# VAAKAOHJAIMEN SOVELLUSKEHITTIMEN SUUNNITTELU JA TOTEUTUS

LAHDEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2006
Kari-Antti Kuosa

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

KUOSA, KARI-ANTTI: Vaakaohjaimen sovelluskehittimen suunnittelu ja toteutus

Ohjelmistotekniikan opinnäytetyö, 63 sivua

Kevät 2006

## TIIVISTELMÄ

Tämä opinnäytetyö käsittelee vaakaohjaimen sovelluskehittimen käytännön suunnittelua, toteutusta ja esittelee työssä eteen tulleita ongelmia ja niiden ratkaisuja sekä käytettyjä tekniikoita. Työssä vastataan myös kysymykseen, kuinka XML:ää ja Javan grafiikkakirjastoja voidaan hyödyntää toteutettaessa vaakaohjaimen sovelluskehitintä.

Sovelluskehitintä käytetään aputyökaluna Raute Precision Oy:n toimittamien asiakasprojektien ja niiden sisältämien vaakaohjainlaitteiden tuottamiseen. Kehittimellä asetetaan projektin yleiset asetukset sekä määritellään laitteiden rekisterialueet sekä niiden käyttöliittymien ulkoasu ja toiminta.

Sovelluskehittimen käyttämät mallit ja määritykset sekä sillä tuotetut projekti- ja laitemääritykset tallennetaan XML-rakenteeseen. Näiden rakenteiden lukemiseen, tuottamiseen ja hallinnointiin käytetään JDOM-ohjelmistorajapintaa. Kehitinohjelma toteutettiin kokonaisuudessaan Java-kielellä. Käyttöliittymien tuottamiseen valittiin Swing-grafiikkakirjasto sen MVC-mallin mukaisen rakenteen ja kattavan komponenttivalikoiman ansiosta.

Sovelluskehitintyön pääpaino ja ongelmat olivat tietojen varastoinnin ja hallinnan toteuttamisessa sekä erilaisten käyttöliittymäkomponenttien yhteiskäyttö XML:n ja JDOM-rakenteen kanssa. Näiden ongelmien ratkaisemiseksi työssä toteutettiin, normaalien Swing-komponenttien lisäksi, erikoistettuja käyttöliittymäelementtejä, joiden päätehtävänä oli XML-rakenteen kuvaaminen graafisesti ja luoda käyttöliittymät rakenteiden muokkaamiseen.

Sovelluskehitin rakentuu pääohjelmasta sekä siinä käytettävistä hallintamoduuleista. Moduulit ovat toisistaan riippumattomia ohjelmanosia, ja ne ovat erikoistuneet kukin tiettyä tehtävää varten. Rakenne antaa mahdollisuuden uusien osien nopealle toteuttamiselle sekä helpottaa useiden ohjelmoijien samanaikaisesti tapahtuvan ohjelmistokehityksen.

Sovelluskehittimen käytännön toteutuksessa päästiin asetettuihin tavoitteisiin ja kaikki asetetut vaatimukset täytettiin. Työssä toteutettu vaakaohjaimen sovelluskehitin on otettu käyttöön Raute Precision Oy:n tuotekehitysosastolla, ja sen jatkokehitys on täydessä käynnissä.

Asiasanat: Java, XML, Suunnittelumallit, Swing, JDOM, ohjelmistorajapinnat

Lahti University of Applied Sciences
Faculty of Technology

KUOSA, KARI-ANTTI: Designing and implementing an application development environment for a weighing controller

Bachelor's thesis in Software Engineering, 63 pages

Spring 2006

ABSTRACT

This study deals with how to design and implement an application development environment (ADE) for a weighing controller. The study answers the question how to utilize XML structures and Java's graphic libraries to the ADE.

The purpose of the ADE is to help produce weighing controllers for the customer project of Raute Precision Oy. It makes it easier to produce the project's genral settings and the controllers' register areas. It is also possible to design the graphical user interfaces and functions of the controllers with the ADE.

The models and definitions used by the ADE and project and controller configurations developed by it are stored in XML structures. The structures are read, designed and controlled by the JDOM application programming interface. The application was produced with the Java language and Swing graphic library.

The development work focused on storing and controlling XML data. Graphical user interface components, which are used with the JDOM structure, caused extra challenge and problems. These user interface components were generalized by Swing components. Specialized components took care of displaying the XML data and creating a user interface for modifying structures.

The application is composed of the main program and the control modules. The modules are self-contained and they all have their own function. The composition makes it possible to design new modules faster and easier.

The practical implementation of the ADE met all the requirements that had been set. The application is now in use in the development unit at Raute Precision.

Keywords: Java, XML, Design Patterns, Swing, JDOM, application programming interface

TEKIJÄN KIITOKSET

Tämän oppinäytetyön toteutuksen mahdollisti Raute Precision Oy, jonka palveluksessa sain työskennellä opiskelun ohessa niin kesällä kuin talvellakin. Haluan kiittää yritystä ja koko tuotekehitysyksikön henkilöstöä mukavasta, opettavaisesta ja lämminhenkisestä työympäristöstä.

Lisäksi haluan kiittää työni ohjaajaa Matti Weliniä joustavuudesta ja kärsivällisyydestä työskentelyäni ja aikataulujani kohtaan. Tuesta ja kannustuksesta sekä yhteisistä illoista ja öistä työn parissa kiitän opiskelutoveriani ja ystävääni Pasi Widgreniä.

Suuret kiitokset haluan antaa myös Helilleni, jonka myötävaikutuksella ja tuella olen tämän työni saanut viimein valmiiksi. Kiitos myös siitä, ettet kertaakaan patistellut minua työni ääreen.

Lopuksi haluan omistaa opinnäytetyöni pojalleni Pyrylle, jonka syntymä antoi uutta energiaa ja voimia saattaa loppuun kirjallinen osuus työstäni. Kiitos aurinkoisuudestasi ja iloisuudestasi ja siitä, että olet olemassa.

SISÄLLYS

LYHENNELUETTELO

| | |
|---|---|
| XML | Extensible Markup Language |
| W3C | World Wide Web Consortium – Kansainvälinen yritysten ja yhteisöjen yhteenliittymä. Ylläpitää ja kehittää standardeja ja suosituksia. |
| SAX | Simple API for XML – XML:n käsittelyssä käytetty ohjelmistorajapinta, joka perustuu tapahtumapohjaisuuteen. |
| DOM | Dokument Object Model – XML:n käsittelyssä käytetty ohjelmistorajapinta, joka perustuu puurakenteeseen. |
| API | Application programming interface – Ohjelmointirajapinta, jota käyttämällä eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään. |
| GUI | Graphical User Interface – Graafinen käyttöliittymä. |
| XSL | Extensible stylesheet – XML:n muotoilussa käytetty kieli. |
| XSLT | Extensible stylesheet transformations – XML-pohjainen kuvauskieli XML-tiedostojen muuntamiseen toiseen muotoon. |
| JDK | Java Development Kit – Sun Microsystems:in kehittämö Javan kehitysympäristö. |
| MVC | Model-View-Controller architecture – Korkean tason suunnittelumalli, joka määrittelee ohjelman päätason rakenteen. |
| UI | User interface – Käyttöliittymä. |
| JDOM | Java-based "document object model" for XML files – Javalla XML:n käsittelyssä käytetty ohjelmistorajapinta, joka perustuu puurakenteeseen. |

# 1 JOHDANTO

Raute Precision Oy:n toiminta muodostuu useasta punnitusteknologiaa hyödyntävästä liiketoiminta-alueesta. Raute Precision toimittaa punnitus- ja annostusjärjestelmiä ja laitoksia, vaakoja ja muita punnituskomponentteja sekä kunnossapito ja kalibrointipalveluja eri teollisuudenaloille, kaupalle ja julkiselle sektorille. Yritys on lasiteollisuuden raaka-ainelaitosten sekä laasti- ja tasoitetehtaiden toimittajana toinen alan johtavista yrityksistä maailmassa.

