jopa sen aikana. Template Wizard on taas vastaavasti yhdistetty Viz Content Pilot -nimiseen ohjelmaan, jolla käytetään valmiita sapluunoita ja annetaan Viz Enginelle käskyjä grafiikan näyttämiseksi.



Kuva 1. Vizrt:n järjestelmäarkkitehtuuri [1, s. 5]

Tehtyjen grafiikkojen ja niiden osien säilyttämistä varten tarvitaan tietenkin tietokanta. Vizrt käyttää Viz Artistin 3.x.-versioista lähtien Oracleen pohjautuvaa kantaa, jota hallitaan Viz Graphic Hub -nimisellä työkalulla. Kanta pitää sisällään kaikki Viz Artist 3:n elementit:

- scenet
- geometriat
- kuvat
- materiaalit
- fontit
- muut graafiset elementit ja audion.

2.2.1 Viz Artist

Artistilla rakennetaan scenejä. Sceneen kuuluvat objektipuu (Scene Tree) sekä stage.

Objektipuu on nimensä mukaan puumainen rakenne, johon rakennetaan scenen grafiikka. Siihen kootaan yhdistelmä erilaisia geometrioita (pallo, kuutio, jne.), joille voi erilaisilla plugineilla eli liitännäisillä määrittää monenlaisia ominaisuuksia (esim. läpinäkyvyys). Geometrioihin voi liittää myös materiaalin, joka määrittelee, minkä värinen kappale on ja onko sillä esimerkiksi heijastava pinta.

Kaikki geometriat sijoitetaan säiliöihin (Container), jotka sisältävät kaiken tiedon kappaleen sijainnista xyz-avaruudessa. Kaikilla säiliöillä ei tarvitse olla omaa geometriaa, vaan se voi sisältää monta lapsisäiliötä, joilla on omat geometriansa. Näin voi koostaa monimuotoisia kappaleita, jotka liikkuvat kuitenkin yhtenä yksikkönä.

Stagella on taas aikajana, johon kootaan kaikilla scenen geometrioilla olevat animaatiot. Siellä pystyy myös laukaisemaan tapahtumia ns. Action Keyframeihin sijoitettavilla Viz-scripteillä ja Viz Enginelle ohjattavilla suorilla komennoilla.

2.2.2 Viz Content Pilot ja Template Wizard

VCP ja Wizard ovat sapluunoiden käyttämiseen, hallintaan ja tekemiseen liittyvät sovellukset. Wizardilla luodaan sapluunoita sekä hallintaa kantaa, jossa ne sijaitsevat. Con-

tent pilottia taas käytetään sapluunoiden täyttämiseen ja ajamiseen sekä ajolistojen luomiseen.

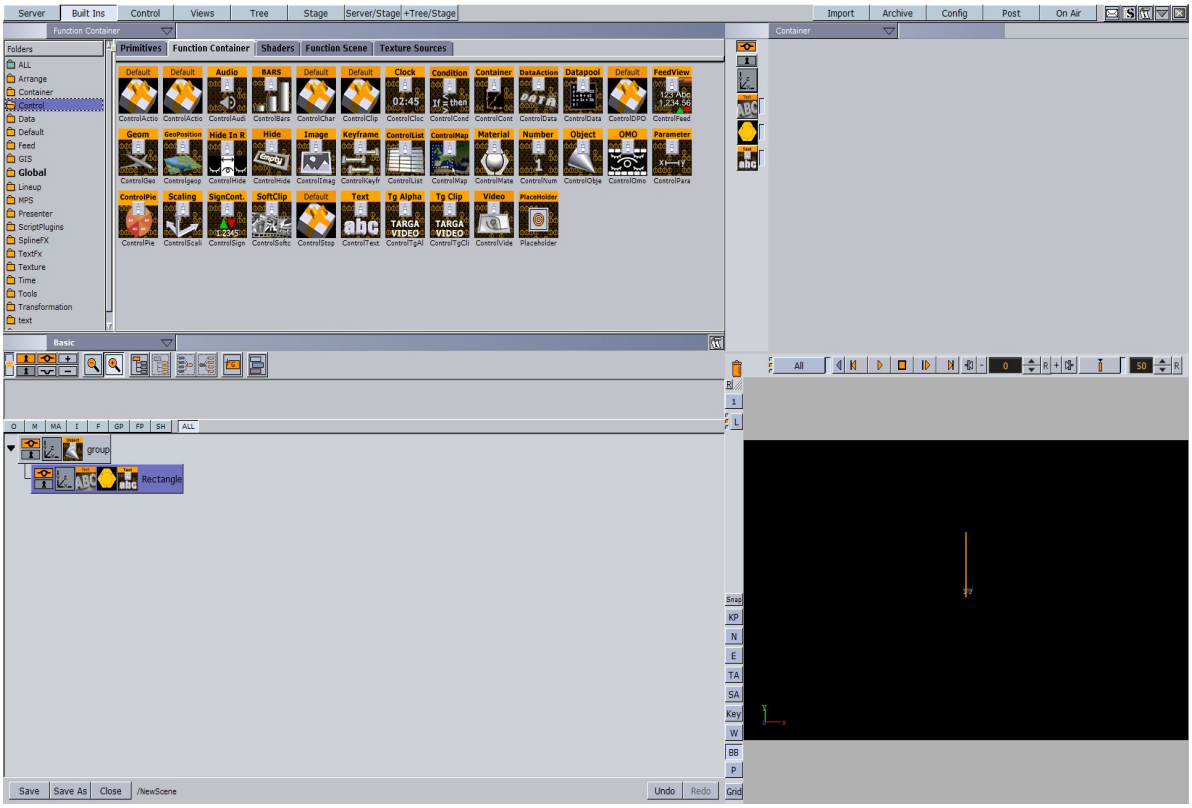
Viz-tuoteperheeseen kuuluu myös Viz Trio -niminen sovellus, joka on toinen tapa ulosajaa grafiikoita ja tehdä ajolistoja. Se eroaa Content Pilotista niin, että se ei ylläpidä omaa tietokantaansa ja sen tarkoitus on enemmänkin olla merkkigeneraattori (CG eli Character Generator), eli sillä voi avaintaa tekstiä ja grafiikoita suoraan videovirtaan. Trio oli Yleisradiolla ulosajokäytössä Content Pilotin sijaan ennen uudistusta. Nykyään Viz Engineä ohjataan Mosart -nimisen automaatiojärjestelmän avulla.

Trio ja VCP tarvitsevat Viz Enginen kanssa keskusteluun vielä yhden komponentin, ja se on Viz Media Sequencer Engine eli MSE. MSE hoitaa kommunikoinnin eri järjestelmien kuten Viz Enginen ja VCP:n välillä. Kommunikointi tapahtuu TCP/IP-liikenteenä.

3 Grafiikan ohjaaminen Vizrt:issa

3.1 Sapluunat ja scenet

Scenet ovat aiemman selityksen mukaan valmiiksi luotua grafiikkaa sekä animaatioita. Uutisten teon sujuvuuden kannalta ei kuitenkaan välttämättä haluta, että graafikko joutuu käsittelemään tai luomaan uutta grafiikkaa ennen jokaista uutislähetystä. Tätä varten tehdään grafiikanohjaus-sapluunoita.



Kuva 2. Yksinkertainen scene avoinna Viz Artistissa

Sapluunat mahdollistavat grafiikan muokkaamisen ennalta asetetuin rajoituksin kelle tahansa uutistuotantoon kuuluvalle henkilölle. Hyvä puoli on se, että henkilön ei tarvitse tietää grafiikan tekemisestä mitään. Hän vain valitsee valmiista vaihtoehdoista ja pystyy täten vaikuttamaan grafiikan ulkoasuun. Työnkulku nopeutuu ja toimittajat voivat jopa itse syöttää haluamansa uutislähetyksen sisällön grafiikkaan.

Työnkulku ohjattavan grafiikan tekemiseen menee näin:

- Ensin luodaan scene.
- Tehdään sceneen grafiikka.
- Liitetään sapluunasta hallittaviin elementteihin ohjausliitännäiset.
- Määritellään ohjausliitännäisille uniikit ohjaustunnukset(Control id).
- Luodaan uusi sapluuna Template Wizardissa olevalla velholla. Velho osaa hakea sapluunaan liitettävästä scenestä kaikki määritellyt liitännäiset tunnuksineen ja tehdä niille alustavat syöttökentät.
- Liitetään sapluunaan tarvittavat syöttökentät ja valikot ja määritellään varsinaisille ohjauskentille sama tunnus, joka määriteltiin vastaavasti scenessä oleville elementeille.

- Tehdään script-koodi, joka hallitsee sapluunan visuaalisten elementtien (esim. syöttökenttien) toimintaa.
- Määritellään variaatiot (Variant).
- Testataan toiminta.

3.2 Variaatiot

Sapluunalla voi olla useampia variaatioita. Niille määritetään Template Wizardissa olevan Template Managerin kautta sekä nimet että scenet, mihin ne on liitetty. Yksi scene voi olla liitetty useampaan sapluunaan tai useampaan variaatioon samanaikaisesti. Yksittäinen sapluuna voi myös ohjata useampaa sceneä variaatioiden avulla.

Käyttäjälle variaatiot näkyvät VCP:ssä liukuvalikkona. Tämän liukuvalikon käyttäminen laukaisee Template Wizardissa tapahtuman, joka mahdollistaa sapluunan ja grafiikan muokkaamisen valitun variaation mukaan.

3.3 Ohjausliitännäiset (control plugin)

Ohjausliitännäiset ovat Vizrt:n keino välittää tietoja sapluunasta scenelle. Tietojen välitys tapahtuu olion kautta, jonka VCP luo, kun täytetystä sapluunasta tallennetaan dataelementti. Kaikille välitettäville tiedoille määritetään oma uniikki ohjaustunnus (control id).

Liitännäisillä on mahdollista välittää tekstiä, kuvia ja erilaisia komentoja suoraan scenelle. Niillä pystyy myös hallitsemaan Vizrt:in tekemiä valmiita liitännäisiä sceneissä.

Hyvänä esimerkkinä tästä voidaan käyttää Omo-liitännäistä ja siihen yhdistettävää Omo-ohjausliitännäistä. Omo liitetään säiliöön niin kuin kaikki muutkin liitännäiset ja sen jälkeen sillä on mahdollista hallita kyseisen säiliön alisäiliöiden aktiivista tilaa eli ovatko säiliöiden graafiset elementit näkyvissä ja toimivatko siihen liitetyt liitännäiset. Omo-liitännäistä on hyvä käyttää esimerkkinä, koska se on käytetyin liitäntä variaatioiden hallintaan.

Valmiita liitännäisiä on myös helppo ja erittäin suositeltavaa käyttää Viz scriptin kanssa. Toisin sanoen ei kannata alkaa keksiä pyörää uudestaan, vaan valmista liitännäiskirjastoa kannattaa ehdottomasti käyttää oman koodin tukena.

Liitännäisiä löytyy useaa eri lajia, mutta suurin osa niistä liittyy säiliöissä olevien kappaleiden käyttäytymiseen tai niiden animoimiseen. Liitännäisistä enemmän tietoa haluava voi käydä tutkimassa Vizrt:n virallisen sivuston tukiosioita, josta löytyvät kaikki julkisesti saatavilla olevat Viz Artist 3.xx -ohjekirjat.

3.4 Ohjelmoinnin merkitys

Koodaaminen on tärkeä osa grafiikan hallintaa Vizrt-ympäristössä. Se avaa useita mahdollisuuksia, joita Vizrt:n valmiit toiminnot eivät anna. Esimerkkeinä tässä on muutamia konkreettisesti käytössä olevia asioita:

- grafiikan dynaamisuus
- parempi grafiikan hallinta
- ohjaussapluunoiden joustavuus
- käskyt suoraan enginelle
- stagen hallinta
- valmis data suoraan grafiikkaan (esim. säätiedot)
- automaattinen sisällönhallinta.

Nämä osa-alueet käydään tarkemmin läpi seuraavissa aliluvuissa.

3.4.1 Grafiikan dynaamisuus ja hallinta

Dynaamisuudella tarkoitetaan sitä, että graafiset elementit muokkautuvat silloin, kun engine näyttää graafista sisältöä. Tästä esimerkkinä voi käyttää myöhemmin esiteltävässä modulaarisessa mallissa käytettyä grafiikkaa. Sen liitännäisten hallintaan ei ole valmista ohjausliitännäistä, joten elementtien dynaaminen asemointi on pakko hoitaa koodaamalla.

Toinen asia, missä dynaamisuus konkretisoituu, on kosketusnäyttögrafiikka. Käyttäjä antaa studion kosketusnäytöltä komentoja, joihin grafiikan täytyy reagoida. Silloin sceneen täytyy olla mahdollista luoda uusia objekteja ja animaatioita suoraan lennosta.

3.4.2 Ohjaussapluunoiden joustavuus

Osa yksinkertaisimmista sapluunoista on mahdollista tehdä ilman skriptaamista. Tälläinen sapluuna tarkoittaa yleensä tekstin muokkaamista scenestä.

Sceneen on luotu valmiilla fontilla teksti, jota vaihdetaan Control Text -nimisen liitännäisen avulla. Tällöin sapluunassa tarvitsee olla vain tekstikenttä, jolle on määritelty sama ohjaustunnus, kuin Control Text -liitännäiselle.

Ohjelmointi tulee tarpeeseen, jos halutaan, että sapluunan käyttäjä pystyykin vaihtamaan esimerkiksi tekstin asemointia grafiikassa. Yksi luonnollisimmista tavoista on määrittää sapluunalle yksi tai useampi variaatio, jonka vaihtaminen sitten laukaisee OnVariantChange-tapahtuman. Myös variaation nimi tallentuu Template Wizardin omaan SC_VariantName -nimiseen muuttujaan.

SCV_OnVariantChange-tapahtuma käsitellään samannimisessä alifunktiossa, jossa voidaan vaikka select case -tyyppisellä rakenteella käsitellä variaation nimen mukainen toiminta. Sapluunoiden elementteihin on myös mahdollista liittää näppäimistön ja hiiren kuuntelijoita, sekä niistä on mahdollista laukaista tapahtumia, kun käyttäjä vaikuttaa niihin jollain tavalla.

3.4.3 Käskyt enginelle

Ennen Artistin versiota 3.xx engine-käskyt olivat ainut tapa hallita grafiikkaa dynaamisesti. Nykyään niiden käytöstä pyritään siirtymään kokonaan pois, koska Vizrt ei virallisesti tue järjestelmän ulkopuolisia sovelluksia. Ulkopuolista sovellusta tarvitaan enginekomentojen lähettämiseen tcp/ip:n välityksellä suoraan enginelle. Niillä on mahdollista hallita enginen toimintaa aivan täydellisesti, esimerkiksi mitä animaatioita stagelta käynnistyy.

Itse asiassa enginen toiminta perustuu lähes kokonaan tähän käskykieleen ja avaamalla enginen konsoli-ikkunan näkyville voi huomata, että kaikki komennot, jotka käyttäjä antaa graafisesta käyttöliittymästä, tulostuvat tuohon konsoliin.

3.4.4 Stagen hallinta

Stagelle voidaan sijoittaa avainkehyksiä (Keyframe), jotka laukaisevat erilaisia tapahtumia. Näitä kutsutaan toiminta-avainkehyksiksi. Niihin voi sijoittaa edellä mainittuja enginen komentokäskyjä, joilla voidaan esimerkiksi pysäyttää animaatio tai muuttaa stagelta näytettävää fieldiä, joka on Artistissa käytetty aikayksikkö ja merkitsee 1/50 sekuntia.

Toiminta-avainkehyksestä on myös mahdollista laukaista Viz scriptillä tehtyjä funktioita ja alifunktioita.

3.4.5 Data suoraan grafiikkaan ja automaatio

Sapluuna-ohjelmoinnissa käytettävällä VBScriptillä ja Artistin omalla VizScriptillä on molemmilla mahdollista lukea tekstisisältöisiä tiedostoja. Tavallisin käyttötapa tälle toiminnolle on lukea xml-formaatissa olevaa dataa. Näin pystytään toimitusjärjestelmän tuottamaan dataa käyttämään suoraan grafiikassa ilman, että kaikkia tilastotietoja tarvitsee manuaalisesti kirjoittaa.

Toisin sanoen sapluuna voidaan koodata lukemaan datatiedostoa, antaa käyttäjän tarkistaa tuleva data ja syöttää data suoraan grafiikkaan parilla näppäimen painalluksella.

Vastaavaa automaatiota voidaan käyttää scenessä silloin, kun grafiikan ajamiseen käytetään sellaista järjestelmää, jossa ei ole mahdollista käyttää Template Wizardilla tehtyjä sapluunoita. Esimerkkinä Ylen kanavilla täyteohjelmana pyörivä Uutisikkuna on toteutettu tällä tavalla.

4 Ohjemointi Vizrt-ympäristössä

4.1 Sapluunoiden ohjelmointi

Sapluunoiden ohjelmointiin käytetään Visual Basic Script -nimistä scriptikieltä. Se on Microsoftin luoma ja ylläpitämä oliopohjainen kieli, joka toimii suoraan Windows-alustalla. Windows osaa kääntää koodin suoraan ".vbs"-päätteisistä tiedostoista eikä täten koodia tehdessä tarvitse erikseen kääntäjää. VBS on myös ollut käytössä internet sivustojen luomisen yhteydessä ennen kuin JavaScriptiä ruvettiin käyttämään sen sijaan.

VBS tarjoaa helpon tavan koodata sapluunoita ja siinä on suhteellisen hyvä kirjasto tarvittavine elementteineen. Syntaksin erikoisuutena mainittakoon, että VBS-ohjelmoinnissa ei käytetä puolipistettä koodirivin lopettamiseen. Tästä johtuen komennot on aina pakko laittaa kokonaan samalle riville. Myöskään aaltosulkeita ei käytetä, vaan funktiot, alifunktiot, silmukat ja ehtorakenteet merkitään sanoin. Esimerkkinä "If-Else"-rakenne, joka näkyy esimerkissä.

```
If ehto Then
                jotain tapahtuu...
Else
 jotain muuta tapahtuu...
End if
```

Template Wizard sisältää joukon valmiita tapahtumankäsittelijöitä, joilla luodaan sapluunoiden toiminnallisuus. Sapluunoihin on mahdollista tehdä muun muassa

- nappuloita
- tekstikenttiä
- raksittavia valintalaatikoita
- valintalistoja.

Näillä komponenteilla on muiden ohjelmointiympäristöjen tapaan omat vapaasti määriteltävät ominaisuutensa ja niille on mahdollista määrittää tapahtumankäsittelijöitä.

Yksi yleisimmistä tavoista käyttää skriptiä on kuvien vaihtaminen. Sapluunaan luodaan "Image with Name and Title Linking"-niminen komponentti ja määritellään sille "On-

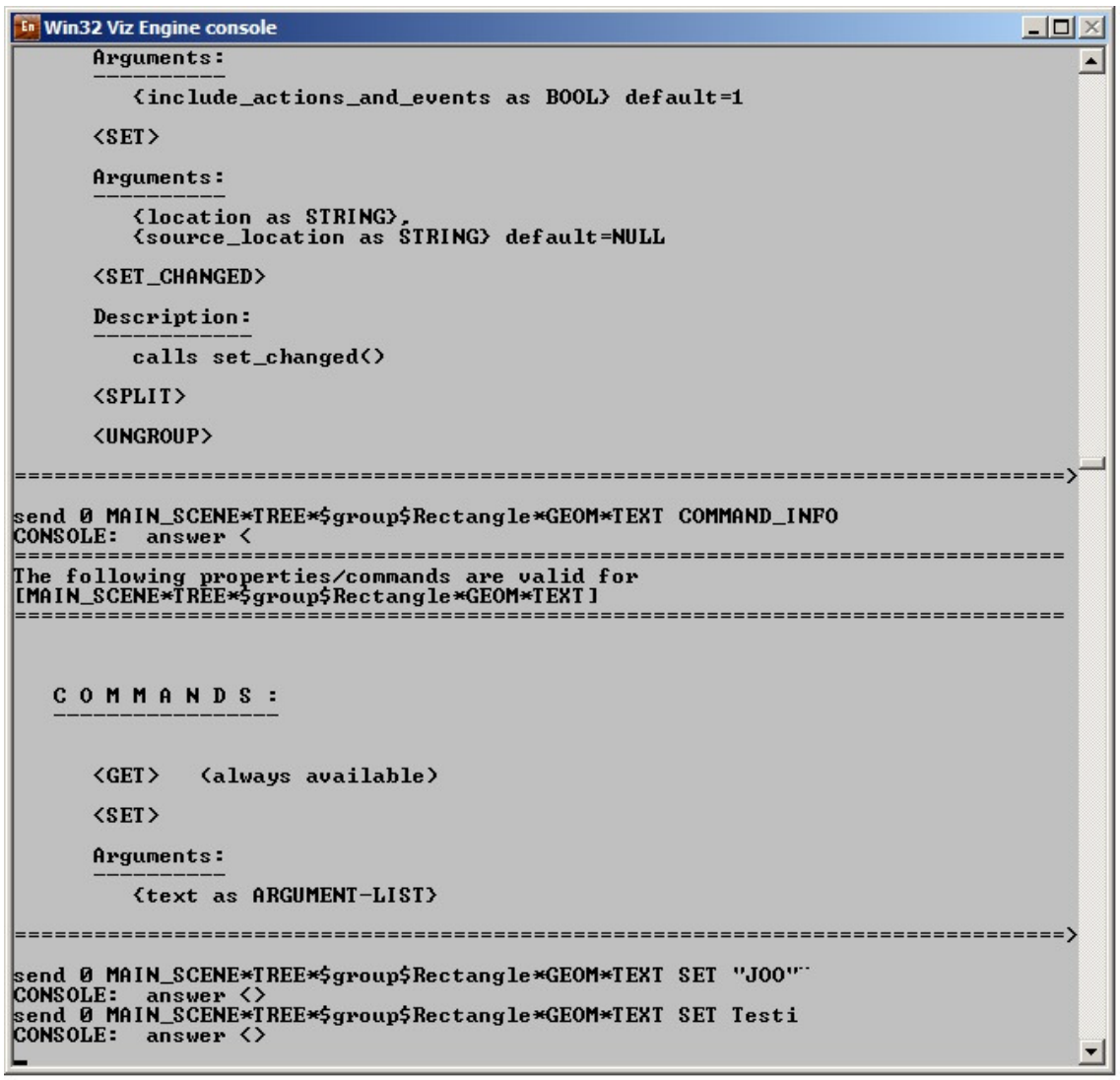
DoubleClick”-tapahtuma, joka avaa Windowsin tiedostoselaimen kuvan noutamista varten. Kuvakomponentille määritetään myös ohjaustunnus, joka linkitetään scenessä olevaan kuvasäiliöön ”ImageControl”-liitännäisen avulla. Nyt kun kaikki on valmista, voidaan avata VCP:llä tehty sapluuna, vaihtaa siihen haluttu kuva, ja painaa VCP:n käyttöliittymästä ”Start”-nappulaa. Nappula antaa enginelle käskyn aloittaa scenen toistaminen ja samalla kuvan tiedot välittyvät scenelle. Nyt toistettavassa grafiikassa näkyy sapluunaan valittu kuva.

Kaikki lisättävissä olevat stardardi-komponentit ja niiden ominaisuudet löytyvät Template Wizardin ohjekirjasta.

4.2 Komentokäskyt enginelle

Viz Enginen komentokieli ei ole yhtään monimutkaisempaa käyttää kuin mikä tahansa skriptikieli. Komentokielen ainoana ongelmana on se, että sitä ei ole kunnolla dokumentoitu, joten ohjelmoijan täytyy opetella komennot kokeilemalla tai syöttämällä konsoliin ”COMMAND_INFO”-komento, joka tulostaa kaikki mahdolliset käytössä olevat komennot.