Raute Precision Oy:llä on vuosikymmenien kokemus punnitus- ja annostusprosessien toiminnasta, sillä ensimmäisten vaakojen valmistus aloitettiin jo vuonna 1914. Yrityksen henkilöstömäärä on noin 230, ja sen toimipisteet sijaitsevat Lahdessa, Espoossa, Jyväskylässä, Paraisilla sekä Shanghaissa Kiinassa. Raute Precisionin avainkomponentit ja tuotteet valmistetaan omassa konepaja-elektroniikkayksikössä Lahdessa.

Opinnäytetyössä toteutettava vaakaohjaimen sovelluskehitin on osa laajempaa Raute Precisionin tuotekehitysyksikössä käynnissä olevaa kehitysprojektia, jolla pyritään saamaan kilpailuetua kovenevilla punnitusalan markkinoilla.

Toteutettavalla sovelluskehittimellä tarkoitetaan ohjelmaa, jota hyödyntämällä toteutetaan ennalta määriteltyjen rajoitteiden puitteissa uusia vaakaohjain variaatioita, niiden käyttöliittymiä ja kehitetään ja ylläpidetään jo olemassa olevia. Perinteisemmästä sovelluskehityksestä toteutustapa eroaa siinä, ettei kehittimen käyttöön tarvita ohjelmointitaitoa tai yksityiskohtaista tietoa toteutettavassa sovelluksessa käytetyistä teknisistä ratkaisuista. Sovelluskehittimen yksi tärkeimmistä ominaisuuksista onkin, että se mahdollistaa laajemman ohjelmistokehittäjien, käyttöönottajien ja ylläpitäjien joukon. Tähän joukkoon voi kuulua henkilöitä, joilla ei ennen ole ollut mahdollisuutta osallistua henkilökohtaisesti sovelluksen toteutukseen.

Sovelluskehitintä toteutettaessa on tärkeää määritellä sillä toteutettavien sovellusten luonne ja ominaisuudet sekä itse kehittimessä olevat toiminnallisuudet, jotta on mahdollista luoda helppokäyttöinen työskentely-ympäristö. Opinnäytetyön

tavoitteena on selvittää teknisiä malleja ja keinoja sovelluskehittimen toteutukselle, joka mahdollistaa:

- ohjainlaitteen graafisien käyttöliittymien toteutuksen
- ohjainlaitteen rekisterialueiden suunnittelun
- eri laitevariaatioiden tuottamisen rekisterialueiden pohjalta
- olemassa olevien ohjainlaitteiden asetusten ja tilojen ylläpidon.

Samalla pyritään vastaamaan kysymykseen, kuinka XML:ää ja Javan grafiikkakirjastoja voidaan hyödyntää toteutettaessa vaakaohjaimen sovelluskehitintä?

Opinnäytetyö koostuu neljästä osa-alueesta. Ensimmäisessä osiossa käsitellään opinnäytetyön käynnistämiseen vaikuttaneita seikkoja, työn tavoitteita ja rajauksia. Toisessa osassa esitellään käytettyjä tekniikoita, kuten Javan käyttöliittymäkomponentteja, XML-kieltä ja siihen liittyviä ohjelmistorajapintoja sekä yleisesti ohjelmistotuotantoon liittyviä suunnittelumalleja. Kolmannessa osassa käydään läpi sovelluskehittimen suunnittelua, toteutusta ja teoriaosassa esiteltyjen tekniikoiden soveltamista käytännön työssä. Viimeiseksi tarkastellaan opinnäytetyön onnistumista ja esitetään yleinen arvio ohjelmiston nykytilasta ja sen jatkokehittämisestä ja tulevaisuudesta.

# 2 TYÖN TAUSTAT

## 2.1 Yleistä

Tarve sovelluskehittimelle syntyi, kun sillä toteutettavan ohjainlaitteen ja sen käyttöliittymän ominaisuuksia oli suunniteltu ja toteutettu riittävän pitkälle. Tuolloin huomattiin, että asetettuihin tavoitteisiin päästäisiin ainoastaan jonkinlaisella sovelluskehittimellä. Oman sovelluskehittimen toteutus ei ollut itsestään selvää, vaan ennen sen suunnittelun käynnistämistä selvitettiin olemassa olevia valmiita kehittimiä ja niiden tarjoamia ominaisuuksia.

Ajan kuluessa ohjainlaitteen teknisten vaatimusten lisääntyminen johti tilanteeseen, jossa päädyttiin oman sovelluskehittimen suunnittelun ja toteutuksen käynnistämiseen. Ratkaisuun vaikuttaneita tärkeimpiä ohjainlaitteelle asetettuja vaatimuksia olivat käyttöliittymien räätälöintimahdollisuus, erilaisten tuote varianttien nopea toteutus, alustariippumattomuus sekä ohjainlaitteen alustamisen helpottaminen ja automatisointi.

## 2.2 Kehittämisen tarve ja työn tavoitteet

Verrattaessa vanhempaa ohjainlaitesukupolvea kehitteillä olevaan uuteen suurimpia eroja ovat samat tekniset ominaisuudet, jotka vaikuttivat oman sovelluskehittimen toteutuksen käynnistämiseen. Taustalla on uudentyyppinen tuoteperhe ajattelu, jossa samalle tekniselle alustalle on mahdollista toteuttaa nopealla tahdilla erilaisia tuotevariantteja. Ajattelumalliin ovat johtaneet monet asiakasprojektit, joita on jouduttu räätälöimään asiakkaan vaatimusten ja toiveiden perusteella. Vanhan ohjainlaitetoteutuksen kanssa tällainen räätälöinti on aina tarkoittanut uuden variaation ohjelmallista kääntämistä laitteelle.

Useiden erilaisten asiakasprojektien toteutukseen ja räätälöimiseen kuluva aika on pitkä, kun jokainen uusi versio täytyy erikseen kääntää laitteelle. Sovelluskehitin nopeuttaa tuotantoprosessia, ja se myös antaa mahdollisuuden siirtää asiakkaalle

suunniteltavien tuotteiden määritystä ja suunnittelua ajallisesti eteenpäin. Tämä antaa projektin suunnittelijoille mahdollisuuden työstää ohjainlaitteen käyttöliittymiä ja sen ominaisuuksia huomattavasti laajemmalla ajanjaksolla kuin aiemmin. Mahdollisuus siirtää ohjainlaitteen käyttöliittymien ja ominaisuuksien toteutusta, jopa käyttöönottotilanteeseen, helpottaa suunnittelijoiden työtä, kun kaikkia mahdollisia tapauksia ei tarvitse ennakolta suunnitella.

## 2.3 Työn rajaukset

Tässä tutkimuksessa käsitellään ohjainlaitteen sovelluskehittimen suunnittelua ja toteutusta. Ennen sovelluskehittimen suunnittelun ja toteutuksen aloittamista itse ohjainlaiteen ja sen käyttöliittymäohjelman toteutus oli jo käynnistynyt. Työssä viitataan ohjainlaitteeseen ja siinä käytettävään käyttöliittymäohjelmaan, mutta niiden toteutukseen ja teknisiin yksityiskohtiin ei syvennytä enempää kuin on tarpeellista, jotta ymmärretään näiden yhteys itse sovelluskehittimeen.