Hyvä niksi komentokielen opetteluun on avata Viz Enginen komentoikkuna ja tehdä jotain Artistin visuaalisesta käyttöliittymästä. Jokainen toiminto, minkä engine tekee, tulostuu komentoikkunaan ja on pienin muutoksin käytettävissä ohjelmoitaessa.



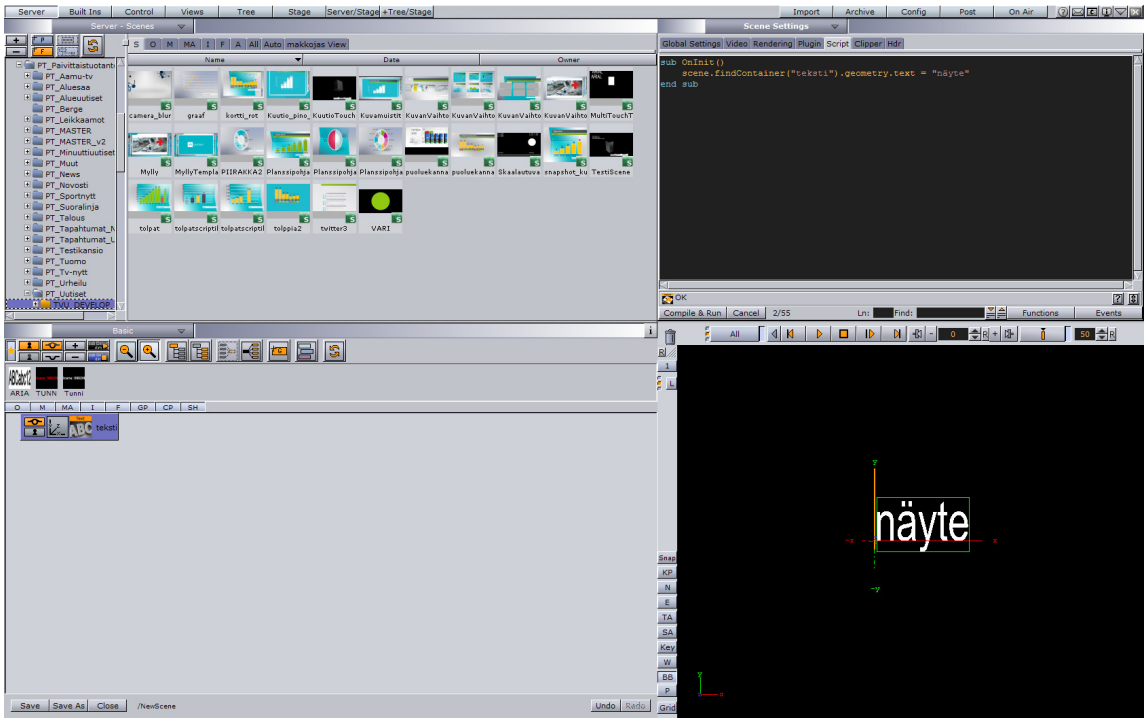
Kuva 3. Yksinkertainen konsolikomento

Komentokäskyjä on mahdollista antaa enginelle sen omasta konsolista, ohjaussapluunasta tai ulkopuolisesta sovelluksesta tcp/ip-protokollaa käyttäen. Käskyjen kirjoitusmuoto vaihtelee käskyn antajan sijainnista riippuen. Käskyjä voidaan kirjoittaa myös stagelle avainkehyksiin, ja se onkin yksi tapa laukaista skriptattuja funktioita. Tässä tapauksessa käsky olisi muotoa ”THIS_SCENE*SCRIPT INVOKE haluttuFunktio”.

4.3 Ohjelmointi Viz Artistissa

Viz Artist -ohjelmointi tapahtuu sen sisällä olevassa skriptieditorissa. Skriptin voi joko liittää liitännäisenä olemassa olevaan säiliöön, jolloin scriptillä voi helposti muokata suoraan säiliön ominaisuuksia ja tehdä säiliöstä vaikkapa painettavan nappulan koske-

tusnäyttögrafiikkaa varten. Toinen paikka skriptille on yleisesti scenen tasolla oleva editori.



Kuva 4. Oikealla ylhäällä näkyy avoinna oleva skriptieditori

Editorissa olevalla kysymysmerkkipainikkeella saa auki scriptidokumentaation. Siellä on lueteltuna kaikki käytettävissä olevat funktiot ja tapahtumat. Dokumentaatio sisältää myös pienen ohjeen ja muutamia esimerkkejä skriptaamisen aloittamiseksi.

5 Vizrt:n oma skriptikieli

Artistin sisällä olevaa skriptikieltä kutsutaan VizScriptiksi. Sen syntaksi pohjautuu paljolti Visual Basic Scriptiin, mutta se sisältää myös elementtejä javascriptistä. Kieli on hyvin kompakti ja sisältää yhteensä ainoastaan 43 datatyyppiä ja proseduuria. Tapahtumankäsittelijöitä on 33 kappaletta.

5.1 Datatyypit

Useimmat datatyypit ovat tuttuja muista kielistä, tässä kuitenkin lista:

- Boolean
- Integer
- Double
- Vertex
- Matrix
- Array[type]
- Color
- String.

Käyttäjä voi myös itse määrittää datatyyppejä struktuureina esimerkin tyyliin.

```
Structure XmlNoodi
 nimi As String
 sisalto As String
 lapsinoodit As Array[XmlNoodi]
End Structure
```

Struktuuri alustetaan kuten muutkin muuttujat.

```
Dim noodi As XmlNoodi
noodi.nimi = "Esimerkki"
noodi.sisältö = "Näin käytetään struktuuria"
```

5.2 Proseduurit

Proseduureja on kahta tyyppiä. Ne ovat Sub ja Function. Ensimmäiselle ei tarvitse määrittää paluuarvoa ja toinen on paluuarvollinen funktio.

```
Sub alifunktio(parametrit)
 koodi
End Sub

Function funktio(parametrit) as Paluutyyppi
 koodi
 funktio = paluuarvo
End Function
```

5.3 Oliot

Tässä lista VizScriptin kirjastossa olevista olioista:

- Alpha
- Base
- BezierHandle
- Camera
- Center
- Channel
- ClipChannel
- Container
- DateTime
- Director
- Expert
- Geometry
- Grid
- Image
- InfoText
- Key
- Keyframe
- Light
- Material
- PluginInstance
- Position
- Rotation
- Scaling
- Scene

- Script
- ScriptSettings
- SharedMemory
- Stage
- StringMap
- System
- Texture
- Timecode
- Uuid
- Variant
- VizCommunication.

Näiden lisäksi on vielä globaaleja proseduureja, joita voi kutsua viittaamatta näihin olioihin. Esimerkkinä vaikkapa println-proseduuri, joka tulostaa enginen konsoliin annetun parametrin. Kaikilla olioilla on ominaisuuksia ja jäsenproseduureja, joilla pystyy vaikuttamaan olion ominaisuuksiin.

Seuraavaksi käydään läpi muutamia ohjelmoinnissa eniten käytettyjä olioita.

Alpha-oliolla hallitaan säiliön läpinäkyvyysarvoa. Läpinäkyvyyden aikaansaamiseksi säiliöön tulee liittää Alpha-liitännäinen.

Camera-oliolla hallitaan useita kameroita, joita voi liittää sceneen.

Container-oliolla viitataan scenessä oleviin säiliöihin ja niiden ominaisuuksiin. Scenessä olevaan säiliöön pääsee käsiksi Scene.FindContainer(”haettavan_nimi”) komennolla. Tästä syystä on tärkeää, että kaikki säiliöt nimetään uniikein nimin, jotta scriptistä pääsee niihin helposti käsiksi.

Director-olion kautta pääsee käsiksi stagella oleviin avainkehyksiin. Komento on Director.findKeyframe(”keyframen nimi”). Directorin kautta voi myös käynnistää animaatioita Director.StartAnimation() -komennolla.

Scene-olion kautta pääsee käsiksi objektipuussa oleviin säiliöihin.

Script-oliolla viitataan säiliössä tai scenen tasolla olevaan skriptiliitännäiseen.

Stagen kautta pääsee käsiksi siellä oleviin directoreihin komennolla Stage.FindDirector(”directorin_nimi”).

5.4 Tapahtumankäsittelijät

Tapahtumankuuntelijoita on yhteensä 32 kappaletta. Niistä tärkein on OnInit(), joka laukeaa, kun skriptiliitännäisen koodi käännetään ja ajetaan tai kun scene avataan engineen.

Suurin osa tapahtumankuuntelijoista liittyy kosketusnäytön käyttämiseen. Niihin kuuluvat hiiritapahtumat ja TUIO rajapintaa käyttävät multitouch-tapahtumat.

Tapahtumankuuntelijat on paremmin selitetty skriptidokumentaatiossa, jonka saa auki Artistin kautta.

6 Viz ja kosketusnäyttö

Viz mahdollistaa kaksi tapaa tehdä kosketusnäyttögrafiikkaa. Toinen on hiiritapahtumia käyttäen ja toinen on käyttäen TUIO-protokollaa.

Ensimmäinen tapa vaatii enginen esikatseluikkunan koon muuttamista kokoruutuun. Kosketusnäytön emuloimat hiiritapahtumat sitten vaikuttavat esikatseluikkunan kosketuspintaan. Hiiritapahtumien kanssa käytetään tapahtumankuuntelijoina OnLButtonDown() ja OnLButtonUp(). Ne siis nimensä mukaan laukeavat, kun hiiren vasenta nappulaa painetaan. Hiiritapahtumia käyttäen on mahdollista käyttää vaan yksittäistä kosketusta. Kappaleiden skaalaaminen kahdella sormella ei siis onnistu.

Skaalaamista varten voidaan sitten käyttää edellä mainittua TUIO-protokollaa. Se vaatii toimiakseen TUIO-protokollaa tukevan kosketusnäytön ja VizMultitouchServerin, joka on oma erillinen ohjelmansa. Multitouchserver hoitaa keskustelun kosketusnäytön ja Vizin TUIO-rajapinnan kanssa. Siihen määritetään oikea ip-osoite ja porttinumero, että homma alkaa toimimaan. Kun asetukset on saatu kuntoon, voidaan alkaa käyttää skriptissä tapahtumankuuntelijoina:

- OnMTHit(stroke as Integer, x as Integer, y as Integer)
- OnMTMenu(x as Integer, y as Integer)
- OnMTControlPZR2D(x as Integer, y as Integer, rot as Vertex, scale as Vertex, pressure as Double)
- OnMTControlButton(strokes as Integer, pressure as Double)
- OnMTControlInactive().

OnMTHit laukeaa, kun käyttäjä koskee multitouch laitteella objektia, jonka scriptissä tapahtumankuuntelijaa on käytetty. Stroke on multitouch-tapahtuman id numero, x ja y kertovat kosketuksen kohdan ruudun koordinaateissa. Käytetään rekisteröimään ohjain multitouch serverille.

```
Sub OnMTHit(stroke As Integer, x As integer, y As Integer)
 MTRegister(stroke, 1)
End sub
```

Ohjain voi olla joko nappula tai liikuteltava objekti. Se määritellään numerolla, jonka arvot voivat olla 1=nappula, 2=liikutetava objekti ja 3=liikuteltava objekti kallistuksella.

Ohjaimen asettamisen jälkeen käytetään OnMTControlPZR2D- tai OnMTControlButton –tapahtumankuuntelijoita toimintojen toteuttamiseen.

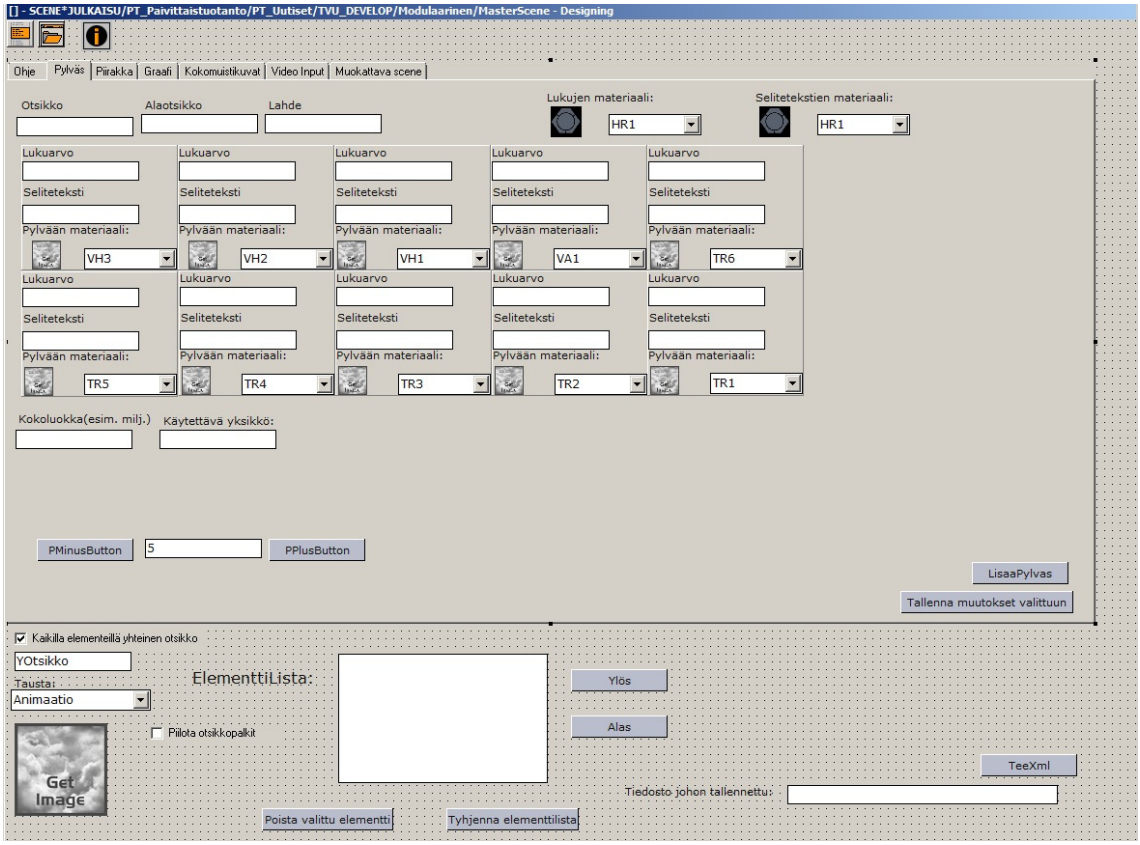
```
Sub OnMTControlPZR2D(x As Integer, y As Integer, rot As Vertex, scale As Vertex, pressure
As Double)
 this.Position.x = x
 this.Position.y = y
 this.Rotation.xyz = rot
 this.Scaling.xyz = scale
End Sub

Sub OnMTControlButton(strokes As Integer, pressure As Double)
 jotain tapahtuu...
End Sub
```

Esimerkissä vaikutetaan skriptiliitännäisen sisältävän säiliön paikkaan, rotaatioon ja skaalaukseen.

7 Modulaarinen malli

Modulaarinen malli on scenejen ja templaattien yhdistelmä, jonka tein Ylelle yhteistyössä Annette Stenroosin kanssa. Sen tekeminen vaati n. 2-3 työkuukautta. Tämän tyyppistä tai näin suuritöistä scenen ja templaatin yhdistelmää ei ole aikaisemmin Ylellä tehty. Työn pointtina oli se, että voitiin esittää erilaista dataa graafisessa muodossa modulaarisesti.

Käyttäjä voi tehdä oman soittolistan, jossa voi olla perätysten kuvia, käyriä, piirakkagrafiikkaa ja pylväsgrafiikkaa. Se koostuu useasta scenestä, joita ohjataan master scenen kautta. Master scene sisältää tausta-animaation tai taustakuvan, ja muu grafiikka näkyy ohjattavissa sceneissä. Master scene toistetaan enginen keskimmäisellä kerroksella(middle layer) ja ohjattavat scenet ovat päällimmäisellä kerroksella (front layer).



Kuva 5. Modulaarisen mallin sapluuna

Sapluunan kautta kootaan xml-tiedosto, joka sisältää kaiken tarvittavan datan grafiikan muodostamista varten. Tiedoston sijainti välitetään master scenelle ohjausliitännäisen kautta.

Modulaarisen mallin avulla voi esittää pylväsgrafiikkaa, piirakkagrafiikkaa, käyriä, kuvia, videota, ja se sisältää scenen, jota graafikko voi muokata haluamallaan tavalla ennen grafiikan näyttämistä.

7.1 Sapluunan rakenne

Sapluunassa on välilehti jokaista grafiikkavaihtoehtoa varten. Jokaisella välilehdellä on tarvittavat syöttökentät ja valinnat sekä nappula, joka lisää graafisen elementin sapluunan elementtilistaan. Elementtilistassa olevia elementtejä voi hiiren avulla selata läpi ja niihin voi tallentaa muutoksia tarvittaessa. Elementtilistasta luodaan xml-tiedosto, joka sisältää kaikki sapluunaan asetetut arvot.

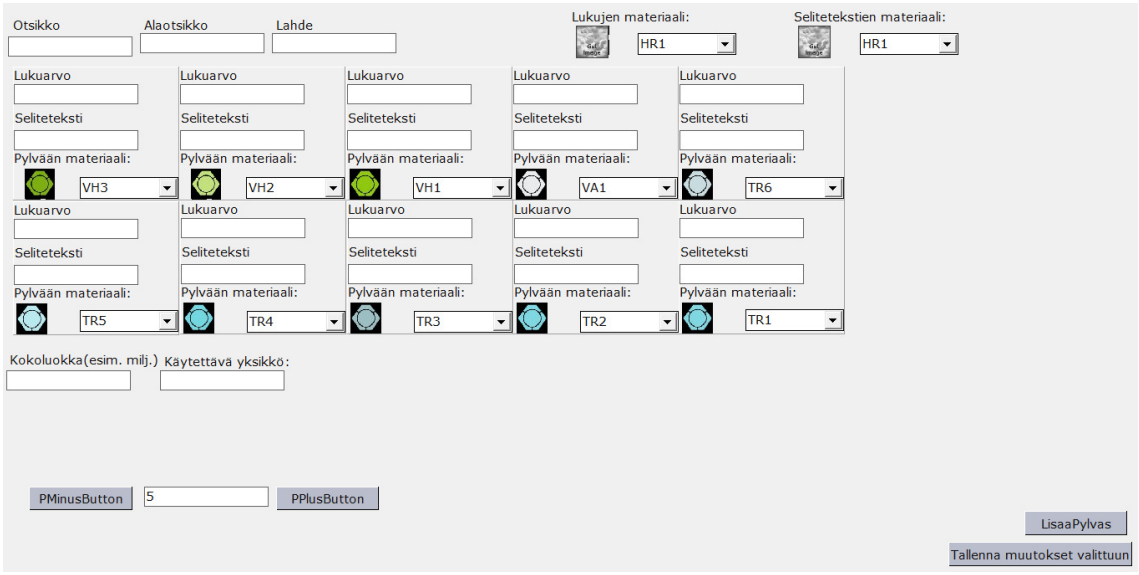
Sapluunasta pystyy myös valitsemaan taustan. Se on joko tyhjä, animaatio tai valittu taustakuva. Kaikille grafiikoille yhteisen otsikon voi myös asettaa näkyviin sapluunasta.

7.1.1 Pylväs

Sapluunasta voi asettaa pylväsgrafiikalle:

- otsikon
- alaotsikon
- lähteen
- 1-10 pylvästä
- pylväille materiaalit
- pylvään lukuarvon
- pylvään selitetekstin.

Käyttäjän ei tarvitse huolehtia pylväiden pituuksista mitään, koska scenessä on skripti, joka laskee pylväiden pituudet ja niiden lukuasteikon lukuarvojen mukaan.



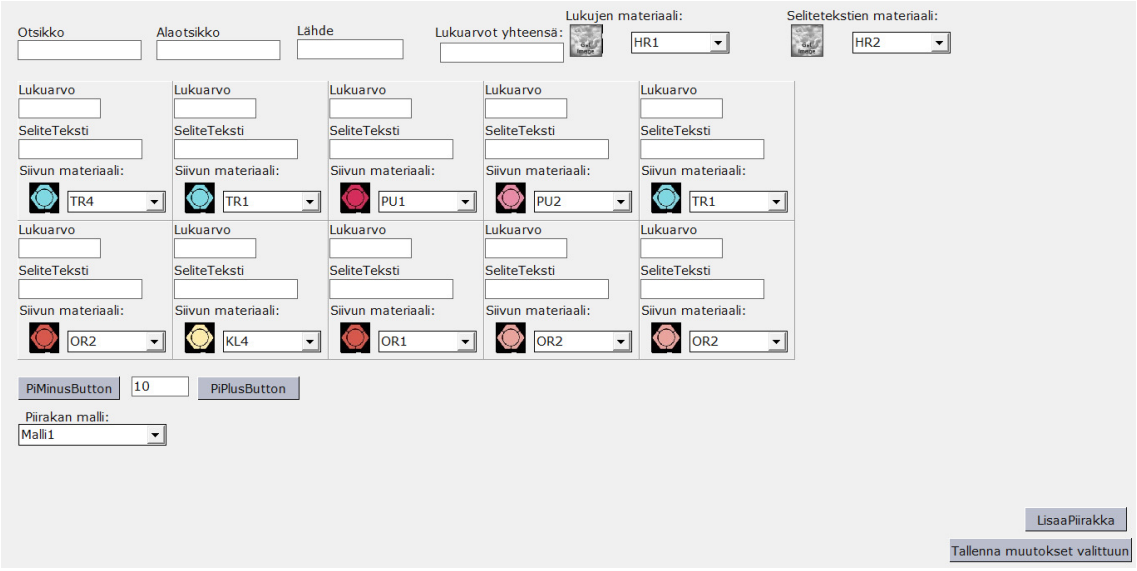
Kuva 6. Pylväät sapluunassa

7.1.2 Piirakka

Sapluunasta voi asettaa piirasgrafiikalle:

- otsikon
- alaotsikon
- lähteen
- 2-10 viipaletta
- materiaalit viipaleelle, lukuarvoille ja selitetekstille
- piirakan mallin
- selitetekstin jokaiselle viipaleelle
- lukuarvon jokaiselle viipaleelle.

Piirakka muodostetaan scenessä valmiilla liitännäisellä. Siinä on mahdollista olla yhteensä kymmenen viipaletta.



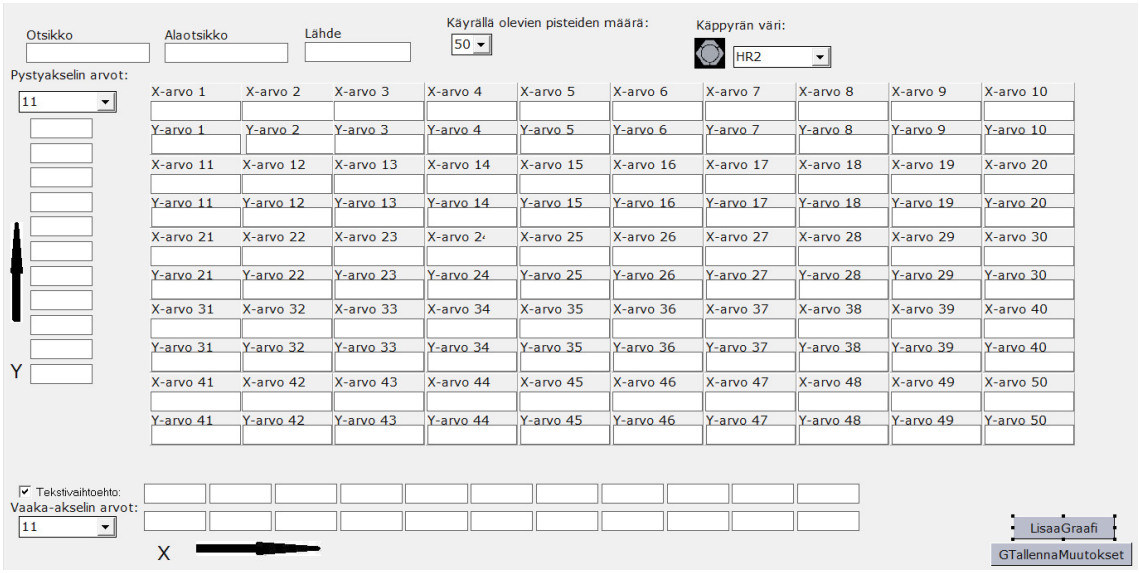
Kuva 7. Piirakka sapluunassa

7.1.3 Käyrä

Sapluunasta voi asettaa käyrälle

- otsikon
- alaotsikon
- lähteen
- yhteensä 50 pistettä
- lukuasteikon x- ja y-suunnassa
- käyrän materiaalin.