Työssä käytettävästä XML-kielestä esitellään ainoastaan sen perusteet ja yleinen rakenne. Kielen perusteisiin lukeutuvia tekniikoita, kuten DTD-rajoitukset, schema-tekniikka, transformaatio (XSL, XSLT) tai muita vastaavia menetelmiä ei esitellä.

Tutkimuksessa oletetaan, että lukijalla on tuntemusta Java-kielestä, ja itse kieltä ei esitellä muuten kuin hyvin yleisellä tasolla luvussa Javan graafisen käyttöliittymän ohjelmointi. Graafisia käyttöliittymiä käsittelevässä luvussa käydään läpi Javan käyttöliittymien kehitystä ja esitellään tarkemmin Swing-grafiikkakirjastoa. Kirjastosta esitellään sen sisällä olevien komponenttien perusrakenne sekä niiden tapahtumien käsittely ja tietojen hajauttaminen.

# 3 XML-KIELI

## 3.1 Yleistä

XML:n kehitys alkoi vuonna 1996 ja se on ollut W3C-suositus vuoden 1998 helmikuusta lähtien. XML:n edeltäjänä voidaan pitää SGML-kieltä, joka kehitettiin 80-luvun alussa ja se on ollut ISO-standardi jo vuodesta 1986. XML:n suunnittelussa on otettu mallia SGML:n parhaista puolista, ja siinä on hyödynnetty myös HTML:stä saatuja kokemuksia.

XML-kieli keskittyy pääasiassa määrittelemään dokumentin rakenteen; elementtien täytyy alkaa, ja loppua ja jokaisella attribuutilla täytyy olla vain yksi arvo. XML määrittelee vain vähän dokumentin sisällöstä, joka riippuu pitkälti toteuttajan tarpeista. Jokainen voi kehittää oman dokumenttimuodon ja määritellä, kuinka data esitetään. Tämä mahdollistaa myös yhteiskäytettävyyden eri osapuolien välillä. Osapuolet tai yhteisöt voivat sopia rajoituksista ja säännöistä, joiden puitteissa tietoa siirretään eri järjestelmien välillä. Erilaisista yhteisesti sovituista säännöistä voi ajan kuluessa tulla yleinen standardi, joka helpottaa tietojen siirtoa. (McLaugh, 2001, 21.)

XML-kielen ollessa vain yksinkertaista tekstiä sitä voi siirtää mistä tahansa käyttöjärjestelmästä toiseen tai käyttää millaisessa ympäristössä tahansa. XML-kieli noudattaa W3C:n luomia määrityksiä ja on näin yhteensopiva kaikkialla riippumatta järjestelmästä. Kaikkien noudattaessa samoja määrityksiä XML-kielestä on tullut standardi, jota jokainen sen määrityksiä käyttävä ohjelma tai ihminen ymmärtää. Aina XML-kieltä siirrettäessä voivat siirtäjä ja vastaanottaja olla varmoja siitä, että ne ymmärtävät toimitettavaa dataa. Yksi XML-kielen:n vahvuuksia onkin juuri sen siirrettävyys. (McLaugh, 2001, 20.)

## 3.2 XML-kielen perusteita

XML-dokumentin voi jakaa karkeasti kahtia: otsikkoon ja sisältöön. Otsikossa on tiedot dokumentin käsittelylle ja niitä hyödyntävät mm. XML-jäsentimet (Parser) ja XML-sovellukset. Sisältöosa koostuu varsinaisesta XML-datasta.

Otsikko on yksinkertainen XML-esittely, jossa voi olla tietoja käytettävän XML-kielen:n versiosta, käytetyn merkistön lyhenne sekä tieto siitä onko käsiteltävä dokumentti itsenäinen vai tarvitaanko sen käsittelyyn ja ymmärtämiseen muita dokumentteja (kuvio 1 rivi 4). Otsikkoon voidaan myös liittää XML-dokumentin rakenteen kuvaava dokumenttityyppi (kuvio 1 rivi 1), jonka sisällöstä selviää dokumentissa käytetyt määritykset. Dokumenttityyppisäänöissä määritellään myös mitä elementtejä ja attribuutteja dokumentissa saa olla sekä niiden sijainti toisiinsa nähden. Dokumenttia, joka on oikeinmuodostettu ja noudattaa kaikkia määrityksiä sanotaan validiksi. Otsikossa voidaan määritellä, kuinka dokumentti näytetään katselijalle. Tällaista osaa kutsutaan XML-stylesheet osaksi. (McLaugh, 2001, 31.)

```
1. <!DOCTYPE html PUBLIC "-//WC#//DTD XHTML 1.0 Transitional//EN"
2. http://www.w3c.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>
3.
4. <?xml version="1.0"? encoding="UTF-8" standalone="no">
```

KUVIO 1. XML dokumentin otsikko

XML-dokumentin otsikon jälkeinen sisältöosa koostuu elementeistä, attribuuteista ja itse datasta. Sisällön aloittaa juurielementti, joka on korkeimman tason elementti XML-dokumentissa. Juurielementin täytyy olla dokumentin ensimmäinen avaava tagi ja viimeinen sulkeva tagi, sillä sen avulla XML-jäsennin tai XML:ää muuten käyttävä ohjelma tunnistaa dokumentin alun ja lopun. (McLaugh, 2001, 31.)

Elementit ovat XML-dokumentin perusta. Elementille annetaan nimi ja sen ympärille asetetaan kulmasulkeet. Elementin nimen ensimmäinen merkki tulee olla

kirjain tai alaviiva, ja sen jälkeen voi olla mielivaltainen määrä kirjaimia, numeroita, alaviivoja, yhdysviivoja tai pisteitä. Nimen sisällä ei saa olla välilyöntejä.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <register_block name="Gateway config" >
3.   <register_block name="" />
4.
5.   <register_block name="Serial config" >
6.     <register_block name="Serial_port_1">
7.       <register name="serial1_baud" value="38400" />
8.       <register name="serial1_mode" value="0" >
9.         <option name="RS232">0</option>
10.       </register>
11.       <register name="serial1_flow_ctrl" value="0"
12.         <option name="On" >1</option>
13.         <option name="Off" >0</option>
14.       </register>
15.     </register_block>
16. </register_block>
```

KUVIO 2. XML-dokumentti

Elementti voidaan sulkea kahdella eri tavalla. Sulkeminen tapahtuu aina lopetustagilla, joka koostuu elementin nimestä ja vinoviivasta sen edessä, kun elementti sisältää dataa. Elementti voidaan lopettaa myös aloitustagin yhteydessä lisäämällä sen loppuun vinoviiva (kuvio 2, rivi 3). Yksinäistä elementtiä kannattaa käyttää tilanteissa, joissa se selventää dokumentin ymmärrettävyyttä ja selkeyttää rakennetta. Isoissa dokumenteissa, joissa on paljon yksinäisiä elementtejä, esitelty sulkumekanismi voi vähentää dokumentin kokoa. (McLaugh, 2001, 33–35.)

Elementin sisällä olevan datan lisäksi elementillä voi olla attribuutteja. Attribuutit määritellään elementin nimen jälkeen (kuvio 2, rivi 7), ja ne erotetaan toisistaan välilyönnillä. Attribuuttien nimet noudattavat samoja sääntöjä elementtien nimien kanssa, eikä niiden lukumäärää ole rajoitettu. Attribuutteja on hyvä käyttää tilanteissa, joissa datalla on vain yksi arvo. Jos data voi saada useita arvoja tai sitä on hyvin paljon, on perusteltua käyttää elementtiä. Attribuuteilla voidaan myös määritellä elementtiä tarkemmin ja käyttää niitä hyväksi yhdessä sovelluslogiikan kanssa. (McLaugh, 2001, 35–36.)

Ennalta määritelly entiteetit

```
&  &amp;
<  &lt;
>  &gt;
'  &apos;
"  &quot;
```

Parsimaton data

```
<dataa>
     <![CDATA
          Sisältöä, jossa on
          "vaarallisia" merkkejä, kuten
          >, <, ", ' tai &
     ]]>
</dataa>
```

KUVIO 3. XML:n ennalta määritellyt entiteetit ja parsimaton data esimerkki (Nakhimovsky, Myers, 2002, 224.)

XML-kielen kannalta ongelmallisia ovat merkit, joita käytetään kielen syntaksin kuvaamiseen. Tällaisia ovat kuviossa 3 esitetyt entiteeteillä korvattavat merkit. Näitä merkkejä ei voi käyttää elementtien tai attribuuttien arvoissa, koska tuolloin merkkejä tulkitseva ohjelma ymmärtää ne dokumentin rakennetta kuvaaviksi merkeiksi.

XML-kieleen on toteutettu kaksi mahdollista tapaa selviytyä ongelmasta: Ensimmäinen tapa on käyttää ennalta määritettyjä entiteettiviittauksia korvaamaan ongelmalliset merkit. Entiteettiviittaus koostuu yksilöllisestä nimestä, jonka edessä on & -merkki ja sen jäljessä olevasta puolipisteestä (McLaugh 2001, 37). XML-dokumenttia tulkitseva ohjelma osaa korvata käytetyt viittaukset oikealla merkillä. Toinen tapa on sisällyttää data kuviossa 3 kuvatun CDATA-osion sisälle. CDATA-osiota käytetään tilanteissa, joissa dataa halutaan siirtää ilman, että XML-jäsennin tulkitsee sen sisältöä. Tilanne syntyy, jos suuri määrä merkkejä täytyisi korvata entiteettiviittauksilla tai, kun tekstin asemointi halutaan säilyttää. (McLaugh, 2001, 37–38.)

## 3.3 XML:n jäsentimet

XML-dokumentin ja sitä käyttävän sovelluksen välillä täytyy aina olla joko itse toteutettu tai vapaasti saatavilla oleva XML-jäsennin. Jäsennin lukee XML-dokumentin ”raakamuodossa” ja tulkitsee sen sekä varmistaa, että se on oikeanmuotoinen. XML:n ollessa jäsennetty, tuloksena on useimmiten jonkinlainen tietorakenne, jota muut XML-työkalut tai rajapinnat voivat käyttää hyväkseen. (McLaugh, 2001, 25.)

XML-jäsentimiä on erilaisia, ja niiden ominaisuuksissa on eroja. Jäsentimen valintaan vaikuttavat sen nopeus suoriutua tehtävästään sekä ominaisuudet, joita siltä vaaditaan. Ohjelman käyttäessä XML-dataa paljon jäsentimen tehokkuus on usein olennaisempaa kuin kaikkien XML:n ominaisuuksien tukeminen. Tehokkuuden nimissä jotkin jäsentimet jättävätkin vähemmän tarpeellisia ominaisuuksia huomioimatta ja näin toimivat nopeammin. Käyttökelpoista jäsennintä etsittäessä kannattaakin kiinnittää huomiota vaadittavien ominaisuuksien ja nopeuden väliseen tasapainoon. Suosittuja käytössä olevia jäsentimiä ovat esimerkiksi:

- Apache Xerces
- IBM XML4J
- James Clarkin XP
- Oracle XML parser
- Sun Microsystems Crimson
- Tim Brayn Lark ja Larval
- Electric XML
- Microsoft MSXML Parser.

(McLaugh, 2001, 26.)

## 3.4 XML ohjelmistorajapinnat

### 3.4.1 Yleistä

Jäsennin toteuttaa rajapinnat korkeamman tason ohjelmistorajapinnoille kuten SAX, DOM ja JDOM. Ohjelmistorajapintojen ei näin tarvitse huolehtia itse XML-datan matalantason käsittelystä. Käytännössä jäsennin huolehtii kokonaan siitä, kuinka XML-dokumentti saadaan käyttöön, sen lukemisesta sekä jäsentelemisestä

Ohjelmointirajapinnat toimivat sovelluslogiikan ja XML-jäsentimen välissä. Ne tarjoavat tavan, jolla sovellus pääsee käsiksi XML-dokumenttiin ja voi käsitellä sitä. Toteutukset voidaan karkeasti jakaa kahteen ryhmään: Toinen laatii dokumentista aikaan ja toinen tilaan perustuvat rakenteen. SAX liittää erilaiset tapah-

tumat kuhunkin XML-elementin aloitukseen ja lopetukseen sekä tekstilohkoon, kun taas DOM ja JDOM kuvaavat dokumentin puumaisena rakenteena. Molemmissa malleissa on hyvät ja huonot puolensa, ja niitä käytetäänkin hyvin erilaisissa tehtävissä. (Nakhimovsky, Myers, 2002, 275–277.)

### 3.4.2 SAX - ohjelmistorajapinta

SAX:n toiminta perustuu kokonaan tapahtumapohjaiseen XML:n käsittelyyn. Tapahtumia syntyy jäsentimen tulkitessa XML-dokumenttia. SAX onkin väline, jolla jäsennystapahtumat ja sovelluskohtainen ohjelmakoodi sidotaan toisiinsa. SAX tarjoaa rajapinnan, jonka kaikkien sen kanssa yhteensopivien XML-jäsentimien tulee toteuttaa. Ilman rajapinnan toteutusta ei SAX:ia voida käyttää kyseisen jäsentimen kanssa. (McLaugh, 2001, 60–66.)

Tapahtumapohjaisen käsittelyn etuja ovat nopeus ja vähäinen muistintarve. Muistiin ei talleteta kuin juuri käsiteltävänä oleva XML-data. SAX:in muistintarpeen vähyys onkin yksi sen parhaista ominaisuuksista, mutta on myös sen heikkous. Kun tietoja ei talleteta muistiin, niitä ei myöskään saada jälkeenpäin esille ilman dokumentin uudelleen käsittelyä. Tapahtumapohjaisessa mallissa ei ole mahdollista liikkua XML-dokumentissa taaksepäin tai siirtyä suoraan tiettyyn kohtaan dokumentissa. (McLaugh, 2001, 60–66.)