Käyrä luodaan scenessä valmiilla liitännäisellä, johon on mahdollista asettaa enintään 50 x- ja y-arvoa.



Kuva 8. Käyrä sapluunassa

7.1.4 Kokomuistikuvat

Sapluunasta voi ladata 1-10 kuvaa näytettäväksi grafiikassa.

Kuva 9. Kuvat sapluunassa

7.2 Sapluunan koodi

Sapluunassa on yhteensä melkein tuhat riviä koodia. Suurin osa siitä on tapahtuman kuuntelijoihin sijoitettua. Tarkoituksena on käydä läpi pääpiireittäin, mitä sapluunan koodi sisältää ja miten sapluunan on tarkoitus toimia.

7.2.1 Initform

Initform on template wizardissa määritelty funktio, joka laukeaa, kun sapluuna avataan Content Pilotissa.

```
'Kun templaatti avataan, tarkistetaan onko siinä jo tallenettu xml-tiedosto(eli se on tallennettu data-
'elementtinä). Jos näin on, niin ladataan kaikki elementit xml-tiedostosta elementtilistaan.
sub InitForm
    Set elementArrayList = CreateObject("System.Collections.ArrayList")
    materialPath = "MATE-
RIAL*JULKAISU/PT_Paivittaistuotanto/PT_Uutiset/TVU_DEVELOP/material/"
    bitmapPath = "//prod-vizrt/vizfiles/MateriaaliTesti/bmp/"
    thumbPath = "//prod-vizrt/vizfiles/MateriaaliTesti/"
    if SavedFilePath.Text <> "" then
        lataaLista()
    end if
    GMaaraTemp = CInt(GMaara.text)
end sub
```

Tässä funktiossa alustetaan elementArrayList, joka pitää sisällään kaikki sapluunasta luodut graafiset elementit. Asetetaan myös polku materiaaleille ja katsotaan, avataanko sapluuna dataelementistä, jolloin datatiedosto on jo luotu. Jos datatiedosto on luotu, suoritettaan laataaLista-funktio, joka avaa datan xml-tiedostosta ja asettaa sen sapluunan elementArrayListiin.

7.2.2 LataaLista

LataaLista-funktio purkaa luodun xml-tiedoston ja asettaa sen sapluunan elementtilistaan.

```
'Ladataan elementtilista olemassa olevasta xml-tiedostosta.
Sub lataaLista()
    ElementList.Items.clear
    Dim filePath
    filePath = SavedFilePath.Text
```

```
    Dim xmlDoc, objNodeList, otsikko

    Set xmlDoc = CreateObject("Msxml2.DOMDocument")
    xmlDoc.load(filePath)

    Set Root = xmlDoc.documentElement
    For Each x In Root.childNodes

        if x.nodename = "YOtsikko" then
            YOtsikko.text = x.text
        elseif x.nodename = "Tausta" then
            Tausta.itemindex = CInt(x.text)
        elseif x.nodename = "TaustaKuva" then
            TaustaKuva.picFilename = x.text
            TaustaKuva.thumbFilename = x.text
        elseif x.nodename = "PiilotaPalkit" then
            if x.text = "0" then
                HidePalkit.checked = false
            else
                HidePalkit.checked = true
            end if

        else
            Set KeyArray = CreateObject("Scripting.Dictionary")

            for each y In x.childNodes
                if y.nodename = "otsikko" then
                otsikko = y.text
                end if
                KeyArray.add y.nodename, y.text
            next
            if x.nodename = "pylvas" then
                ElementList.items.add("Pylvas - " & otsikko)
            elseif x.nodename = "piirakka" then
                ElementList.items.add("Piirakka - " & otsikko)
            elseif x.nodename = "graafi" then
                ElementList.items.add("Graafi - " & otsikko)
            elseif x.nodename = "kuvat" then
                ElementList.items.add("Kuvat - " & otsikko)
            elseif x.nodename = "video" then
                ElementList.items.add("Video - " & otsikko)
            elseif x.nodename = "muokattava" then
                ElementList.items.add("Muokattava - " & otsikko)
            else
                ElementList.items.add("Tässä on vikaa")
            end if
            elementArrayList.add KeyArray
        end if
    Next

End sub
```

7.2.3 Lisäys elementtilistaan

Tässä on pylväsgrafiikan lisäysfunktio, joka on kaikille graafisille elementeille on samanlainen.

```
'Lisätään luotu pylväs elementtilistaan
Sub LisaaPylvasClick(Sender)

	Set pylvasKeyArray = CreateObject("Scripting.Dictionary")
	pylvasKeyArray.add "otsikko", POtsikko.text
	pylvasKeyArray.add "alaotsikko", PAlaotsikko.text
	pylvasKeyArray.add "lahde", PLahde.text
	pylvasKeyArray.add "PLukuMaterial", PLukuMaterial.picFilename
	pylvasKeyArray.add "PSeliteMaterial", PSeliteMaterial.picFilename
	pylvasKeyArray.add "maara", PMaara.text
	pylvasKeyArray.add "yksikko", Yksikko.Text
	pylvasKeyArray.add "kokoluokka", KokoLuokka.Text
	for i = 1 to CInt(PMaara.text)
		pylvasKeyArray.add "P" & i & "LukuArvo", eval("P" & i & "LukuArvo").text
		pylvasKeyArray.add "P" & i & "SeliteTeksti", eval("P" & i & "SeliteTeksti").text
		pylvasKeyArray.add "P" & i & "PylvasMaterial", eval("P" & i & "PylvasMateri-
al").picFilename
	next
	Dim mj, keys, items
	mj = ""
	keys = pylvasKeyArray.Keys
	items = pylvasKeyArray.Items

	elementArrayList.add pylvasKeyArray
	ElementList.items.add("Pylvas - " & POtsikko.text)

	'lopuksi tyhjennetään syöttökentät
	tyhjennaPylvaanSyottoKentat()
End sub
```

7.2.4 Xml-tiedoston luonti

Seuraava funktio purkaa luodun elementtilistan xml-tiedostoon.

```
'Luodaan xml-tiedosto kaikista elementtilistaan tallennetuista elementeistä.
Sub TeeXmlClick(Sender)
	Dim DataOut, elementinNimi
	DataOut = ""
	'Xml-file alkaa
	DataOut = DataOut & "<?xml version=""1.0"" encoding=""utf-8""?>" & vbCrLf
	DataOut = DataOut & "<root xmlns:ns1=""http://www.w3.org/2001/XMLSchema"">" &
vbCrLf
	if OtsikkoRuksi.Checked then
		DataOut = DataOut & vbTab & "<YOtsikko>"
		DataOut = DataOut & YOtsikko.text
```

```
        DataOut = DataOut & "</YOtsikko>" & vbCrLf
    end if
    if HidePalkit.Checked then
        DataOut = DataOut & vbTab & "<PiilotaPalkit>"
        DataOut = DataOut & "1"
        DataOut = DataOut & "</PiilotaPalkit>" & vbCrLf
    else
        DataOut = DataOut & vbTab & "<PiilotaPalkit>"
        DataOut = DataOut & "0"
        DataOut = DataOut & "</PiilotaPalkit>" & vbCrLf
    end if
    DataOut = DataOut & vbTab & "<Tausta>"
    DataOut = DataOut & Tausta.ItemIndex
    DataOut = DataOut & "</Tausta>" & vbCrLf
    if Tausta.itemindex = 0 then
        DataOut = DataOut & vbTab & "<TaustaKuva>"
        DataOut = DataOut & TaustaKuva.picFilename
        DataOut = DataOut & "</TaustaKuva>" & vbCrLf
    end if

    for i = 0 to elementArrayList.Count-1
        Dim stringi
        stringi = ElementList.Items.AnsiStrings(i)
        Select Case Mid(stringi, 1, InStr(stringi, " ")-1)
            Case "Pylvas"
                elementinNimi = "pylvas"
            Case "Piirakka"
                elementinNimi = "piirakka"
            Case "Graafi"
                elementinNimi = "graafi"
            Case "Kuvat"
                elementinNimi = "kuvat"
            Case "Video"
                elementinNimi = "video"
            Case "Muokattava"
                elementinNimi = "muokattava"
        End Select
        DataOut = DataOut & vbTab & "<" & elementinNimi & ">" & vbCrLf
        Dim keys, items
        keys = elementArrayList.Item(i).Keys
        items = elementArrayList.Item(i).Items
        for u = 0 to elementArrayList.Item(i).Count-1

            DataOut = DataOut & vbTab & vbTab & "<" & keys(u) &">" & items(u) & "</"
& keys(u) & ">" & vbCrLf
        next
        DataOut = DataOut & vbTab & "</" & elementinNimi & ">" & vbCrLf
    next
    DataOut = DataOut & "</root>"
    'Xml-file loppuuu
    Dim fsT, OutFile
    OutFile                                    =                                    "\\prod-
vizrt\vizfiles\JULKAISU\PT_PAIVITTAISTUOTANTO\PT_Testikansio_VIZ\ModularData\"
& getFormattedDate & ".xml"
    Set fsT = CreateObject("ADODB.Stream")
```

```
        fsT.Type = 2
        fsT.Charset = "utf-8"
        fsT.Open
        fsT.WriteText DataOut
        fsT.SaveToFile OutFile, 2

        MsgBox("Data tallennettu tiedostoon: " & OutFile)
        SavedFilePath.text = OutFile

End sub
```

Data laitetaan xml-tageineen DataOut -nimiseen muuttujaan, ja sen jälkeen tämä String muuttuja tallennetaan tekstitiedostoon. Kirjoitusrutiini on itse tehty, että saataisiin oikeanlainen asemointi tageille ja tekstille. Xml-tiedosto nimetään nykyisen päiväyksen mukaan, joka saadaan funktiosta getFormattedDate().

```
'Alustetaan päiväys oikeaan formaattiin data-tiedoston nimeä varten
Function getFormattedDate ()
        Dim result, fday, fmonth, fyear, fhour, fminute, fsecond
        if Day(Now) < 10 then
                fday = "0" & Day(Now)
        else
                fday = Day(Now)
        end if

        if Month(Now) < 10 then
                fmonth = "0" & Month(Now)
        else
                fmonth = Month(Now)
        end if

        fyear = Year(Now)

        if Hour(Now) < 10 then
                fhour = "0" & Hour(Now)
        else
                fhour = Hour(Now)
        end if

        if Minute(Now) < 10 then
                fminute = "0" & Minute(Now)
        else
                fminute = Minute(Now)
        end if

        if Second(Now) < 10 then
                fsecond = "0" & Second(Now)
        else
                fsecond = Second(Now)
        end if

        result = fday & fmonth & fyear & fhour & fminute & fsecond
```

```
    getFormattedDate = result
End Function
```