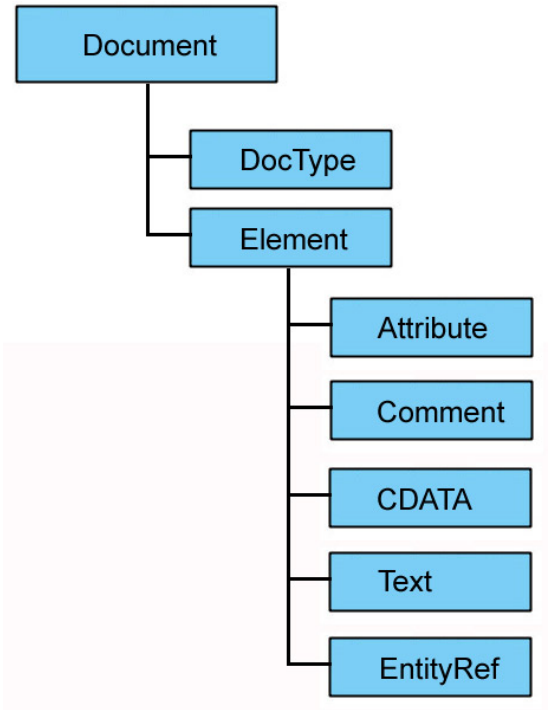
Tapahtumapohjaisessa XML:n käsittelyssä käytetään niin kutsuttuja takaisinkutsuja (callbacks), joita sanotaan myös sisällönkäsittelijöiksi (Content Handlers). Sisällönkäsittelijät ovat takaisinkutsuluokkia, jotka täytyy määritellä ennen XML-dokumentin käsittelyä, jotta sovellus voi hyödyntää jäsennettävän XML-datan. Takaisinkutsuluokat mahdollistavat sovelluskohtaisen ohjelmakoodin liittämisen SAX:n jäsennystapahtumiin. (McLaugh, 2001, 70.)

### 3.4.3 JDOM - ohjelmistorajapinta

#### 3.4.3.1 Perusteet

JDOM ohjelmointirajapinta perustuu ns. puumalliin. Mallin perustana on muodostaa XML-datasta puuesitys, jossa voidaan liikkua eteen ja taaksepäin ja jota voidaan myös muokata. Puumallilähestymistavan etuna onkin dokumentissa liikkumisen ja muokkaamisen helppous. JDOM muistuttaa W3C:n määrittelemää DOM rajapintaa puumaisen rakenteen ansiosta, mutta se on DOM:ista poiketen kehitetty erityisesti Java-kielelle ja sopiva valinta Java-ympäristössä. JDOM:in kehitystyö alkoi Java-konferenssissa vuonna 2000 DOM:iin liittyvien ongelmien ratkaisemiseksi. (McLaugh, 2001, 190–191.)

Puumallisessa XML-datan käsittelyssä koko dokumentti täytyy lukea aluksi muistiin, jotta sitä voidaan käsitellä. Suurten XML-dokumenttien kohdalla tämä toimintamalli vie paljon aikaa ja voi olla jopa mahdotonta. JDOM ohjelmistorajapinnan suurimpana heikkoutena onkin suuri muistinkulutus ja se, ettei XML-dataan päästä heti käsiksi.




KUVIO 4. UML-malli tärkeimmistä JDOM-luokista (McLaugh, 2001, 191)

Kuviossa 4 esiteltyjen luokkien nimet kertovat paljon ohjelmistorajapinnan rakenteesta. JDOM on konkreettisiin luokkiin perustuva ohjelmistorajapinta. Kaikki luokat voidaan näin luoda suoraan Javan new-lauseella.

#### 3.4.3.2 JDOM:in luokkia

JDOM:in ydin on Document-luokka (Kuvio 4), joka samalla vastaa XML-dokumenttia. Document-luokka toimii myös säiliönä kaikille muille luokille, ja suora viittaus siihen on saatavilla kaikista sen alla olevista ilmentymistä. Luonnissa Document-luokalle täytyy antaa juurielementti, joka on tavallisesta Element-luokasta muodostettu olio. Dokumentin juurielementti voidaan myöhemmin tarvittaessa korvata uudella elementillä. (McLaugh, 2001, 60–66, 530.)

Element-luokka on Java-esitys XML-elementistä. Java elementti sisältää XML-elementin arvon, ja tämän lisäksi kaikki elementin alla olevat lapsielementit. Elementin arvo saadaan helposti haettua ja asetettua. Lapsielementtien käsittelyyn on useita metodeita, kuten niiden lisäys ja poisto. Tarvittaessa elementtejä voidaan hakea nimen tai indeksin mukaan. Kaikki lapsielementit saadaan haettua Javan listarakenteeseen, ja näin elementtejä voidaan käydä läpi rekursiivisesti. Elementistä päästää käsiksi myös kaikkiin sen attribuutteihin ja niiden arvoihin. Tämä nopeuttaa rajapinnan hyödyntämistä, ja toteutuksen ansiosta ohjelmakoodi on selkeämpää ja ymmärrettävämpää. (McLaugh, 2001, 531.)

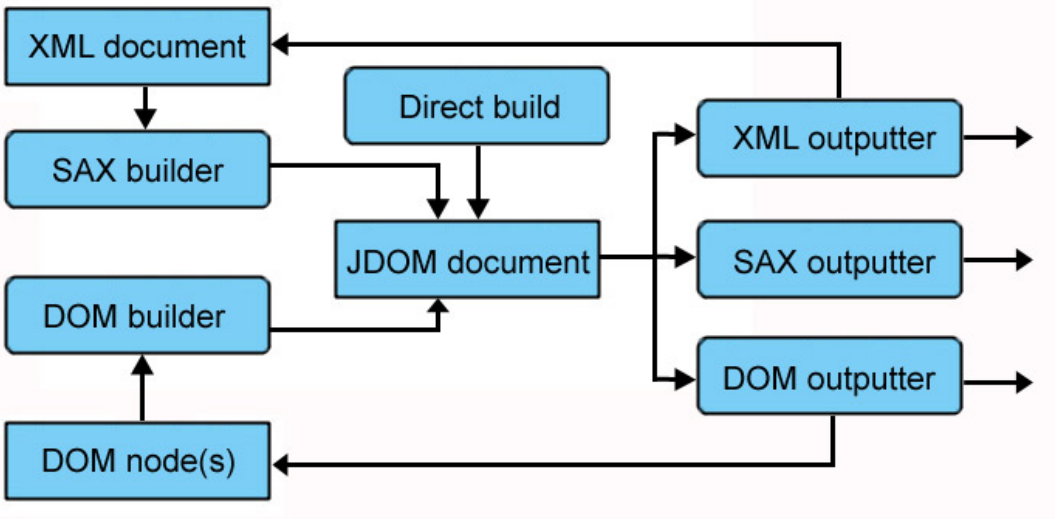
Text-luokka esittää JDOM-elementissä olevan merkkimuotoisen datan. Luokka on yleensä näkymätön käyttäjälle, sillä Element-luokka muuntaa sen yksinkertaiseksi merkkijonoksi, kun elementin arvoa pyydetään. (McLaugh, 2001, 535.)

Attribute-luokka määrittelee XML-attribuutin käyttäytymisen ja siinä olevat metodit sallivat käyttäjää hakea ja asettaa attribuutin arvon. Attribuutti-luokalle on toteutettu helpottavia hakumetodeita, joilla arvon saa suoraan tarvittavana Javan tietotyyppinä, kuten boolean, double, float ja int. (McLaugh, 2001, 528.)

CDATA-luokka määrittelee XML:n CDATA-osioiden käyttäytymisen. Luokalta voidaan ainoastaan pyytää sen sisältöä. Ainoa tapa asettaa sisältöä on luoda luokasta uusi ilmentymä. Comment-luokka määrittelee XML-dokumentissa olevien kommenttien toiminnan ja sisältää myös kommenttien tekstin. Comment-luokasta luodulle ilmentymälle voidaan asettaa arvo. DocType-luokka esittelee XML-dokumentin DOCTYPE-määritykset ja sillä on tarvittavat metodit niin arvojen hakemiseen kuin asettamiseenkin. (McLaugh, 2001, 529–530.)

#### 3.4.3.3 Syöttö ja tulostus

JDOM:in on turvauduttava jäsentimeen puurakenteen luomisessa, luettavan datan ollessa XML-dokumenttina. Se hyväksyy syötteenä myös SAX-tapahtumia ja DOM-puita. Datan syöttöä varten JDOM paketissa on oma osio, joka sisältää erilaisia luokkia, joiden avulla eri lähteistä saadaan rakennettua JDOM-puu. (McLaugh, 2001, 190–193.)




KUVIO 5. Syöte- ja tulostussilmukat JDOM:issa (McLaugh 2001,195)

Syötteitä käsittelevässä osiossa sijaitsevien SAXBuilder- ja DOMBuilder-luokkien (Kuvio 5) avulla rakennetaan kaikki JDOM puut. SAXBuilder on nopea ja tehokas tapa käsitellä XML-dataa, kun se sijaitsee tiedostossa tai data on käytettävissä syöttövirtana. Rakentaja on yhtä tehokas ja hyvä kuin SAX-

ohjelmistorajapinta itse. DOMBuilderilla konvertoidaan olemassa oleva DOM-puu JDOM-puuksi. (McLaugh, 2001, 193–195.)

Tulostukseen JDOM-rajapinta tarjoaa erilaisia toteutuksia. Olemassa olevasta puurakenteesta saadaan tulostettua DOM-puurakenne, SAX-tapahtumia tai normaali XML-tiedosto (Kuvio 5).

# 4 SUUNNITTELUMALLIT

## 4.1 Tarkoitus ja historiaa

Suunnittelumalli on tapa toteuttaa ohjelmiston osa tai kokonaisuus usein esiintyvän ongelman ratkaisemiseksi. Ne antavat yleisen tason ratkaisuja olio-ohjelmoinnissa esiintyviin ongelmiin. Suunnittelumallien isänä voidaan pitää Erich Gamma:a, joka esitti idean väitöskirjassaan. Gamma kirjoitti nk. ”Gang of Four”- ryhmän kanssa suunnittelumallien perusteoksen *Design Patterns – Elements of Reusable Object-Oriented Software*.

Suunnittelumalleja käyttämällä pystytään uudelleen käyttämään hyväksi havaittuja ratkaisuja ja arkkitehtuureja. Erilaisten tekniikoiden esittäminen suunnittelumalleina helpottaa niiden käyttöä uusia ohjelmia rakennettaessa. Suunnittelumallit auttavat ohjelmistokehittäjää valitsemaan parempia ratkaisuja, jotka tekevät ohjelmistosta uudelleenkäytettävämmän ja helpommin kehitettävän. Mallit myös parantavat ohjelmien dokumentaatiota ja ylläpidettävyyttä tarjoamalla selkeän määrityksen luokkien ja olioiden suhteista, rajapinnoista, vastuista sekä niiden käyttötarkoituksista. (Gamma, Helm, Johanson, Vlissides, 2000, 2.)

## 4.2 Mallien rakenne

Suunnittelumalli jaetaan yleensä neljään osaan: mallin nimi, ongelma ja ympäristö, ratkaisu ja seuraukset. Mallin nimi kuvaa muutamalla sanalla suunnittelun kohteena olevan ongelman, sen ratkaisun ja ratkaisun vaikutukset. Nimi on tärkeä, koska sen avulla tiedetään mistä mallista on kysymys. Nimeä käytetään myös ohjelmiston kehityksen eri vaiheissa. Siihen viitataan suunniteltaessa ohjelman rakennetta, määriteltäessä toiminnallista ja teknistä kuvausta ja toteutettaessa itse ohjelmaa. Hyvä nimi kertoo ohjelmistosuunnittelijoille mallin luonteesta ja mahdollisista käyttökohteista sekä millaisiin ongelmiin sitä voidaan käyttää. Suunnittelumallien nimeäminen rikastuttaa suunnittelusanastoa ja auttaa samalla suunnittelemaan korkeammalla abstraktiotasolla. (Gamma ym, 2000, 3.)

Mallin ongelma esittää ne tilanteet, joihin malli tarjoaa ratkaisun tai sitä voidaan soveltaa. Ongelma määritellään tarkasti, ja sen perusteella voidaan päätellä, sopiiko kyseinen malli olemassa olevaan ongelmaan. Ongelmaosan yhteydessä voidaan esitellä jokin tyypillinen ongelma, tai se voi sisältää listan ehdoista, joiden tulee täyttyä, ennen kuin malli on hyödyksi ohjelmistokehittäjälle. (Gamma ym., 2000, 3.)

Ratkaisussa kuvataan mallin osat, joista suunnittelumalli koostuu, osien vastuut sekä niiden väliset suhteet ja niiden keskinäinen yhteistyö. Malli tarjoaa yleisen tason kuvauksen ongelmasta ja ratkaisusta. Mallin ratkaisu tarjoaa rungon, jota ohjelmistokehittäjä voi soveltaa omaan ongelmaansa. (Gamma ym., 2000, 3.)

Seuraukset kuvaavat mallin soveltamisesta syntyviä tuloksia ja sen hyötyjä sekä haittoja. Seurauksien avulla suunnittelija voi harkita eri ratkaisuvaihtoehtoja ja niiden sovellettavuutta. Seurauksien perusteella voidaan perustella tietyn mallin käyttö juuri käsillä olevan ongelman yhteydessä sekä ottaa huomioon ohjelmiston joustavuus, laajennettavuus ja uudelleen käytettävyys tulevaisuudessa. (Gamma ym., 2000, 3.)

## 4.3 Mallien luokittelu

Suunnittelumalleja on olemassa paljon ja ne ovat myös hyvin erilaisia. Yksi malleista voi ratkaista hyvin pienen ja tarkasti rajatun ongelman, kun taas toisen avulla toteutetaan koko ohjelman perusrakenne. Samalla myös mallien karkeus- ja abstraktiotaso vaihtelevat paljon. Näistä syistä suunnittelumallit on jäsennelty ja luokiteltu ryhmiksi. Ryhmittely helpottaa ja nopeuttaa mallien löytämistä ja käyttöönottoa.

TAULUKKO 1. Suunnittelumallien luokittelu (Gamma ym., 2000, 10)

| | | **Tarkoitus** | | |
|---|---|---|---|---|
| | | **Luonti** | **Rakenne** | **Käyttäytyminen** |
| **Kohde** | **Luokka** | Tehdasmetodi | Sovitin | Tulkki<br>Operaatiorunko |
| | **Olio** | Abstrakti tehdas<br>Rakentaja<br>Prototyyppi<br>Ainokainen | Sovitin (olio)<br>Silta<br>Rekursiokooste<br>Kuorruttaja<br>Julkisivu<br>Hiutale<br>Edustaja | Vastuuketju<br>Komento<br>Iteraattori<br>Välittäjä<br>Muisto<br>Tarkkailija<br>Tila<br>Strategia<br>Vierailija |