7.3 Scenejen toiminta

Templaatin täyttämisen ja xml-tiedoston luomisen jälkeen on aika ajaa grafiikka ulos. Master scenessä on ohjausliitännäinen, johon välittyy xml-tiedoston sijainti tekstinä. Scenessä on itse tehty xml-tiedoston lukurutiini, joka tallentaa xml-tiedoston valmiisiin struktuureihin.

```
function loadXmlFromFile(fileName as String) as boolean
                ClearThisObject()
 Dim result as boolean
 result = false

 Dim fileResult as String

 if system.LoadTextFile(fileName, fileResult) then
                dim lines, tags as array [string]
                Dim text, resultText as String

                resultText = ""
                fileResult.trim()
                fileResult.split("\n", lines)

                for i=0 to lines.ubound
                                text=lines[i]
                                text.trim()
                                'Check what attributes the xml-file has and save them
                                if not text.find("<?xml") = -1 then
                                                Dim components as array[String]
                                                text.split(" ", components)
                                                for u=0 to components.ubound
                                                                components[u].trim()
                                                                if not components[u].find("version")
= -1 then
                                                                                this.version = compo-
nents[u].GetSubString(components[u].FindFirstOf("\"")+1,                compo-
nents[u].FindLastOf("\"")-1 - components[u].FindFirstOf("\""))
                                                                elseif  not  compo-
nents[u].find("encoding") = -1 then
                                                                                this.encoding = com-
ponents[u].GetSubString(components[u].FindFirstOf("\"")+1,              compo-
nents[u].FindLastOf("\"")-1 - components[u].FindFirstOf("\""))
                                                                end if
                                                next
                                else
                                                setNodeValues(rootNode, i, lines)
                                                exit for
```

```
                    end if
                next
                result = true
else
                Println("Could not load the Xml-file, it either does not exist or is not accessible.")
end if

loadXmlFromFile = result
end function
```

Ensin avataan tiedosto ohjelmaan ja sen jälkeen aletaan käsitellä sitä rivi riviltä. LoadXmlFromFile-funktio laukaisee sitten setNodeValues-funktion, joka tallettaa xml-noodin arvot. Funktio toimii rekursiivisesti kutsuen itseään ja käy läpi koko xml-tiedoston tallentaen sen noodit.

```
function setNodeValues(node as Node, ByRef position as Integer, lines as array[String]) as
Node
 Dim text as String
 text = lines[position]
 text.trim()

 Dim tagSubString as String
 'Since the tag is at the beginning of the string we can get the substring length by finding first
occurence of ">".
 tagSubString = text.GetSubString(text.FindFirstOf("<")+1, text.FindFirstOf(">")-1)
 println("llllllllttttttllllll" & tagSubString & "lllllllllllllllll")
 tagSubString.trim()

 'If tag contains "=" it has atleast 1 attribute
 if not tagSubString.find("=") = -1 then
                Dim components as array[String]
                node.name = tagSubString.GetSubString(0, tagSubString.FindFirstOf(" "))
                tagSubString.split(" ", components)
                for u=0 to components.ubound
                                components[u].trim()
                                if not components[u].Find("=") = -1 then
                                                Dim attribute as Attribute
                                                attribute.name = components[u].GetSubString(0,
components[u].FindFirstOf("="))
                                                attribute.value = compo-
nents[u].GetSubString(components[u].FindFirstOf("\"")+1, compo-
nents[u].FindLastOf("\"")-1 - components[u].FindFirstOf("\""))
                                                node.attributes.push(attribute)
                                end if
                next
 elseif not tagSubString.find("/") = -1 then
                node.name = tagSubString.GetSubString(0, tagSubString.FindFirstOf("/"))
 else
                node.name = tagSubString
 end if
 println("llllnnnnnnnllllll" & node.name & "lllllllllllllllll")
 println("--" & text & "--")
```

```
 Dim subText as String
 subText = text.Right(text.length-tagSubString.length-2)
 if subText.Find("<") = -1 then
                node.text = subText
 else
                node.text = subText.GetSubString(0, subText.FindFirstOf("<"))
 end if
 Dim pos1, pos2 as Integer
 pos1 = text.Find("</" & node.name & ">")
 pos2 = text.Find("<" & node.name & "/>")
 println("Ehto: " & CStr(pos1) & " ::: " & CStr(pos2) & " ::::::::")
 Dim test = true

 Do While text.Find("</" & node.name & ">") = -1 and text.Find("<" & node.name & "/>") = -1
and not text = "</" & node.name & ">"
                if test then
                                position += 1
                                test = false
                                text = lines[position]
                end if



                if text.Find("<") = -1 then
                                if text = "" then
                                                node.text &= "\n"
                                else
                                                node.text &= "\n" & text
                                end if
                else
                                Dim childNode as Node
                                node.childnodes.push(setNodeValues(childNode, position, lines))
                end if
                println("Looppi tapahtuuu---------------------->>>>>>")



                position += 1
                text = lines[position]

                println("Teksti silmukan lopussa: " & text)
                println("Tekstiin verrattava noodin nimi: " & node.name)
 Loop

 if text.FindFirstOf("<") > 0 then
                                node.text &= "\n" & text.GetSubString(0, text.FindFirstOf("<"))
 end if
 println("EEXXIIITTTTT")
 setNodeValues = node
end function
```

Lukurutiinin heikkous on se, että se vaatii xml-tiedoston oikein rivitettynä. Se ei siis pysty lukemaan tiedostoja, jotka ovat tekstiä yhdessä pötkössä.

Tallennuksen jälkeen data on käytettävissä checkNode-funktion avulla. Se käy läpi xml-struktuurit ja etsii noodia nimen perusteella. Palauttaa sitten Node-tyyppisen struktuurin.

```
'Tarkistetaan noodin nimi. Jos noodin nimi on haettu nimi, niin palautetaan se. Muuten tarkistetaan noodin lapsinoodit
function checkNode(node as Node, nameToFind as String) as Node
 Dim result as Node

 if node.name = nameToFind then
                result = node
 elseif node.childNodes.size > 0 then
                for i = 0 to node.childNodes.UBound
                                result = checkNode(node.childNodes[i], nameToFind)
                                if result.name <> "" then
                                                exit for
                                end if
                next

 end if

 checkNode = result
end function
```

Dataa voi myös lukea suoraan käymällä läpi xml-olioon tallennettua struktuuritaulukkoa.

Master scene tallentaa xml-tiedoston noodit taulukkoon, jota se alkaa käydä läpi.

```
sub lataaDataMuistiin()
 Dim rootNode as Node
 Dim nimi as String
 rootNode = (Node)xmlObjekti.getRootNode()
 typeNodeArray.clear()
 grafiikanNumero = -1
 grafiikkojenMaara = 0

 for i = 0 to rootNode.childNodes.UBound
                nimi = rootNode.childNodes[i].name
                if nimi = "pylvas" or nimi = "piirakka" or nimi = "kuvat" or nimi = "graafi" or nimi = "video" or nimi = "muokattava" then
                                typeNodeArray.push(rootNode.childNodes[i])
                                grafiikkojenMaara += 1
                elseif nimi = "YOtsikko" then
                                system.map["YOtsikko"] = rootNode.childNodes[i].text
```

```
                elseif nimi = "Tausta" then
                        if rootNode.childNodes[i].text = "0" or root-
Node.childNodes[i].text = "1" then
                                scene.FindContainer("BG").active = true
                                sce-
ne.FindContainer("kuva_tai_grafiikka").GetFunctionPluginInstance("Omo").setParameterInt("v
is_con", CInt(rootNode.childNodes[i].text))
                        elseif rootNode.childNodes[i].text = "2" then
                                scene.FindContainer("BG").active = false
                        end if
                elseif nimi = "TaustaKuva" then
                        sce-
ne.FindContainer("kuva_tai_grafiikka").FindSubContainer("BG1").CreateTexture(rootNode.chi
ldNodes[i].text)
                end if
 next
 this.scene.FindContainer("Otsikko").geometry.text = system.map["YOtsikko"]

 grafiikkojenMaara -= 1

end sub
```

Jotta datan pystytäisiin välittämään eri kerroksilla olevien scenejen välillä, pitää käyttää tekniikkaa, jota kutsutaan datan jakamiseksi. Tiedot tallennetaan system.map -muuttujaan, jota muiden kerroksien (tässä tapauksessa etummaisen) scenet voivat lukea sitä.

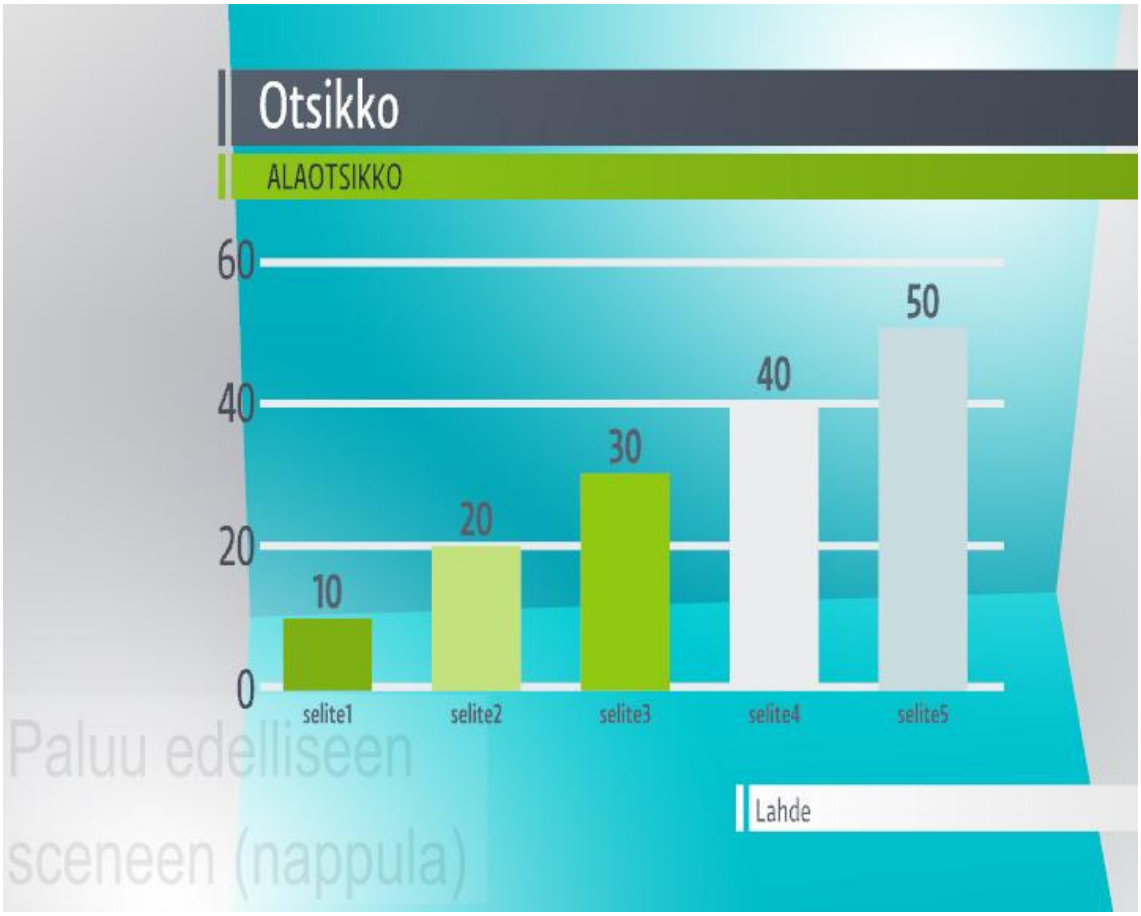
```
sub ajaGrafiikka()
 grafiikanNumero += 1

 if grafiikanNumero <= grafiikkojenMaara then
                system.map["node_used"] = typeNodeArray[grafiikanNumero]
                if typeNodeArray[grafiikanNumero].name = "pylvas" then
                        system.LoadFrontScene(scenePath & "tolpatScriptilla")
                        this.stage.FindDirector("Pylvas").show(0.1)
                        this.stage.FindDirector("Pylvas").ContinueAnimation()
                        system.FrontScene.Script.OnInit()
                elseif typeNodeArray[grafiikanNumero].name = "piirakka" then
                        system.LoadFrontScene(scenePath & "Planssipohja2WithScript")
                        this.stage.FindDirector("Piirakka").show(0.1)
                        this.stage.FindDirector("Piirakka").ContinueAnimation()
                        system.FrontScene.Script.OnInit()
                elseif typeNodeArray[grafiikanNumero].name = "kuvat" then
                        system.LoadFrontScene(scenePath & "KokoMuistiKuvat")
                        'Katsotaan monta kuvaa tässä elementissä on ja muutetaan vähän
stagea sen mukaan.
                        Dim noodi as Node
                        noodi = (Node)checkNode(typeNodeArray[grafiikanNumero],
"maara")

                        if noodi.text = "1" then
                                this.stage.FindDirector("Kuvat").show(4.6)
```

```
                        elseif noodi.text = "2" then
                                this.stage.FindDirector("Kuvat").show(4.1)
                        elseif noodi.text = "3" then
                                this.stage.FindDirector("Kuvat").show(3.6)
                        elseif noodi.text = "4" then
                                this.stage.FindDirector("Kuvat").show(3.1)
                        elseif noodi.text = "5" then
                                this.stage.FindDirector("Kuvat").show(2.6)
                        elseif noodi.text = "6" then
                                this.stage.FindDirector("Kuvat").show(2.1)
                        elseif noodi.text = "7" then
                                this.stage.FindDirector("Kuvat").show(1.6)
                        elseif noodi.text = "8" then
                                this.stage.FindDirector("Kuvat").show(1.1)
                        elseif noodi.text = "9" then
                                this.stage.FindDirector("Kuvat").show(0.6)
                        elseif noodi.text = "10" then
                                this.stage.FindDirector("Kuvat").show(0.1)

                        end if

                        this.stage.FindDirector("Kuvat").ContinueAnimation()
                        system.FrontScene.Script.OnInit()
                elseif typeNodeArray[grafiikanNumero].name = "graafi" then
                        system.LoadFrontScene(scenePath & "Graafi")
                        this.stage.FindDirector("Graafi").show(0.1)
                        this.stage.FindDirector("Graafi").ContinueAnimation()
                        system.FrontScene.Script.OnInit()
                elseif typeNodeArray[grafiikanNumero].name = "video" then
                        system.LoadFrontScene(scenePath & "VideoInput")
                        this.stage.FindDirector("Video").show(0.1)
                        this.stage.FindDirector("Video").ContinueAnimation()
                elseif typeNodeArray[grafiikanNumero].name = "muokattava" then
                        system.LoadFrontScene(scenePath & "Muokattava")
                        this.stage.FindDirector("Muokattava").show(0.1)
                        this.stage.FindDirector("Muokattava").ContinueAnimation()
                end if
else
                this.stage.findDirector("Director").stopAnimation()
                system.LoadFrontScene("")
                this.stage.findDirector("Poistuminen").show(0.2)
                this.stage.findDirector("Poistuminen").ContinueAnimation()
end if

end sub
```