Erich Gamma ym. luokittelevat kirjassaan Design Patterns suunnittelumallit kahden kriteerin mukaan (Taulukko 1.). Ensimmäinen kriteeri, tarkoitus, kuvaa mallin toimintaa. Mallit jaetaan vielä ryhmiin käyttötarkoituksensa perusteella. Luontimallit käsittelevät olioiden luontia, rakennemallit luokkien ja olioiden koosteita ja käyttäytymismallit tapoja, joilla luokat ja oliot ovat vuorovaikutuksessa ja joilla ne jakavat vastuita keskenään.

Kohde kriteeri kertoo, liittyykö suunnittelumalli ensisijaisesti luokkiin vai olioihin. Luokka- ja oliomallit käsittelevät luokkien ja olioiden välisiä suhteita. Näiden kahden mallin tärkein ero on suhteiden toteutuksessa. Luokkamallien suhteet luodaan periytymisen kautta, joten ne ovat pysyviä eli käännösaikaisia, eikä uusia suhteita voida luoda ajonaikaisesti. Oliomallit käsittelevät olioiden välisiä suhteita, joita voidaan muuttaa ajonaikaisesti ja ovat näin dynaamisempia. Lähes kaikki mallit käyttävät perintää ainakin joissain määrin, ja tästä johtuen luokkamalleiksi on hyväksytty vain ne mallit, jotka keskittyvät ainoastaan luokkien välisiin suhteisiin. Ilman tätä rajoitetta lähes kaikki mallit olisivat luokkamalleja. (Gamma ym., 2000, 10.)

Mallien toiminta on erilaista riippuen, liittyvätkö ne ensisijaisesti luokkiin vai olioihin. Luokkiin liittyvät luontimallit siirtävät olioiden luontia aliluokille, kun taas olioihin liittyvät mallit siirtävät toimenpiteen toiselle oliolle. Rakennemalleissa luokat käytävät perintää luokkakoosteiden muodostamiseen, kun taas olioi-

hin liittyvät rakennemallit esittävät, kuinka erilaisia olioita koostetaan. Käyttäytymismalleissa luokat käyttävät perintää algoritmien ja kontrollin ohjauksessa. Olioihin liittyvät käyttäytymismallit määrittelevät miten useampi olio tekee yhteistyötä tehtävässä, johon yksittäisen olion suorituskyky ei riitä. (Gamma ym., 2000, 10.)

Kaikki mallit eivät ole täysin erilaisia. Joitakin malleja käytetään toistensa kanssa, kun taas toiset ovat toistensa vastakohtia. Joidenkin mallien tulos saattaa olla täysin sama, vaikka niiden tarkoitus on erilainen.

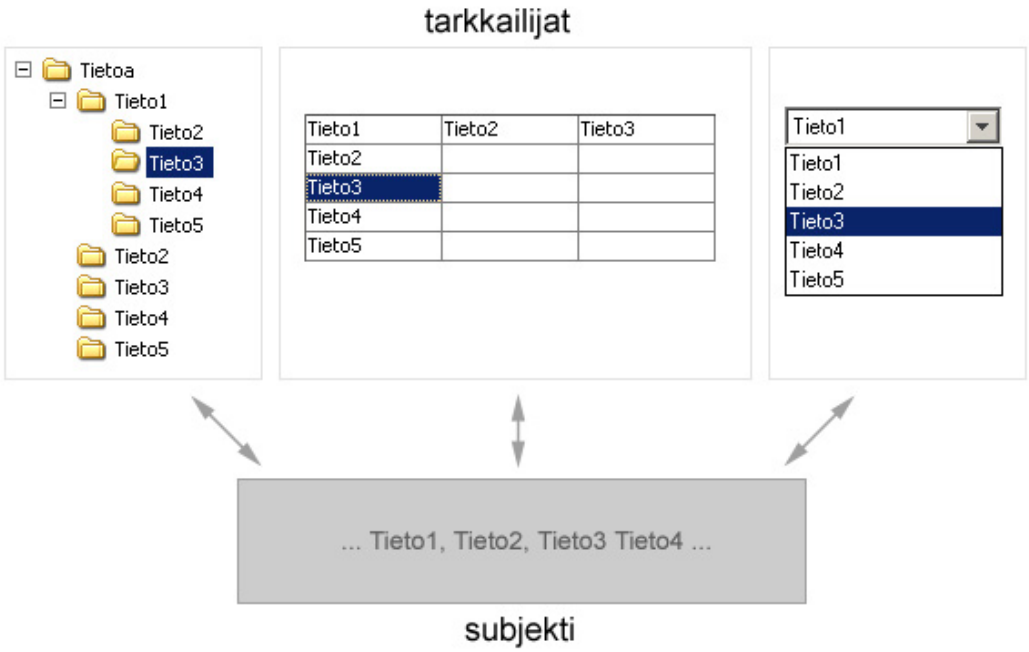
Edellä esitelty luokittelu ei ole ainoa tapa luokitella malleja. Muita luokittelutapoja voisivat olla esimerkiksi mallien käyttökohteiden, laajuuden tai riippuvuuksien perusteella. Malleja olisi mahdollista luokitella myös sen mukaan, millaisessa ympäristössä niitä käytetään.

## 4.4 Tarkkailija

Tarkkailija suunnittelumallin tarkoituksena on välittää tietoa olioin tilan muutoksista toisille olioille, jotka ovat kiinnostuneita kyseisistä tiloista. Tarkkailija (observer) mallia käytetään usein silloin, kun joukko olioita toimii yhteistyössä tai käyttää ja muuttaa yhteistä sovellusdataa. Tällaisissa tilanteissa on tärkeää huolehtia siitä, että olioiden tilat ovat ajan tasalla ja yhtenäisiä. Mallia käytetään myös tilanteissa, joissa luokkia ei haluta sitoa kiinteästi toisiinsa ja näin edistetään niiden uudelleenkäytettävyyttä. (Gamma ym., 2000, 293.)

Tarkkailija-mallin tärkeimmät oliot ovat subjekti ja tarkkailija. Kaikki tarkkailijat ilmoittautuvat subjektille ja näin ollen subjekti tuntee kaikki oliot, jotka ovat kiinnostuneita tilojen muutoksista. Tarkkailijaluokka määrittelee olioille rajapinnan, joiden kautta ne saavat tiedon subjektin tilassa tapahtuneista muutoksista.

Tarkkailija-malli soveltuu tilanteisiin joissa muutos yhdessä oliossa vaatii muidenkin muuttamisen eikä tiedetä kuinka monta oliota on muutettava sekä tilanteisiin joissa olion on lähetettävä ilmoitus muille olioille ilman tietoa muista olioista.




KUVIO 6. Tarkkailijamalli käyttöliittymäkomponenttien yhteydessä

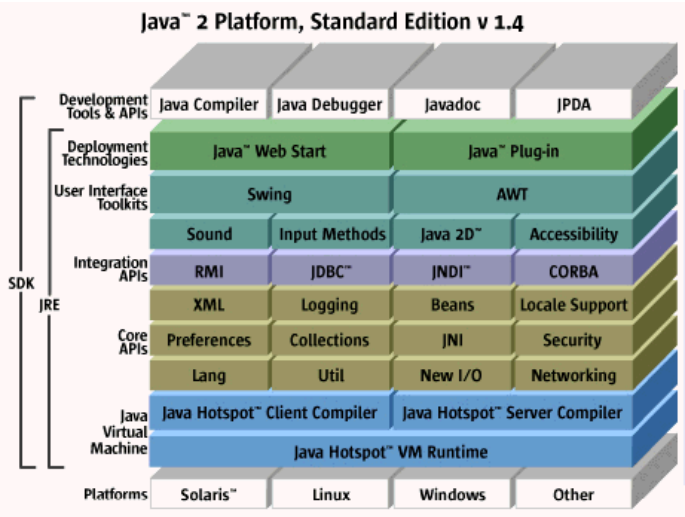
Tarkkailija malli on erityisen hyödyllinen, kun erilaista dataa esitetään ja muokataan monilla graafisilla käyttöliittymäkomponenteilla. Jokaisen komponentin tulee olla ajan tasalla ja esittää käyttäjälle yhtenäinen tilanne suhteessa käytössä olevaan dataan. Jos jokaisella komponentilla ole yhteneväistä tietoa datan tilasta, ne voivat esittää käyttäjälle tilanteita, jotka eivät ole enää voimassa. Kriittinen tilanne syntyy, kun jonkin komponentin välityksellä esimerkiksi "Tieto3" kansio (Kuvio 6) poistetaan ja tieto tapahtumasta ei siirry kaikille käyttöliittymäkomponenteille. Ohjelman käyttäjä näkee jo poistetun tiedon päivittymättömistä komponenteista ja voi valita kyseisen poistetun hakemiston. Kuvatunlaisessa tilanteessa käyttöliittymäkomponentti yrittää käsitellä tietoa, jota ei enää ole olemassa; todennäköisyys ohjelman kaatumiselle tai virhetoiminnoille on suuri riippuen olion ja datan toteutuksesta.