AjaGrafiikka-funktio lataa grafiikkatyypin mukaisen scenen etummaiselle kerrokselle ja käynnistää tarvittavat animaatiot. Tämän jälkeen toiminta siirtyy etummaiselle kerrokselle ladatulle scenelle.

7.3.1 Pylväs-scene

Pylväs-scene näkyy seuraavassa kuvassa.



Kuva 10. Pylväsgrafiikka

Pylväs-scenen skripti purkaa datan ja asettaa sen pylväisiin. Grafiikka on siinä mielessä dynaamista, että pylväiden korkeuksia ei ole mitenkään valmiiksi asetettu, vaan skripti laskee lukuasteikon ja pylväiden korkeudet syötettyjen lukuarvojen mukaan.

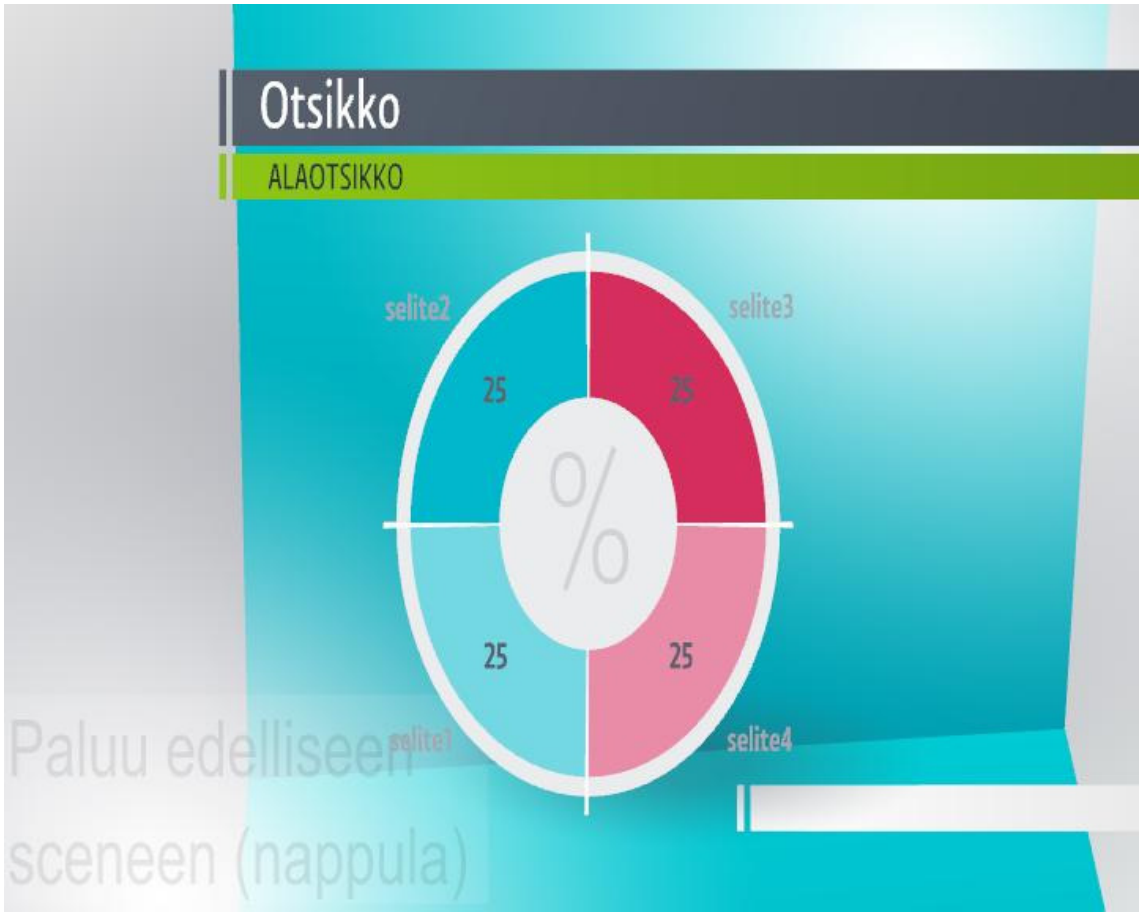
Seuraavaksi tulee silmukka, joka käsittelee pylväät yksitellen.

```
'Mennään silmukkaan, joka käsittelee kaikki pylväät
for i = 1 to maara
 'otetaan käsittelyyn yksi pylväs kerrallaan
 Dim barContainer as Container
 barContainer = barsContainer.FindSubContainer("bar" & i)
 'ensin pylvään materiaali
 noodi = (Node)checkNode(memoryNode, "P" & i & "PylvasMaterial")
 barContainer.CreateMaterial(noodi.text)
 'sitten pylvään lukuarvo ja sen materiaali
```

```
 noodi = (Node)checkNode(memoryNode, "P" & i & "LukuArvo")
 'jos pylvään lukuarvossa on pilkku pisteen sijaan, korjataan asia
 if noodi.text.FindFirstOf(",") > -1 then
        noodi.text.ReplaceChar(noodi.text.FindFirstOf(","), ".")
 end if
 barContain-
er.FindSubContainer("Bar_text").GetFunctionPluginInstance("ControlNum").setParameterStrin
g("input", noodi.text)
 'jos käytetetään desimaaleja, niin
 if noodi.text.FindFirstOf(".") > -1 then
        desimaalit = noodi.text.length - noodi.text.FindFirstOf(".")-1

 end if

 if desimaalit > 0 then
        barContain-
er.FindSubContainer("Bar_text").GetFunctionPluginInstance("ControlNum").setParameterInt("
decimal_places", desimaalit)
 else
        barContain-
er.FindSubContainer("Bar_text").GetFunctionPluginInstance("ControlNum").setParameterInt("
decimal_places", 0)
 end if
 noodi = (Node)checkNode(memoryNode, "PLukuMaterial")
 barContainer.FindSubContainer("Bar_text").CreateMaterial(noodi.text)
 'sitten pylvään korkeus
 noodi = (Node)checkNode(memoryNode, "P" & i & "Korkeus")
 barContainer.FindSubContainer("Bar_object").geometry.SetParameterDouble("height",
CDbl(noodi.text))
 'sitten pylvään seliteteksti ja sen materiaali
 noodi = (Node)checkNode(memoryNode, "P" & i & "SeliteTeksti")
 barContainer.FindSubContainer("title_text").geometry.text = noodi.text
 noodi = (Node)checkNode(memoryNode, "PSeliteMaterial")
 barContainer.FindSubContainer("title_text").CreateMaterial(noodi.text)
next
```

7.3.2 Piirakka-scene

Piirakka-scene näkyy seuraavassa kuvassa.



Kuva 11. Piirakkagrafiikka

Piirakka on tehty Artistista löytyvällä Pie-liitännäisellä. Skripti laskee annettujen lukuarvojen perusteella piirakan viipaleille oikeat leveydet ja asettaa ne liitännäiseen.

Seuraavassa esimerkissä on silmukka, joka asettaa kaikki tarvittavat arvot ja seliteteksti viipaleille.

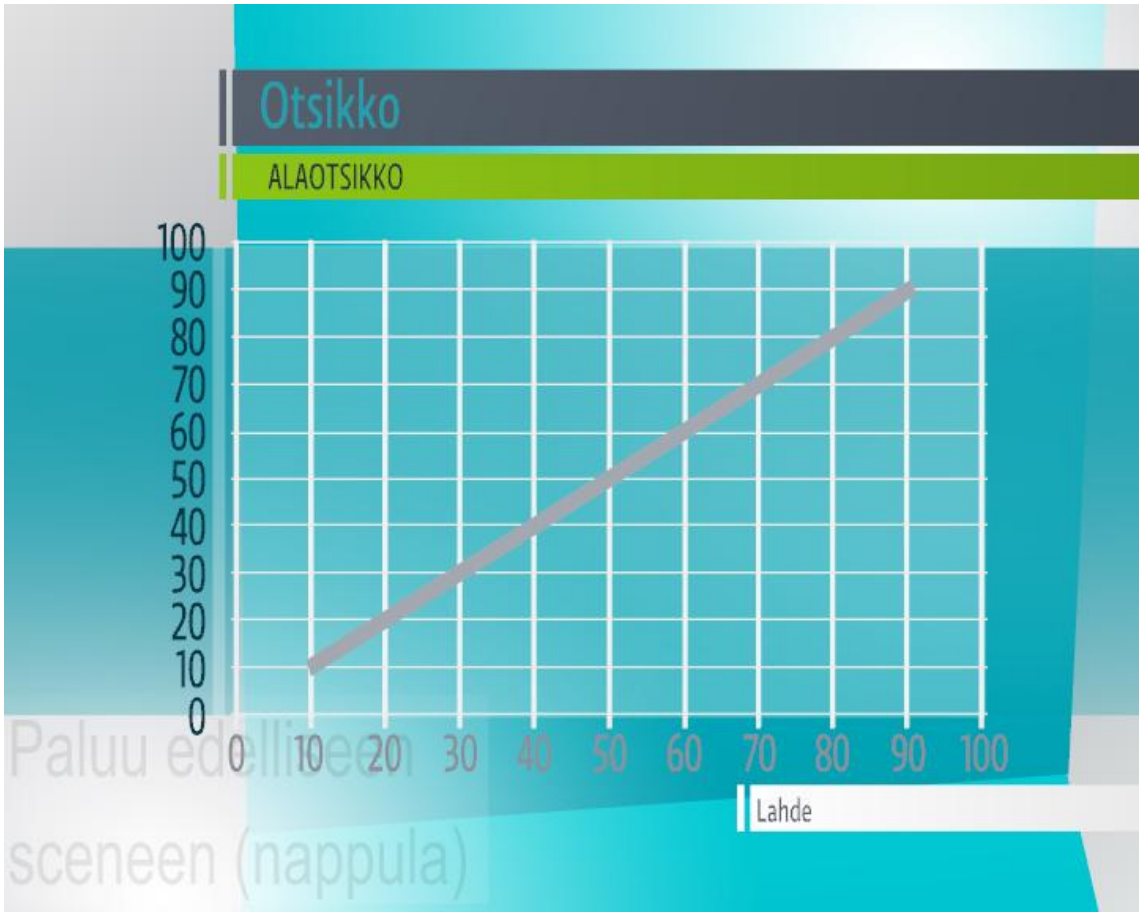
```
'mennään silmukkaan, joka käsittelee kaikki slicet
for i = 0 to maara-1

 'otetaan slice containeri käsittelyyn
 Dim sliceContainer as Container
 sliceContainer = piirakkaContainer.FindSubContainer("Slice" & i)

 'Otetaan noodi, missä on slicen lukuarvo
 noodi = (Node)checkNode(memoryNode, "Pi" & CStr(i+1) & "LukuArvo")
 println("------------nnoooodin teksti-------------")
 println(noodi.text)
 println("----------------------------------------")
 'otetaan slicejen yhteinen maksimiarvo laskuun mukaan:                    'x*x/(z*y), jossa
 'y = muunnettava lukuarvo
 'x = paljonko slicejen arvot tekevät yhteensä
```

```
                'z = asteikko johon muunnetaan
'Asetetaan slicen "paksuus"
Dim temp as String
temp = noodi.text
if noodi.text.FindFirstOf(",") <> -1 then
                temp.ReplaceChar(temp.FindFirstOf(","), ".")
end if

piirakkaContainer.GetFunctionPluginInstance("PieValues").SetParameterDouble("value"      &
CStr(i*2), (100.0/yhteensa)*CDbl(temp)-thinSlicenPaksuus)
'Asetetaan slicen materiaali
variNoodi = (Node)checkNode(memoryNode, "Pi" & CStr(i+1) & "Material")
sliceContainer.CreateMaterial(variNoodi.text)
'Asetetaan lukuarvoteksti slicen keskelle
sliceContainer.FindSubContainer("LUKU0").geometry.text = CStr(noodi.text)
'Asetetaan lukujen materiaali
variNoodi = (Node)checkNode(memoryNode, "LukuMaterial")
sliceContainer.FindSubContainer("LUKU0").CreateMaterial(variNoodi.text)

'sliceContainer.FindSubContainer("LUKU0").geometry.text = "OLEN LUKU"

if piirakkaContainerString = "PIE_10" then
                'Asetetetaan selitetekstin sisältö
                noodi = (Node)checkNode(memoryNode, "Pi" & CStr(i+1) & "Selite")
                sliceContainer.FindSubContainer("Lukuarvo").geometry.text = Cstr(noodi.text)
                'Asetetaan selitetekstin materiaali
                variNoodi = (Node)checkNode(memoryNode, "SeliteMaterial")
                sliceContainer.FindSubContainer("Lukuarvo").CreateMaterial(variNoodi.text)
else
                'Skaalaväli on 0.1 - 0.8
                'Offsetväli on 58.7 - 71.4
                Dim doubleTemp, oldMax, oldMin, newMax, newMin, a, b as Double
                oldMax = 0.8
                oldMin = 0.1
                newMax = 75
                newMin = 58.7

                Dim intTemp as Integer
                doubleTemp                              =                              piirakkaContain-
er.GetFunctionPluginInstance("PieValues").GetParameterDouble("value" & CStr(i*2))

                doubleTemp = doubleTemp/100

                println("----------------Doubletemp----------")
                println(doubleTemp)
                println("--------------------------------------")
                if doubleTemp < 0.1 then
                                doubleTemp = 0.1
                elseif doubleTemp > 0.8 then
                                doubleTemp = 0.8
                end if



Dim vertexi as Vertex
```

```
                vertexi.x = doubleTemp
 vertexi.y = doubleTemp
 vertexi.z = doubleTemp
 sliceContainer.FindSubContainer("LUKU0").Scaling.xyz = vertexi

                println(sliceContainer.FindSubContainer("LUKU0").Scaling.x)
                println(sliceContainer.FindSubContainer("LUKU0").Scaling.y)
                println(sliceContainer.FindSubContainer("LUKU0").Scaling.z)

'Tehdään asteikonmuunnos skaalan lukuarvolle, jotta voidaan määrittää piirakkatekstin oikea offset
                a = (newMax-newMin)/(oldMax-oldMin)
                b = newMax - a*oldMax
                doubleTemp = a*doubleTemp + b

                sliceContain-
er.GetFunctionPluginInstance("PieSlice").SetParameterDouble("TextOfftse", doubleTemp)
 end if
 println("------------nnoooodin teksti-------------")
 println(noodi.text)
 println("----------------------------------------")
 'sliceContainer.FindSubContainer("Lukuarvo").geometry.text = "OLEN SELITE"



 next
```

7.3.3 Käyrä-scene

Seuraavassa kuvassa näkyy käyrä-scene.



Kuva 12. Käyrägrafiikka

Itse käyrän piirtämiseen on käytetty valmista liitännäistä, johon voi syöttää maksimissaan 50 x- ja y-koordinaattia. Toisin kuin pylväsgrafiikassa lukuasteikko syötetään sapluunasta käsin.

```
'Mennään silmukkaan, joka käsittelee kaikki graafin pisteet

for i = 1 to maara
 Dim x, y as Double
 'Ensin haetaan x:n ja y:n arvot ja muutetaan ne oikeaan asteikkoon.
 noodi = (Node)checkNode(memoryNode, "GX" & i)
 if noodi.text.Find(",") <> -1 then
                noodi.text.replaceChar(noodi.text.Find(","), ".")
 end if
 x = asteikkoMuunnos(xMax, 0, xAsteikkoMax, xAsteikkoMin, CDbl(noodi.text))
 noodi = (Node)checkNode(memoryNode, "GY" & i)
 if noodi.text.Find(",") <> -1 then
                noodi.text.replaceChar(noodi.text.Find(","), ".")
 end if
 y = asteikkoMuunnos(yMax, 0, yAsteikkoMax, yAsteikkoMin, CDbl(noodi.text))

 'Sitten asetetaan arvot graafiin.
 graafiPlugari.SetParameterDouble("X" & i-1, x)
```

```
 graafiPlugari.SetParameterDouble("Y" & i-1, y)

next
```

Vaikka lukuasteikko syötetäänkin käsin, sen korkeus ja leveys täytyy laskea skriptillä. Seuraavassa esimerkissä lukuasteikon asettavat funktiot asteikonYArvot ja asteikon-XArvot.

```
sub asteikonYArvot()
 Dim noodi as Node
 'Otetaan y-asteikon container käsittelyyn
 Dim cont as Container
 cont = scene.findContainer("asteikkonumerotY")


 noodi = (Node)checkNode(memoryNode, "pystyMaara")
 yMaara = CInt(noodi.text)

 yKork = -yAMax/(Double)(yMaara-1)
 cont.GetFunctionPluginInstance("Grid").SetParameterInt("num_row", yMaara)
 cont.GetFunctionPluginInstance("Grid").SetParameterDouble("row_offset", yKork)

 'Asetetaan y-asteille käyttäjän syöttämät arvot
 for i = 1 to yMaara
                noodi = (Node)checkNode(memoryNode, "GPysty" & i)
                if noodi.text.FindFirstOf(",") > -1 then
                                noodi.text.ReplaceChar(noodi.text.FindFirstOf(","), ".")
                                cont.findSubContainer("asteikko"                                &
i).GetFunctionPluginInstance("ControlNum").setParameterInt("decimal_places",
noodi.text.Length - noodi.text.FindFirstOf(".")-1)
                                cont.findSubContainer("asteikko"                                &
i).GetFunctionPluginInstance("ControlNum").setParameterString("input", noodi.text)
                else
                                cont.findSubContainer("asteikko"                                &
i).GetFunctionPluginInstance("ControlNum").setParameterInt("decimal_places", 0)
                                cont.findSubContainer("asteikko"                                &
i).GetFunctionPluginInstance("ControlNum").setParameterString("input", noodi.text)
                end if
 next

 'Pistetään y-asteikon minimi- ja maksimiarvot muistiin
 noodi = (Node)checkNode(memoryNode, "GPysty1")
 yAsteikkoMin = CDbl(noodi.text)
 noodi = (Node)checkNode(memoryNode, "GPysty" & yMaara)
 yAsteikkoMax = CDbl(noodi.text)
end sub

sub asteikonXArvot()
 Dim noodi as Node
 'Otetaan x-asteikon container käsittelyyn
 Dim cont as Container
 cont = scene.findContainer("asteikkonumerotX")
```

```
'Asetetaan vaakarivien määrä ja välit
noodi = (Node)checkNode(memoryNode, "vaakaMaara")
xMaara = CInt(noodi.text)
'xPit = xPit*(Double)xMaara
xPit = xAMax/(Double)(xMaara-1)
cont.GetFunctionPluginInstance("Grid").SetParameterInt("num_col", xMaara)
cont.GetFunctionPluginInstance("Grid").SetParameterDouble("col_offset", xPit)

noodi = (Node)checkNode(memoryNode, "tekstiRuksi")

if noodi.text = "1" then
		for i = 1 to xMaara
			noodi = (Node)checkNode(memoryNode, "GVaaka" & i & "Text")
			cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").active = false
			cont.findSubContainer("asteikko" & i).FindSubContainer("numero").geometry.text = noodi.text
		next
else
		for i = 1 to xMaara
			noodi = (Node)checkNode(memoryNode, "GVaaka" & i)
			cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").active = true
			if noodi.text.FindFirstOf(",") > -1 then
				noodi.text.ReplaceChar(noodi.text.FindFirstOf(","), ".")
				cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").setParameterInt("decimal_places", noodi.text.Length - noodi.text.FindFirstOf(".")-1)
				cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").setParameterString("input", noodi.text)
			else
				cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").setParameterInt("decimal_places", 0)
				cont.findSubContainer("asteikko" & i).FindSubContainer("numero").GetFunctionPluginInstance("ControlNum").setParameterString("input", noodi.text)

			end if
		next

end if
'Asetetaan x-asteikkolle käyttäjän syöttämät arvot


'Pistetään x-asteikon minimi- ja maksimiarvot muistiin
noodi = (Node)checkNode(memoryNode, "GVaaka1")
xAsteikkoMin = CDbl(noodi.text)
noodi = (Node)checkNode(memoryNode, "GVaaka" & xMaara)
xAsteikkoMax = CDbl(noodi.text)
```

```
end sub
```

8 Loppusanat

Työssä käytiin läpi Vizrt-järjestelmän toimintaa graafikon ja ohjelmoijan näkökulmasta. Lisäksi tutustuttiin kosketusnäyttögrafiikan tekemiseen ja käytiin läpi modulaarinen malli koodiesimerkein.

Modulaarisen mallin pitäisi palvella koko ajan kehittyviä tarpeita uutisgrafiikan toimituksessa. Työ oli paikoittain haastavaa, mutta myös palkitsevaa. Olen siinä käsityksessä, että näin monimutkaista templaatin ja scenen yhdistelmää ei ole ainakaan Suomessa aikaisemmin tehty. Toivon, että työstä on apua, niin grafiikan suunnittelussa, kuin ohjelmointityön ymmärtämisessäkin.

Lähteet

1 Viz Content Pilot User's Guide. 2009. Ohjekirja. Vizrt. http://documentation.vizrt.com/viz-content-pilot-5.3.html. Luettu 20.03.2013.

2 Viz Template Wizard User's Guide. 2008. Ohjekirja. Vizrt. http://documentation.vizrt.com/viz-content-pilot-5.3.html. Luettu 20.03.2013.

3 Viz Engine Administrator's Guide. 2011. Ohjekirja. Vizrt. http://documentation.vizrt.com/viz-engine-3.3.html. Luettu 20.03.2013.

4 Viz Artist User's Guide. 2010. Ohjekirja. Vizrt. http://documentation.vizrt.com/viz-artist-3.3.html. Luettu 20.03.2013.