# 5 JAVAN GRAAFISEN KÄYTTÖLIITTYMÄN OHJELMOINTI

## 5.1 Java

Java on James Gosling:in ja Sun Microsystems:in 1990 alkupuolella kehittämä alustariippumaton oliopohjainen ohjelmistokieli. Kieli muistuttaa kieliopiltaan huomattavasti C++-kieltä, mutta ei C++:n tavoin mahdollista rakenteellisen- ja olio-ohjelmoinnin sekoittamista keskenään. Java onkin suunniteltu kokonaan oliopohjaiseksi kieleksi.

Javan tärkein yksittäinen tekijä on sen alustariippumattomuus. Sen ansiosta Javalla tehtyjä ohjelmia voidaan siirtää ja käyttää useissa eri ympäristöissä ilman, että ohjelmakoodia täytyy muokata. Alustariippumattomuuden taustalla on Javan virtuaalikone (Kuvio 7), joka tulkitsee ohjelmakoodin käytetyn alustan käyttöjärjestelmälle.




KUVIO 7. Javan versio 1.4:n arkkitehtuuri (java.sun.com)

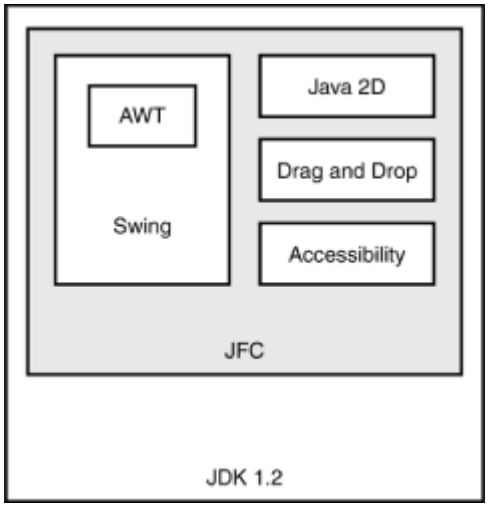
## 5.2 AWT:n ja Swingin historia

Javan 1.0 versiossa oli jo yksinkertainen graafisen käyttöliittymän toteutuksen soveltuva kirjasto. Kirjastosta käytettiin nimeä Abstract Window Toolkit eli AWT. Kirjaston toimintaperiaatteena oli, että se delegoi eri käyttöliittymän osien luomisen ja ohjaamisen käyttöjärjestelmäkohtaisille (Windows, Solaris, Mac, Linux ja jne.) GUI-työkaluille. AWT:n käyttämän mallin idea on se, että ohjelmoija määrittelee käyttöliittymän osien sijainnin ja toiminnot, mutta Java luo niiden lisäksi rinnakkaisosat. Javan luoma rinnakkaisosa huolehtii esimerkiksi tekstilaatikossa tekstin syöttämisestä. (Horsmann, Cornell, 2000, 260)

AWT:llä toteutettu ohjelma toimii ainakin teoriassa kaikissa tuetuissa käyttöjärjestelmissä ja muistuttaa niitä myös ulkoasultaan ja toiminnaltaan. Menetelmä sopii kuitenkin vain yksinkertaisiin sovelluksiin, ja sillä on vaikea kirjoittaa laadukasta ja siirrettävää grafiikkakirjastoa, joka on riippuvainen eri käyttöliittymän osista. Valikot, vierityspalkit, tekstiruudut ja muut käyttöliittymien osat voivat olla erilaisia ja toimia hiukan eri tavalla eri ympäristöissä. Tämän lisäksi joissakin ympäristöissä, kuten Linux alustalla, toimivat X11- tai Motif-kirjastot eivät sisällä yhtä laajoja ominaisuuksia kuin Windows-käyttökärjestelmässä tai Machintosh:issa toimivat kirjastot. Nämä seikat rajoittavat AWT kirjaston ominaisuuksia, ja siksi sen avulla toteutetut sovellukset eivät näytä yhtä hienoilta ja voivat sisältää erilaisia toimintoja kuin alkuperäiselle käyttöjärjestelmälle toteutetut ohjelmat. Eräs AWT:n ongelma on, että kirjastossa esiintyvät virheet ovat erilaisia ympäristöstä riippuen, joten kaikki ominaisuudet täytyy testata jokaisessa ympäristössä erikseen. (Horsmann, Cornell, 2000, 260)

Vuonna 1996 Netscape-niminen yhtiö teki kirjaston nimeltä IFC (Internet Foundation Classes). Kirjaston toteutus käyttöliittymäkomponenttien muodostamiselle perustui erilaiseen menetelmään. Käyttöliittymän eri osat, kuten painikkeet ja valikot, eivät riippuneet suoraan järjestelmäkohtaisista GUI-työkaluista vaan suurin osa niiden toiminnoista ja näkymästä oli mukana kirjastossa. Tästä johtuen Netscapen komponentit näyttivät ja toimivat samalla tavalla eri ympäristöissä. Sun Microsystems paranteli toteutusta yhteistyössä Netscapen kanssa ja kehitti uuden Swing nimisen käyttöliittymäkirjaston. (Horsmann, Cornell, 2000, 260.)

Java-kielestä julkaistiin versio 1.2 (Java2) vuonna 1998. Java2-versio sisälsi kehittyneen grafiikka- ja käyttöliittymäkirjaston nimeltään Java Foundation Classes (JFC). Tärkein teknologia julkaistussa JFC-paketissa oli Swing-grafiikkakirjasto. Kirjasto sisältää swing-komponenttien lisäksi kuviossa kahdeksan esiteltyjä osia kuten vammaisille tarkoitettu-, kaksiulotteinen-, ja vetoon ja pudotukseen perustuva API. (Pantham, 1999, Overview.)




KUVIO 8. Kuinka Javan versio 1.2, JFC ja Swing liittyvät toisiinsa. (Pantham, 1999, Luku 1)

Swing-kirjasto ei korvannut AWT:ta kokonaan vaan se hyödyntää siinä olevia rakenteita, kuten tapahtumankäsittelyä. Monet swing-komponentit periytyvät suoraan AWT-luokista, ja niitä voidaankin käyttää samassa sovelluksessa sekaisin. Lähes kaikki AWT-grafiikkakirjastossa sijaitsevat komponentit on toteutettu myös Swing-kirjastoon. Komponentit on nimetty yhtenevästi sillä erolla, että Swing kirjastossa nimen edessä on J-kirjain. Osa komponenteista on kuitenkin uudelleen nimetty kuvaavammalla nimellä, kuten esimerkiksi, Choice-luokkasta on tullut JComboBox. Joidenkin AWT-komponnettien toiminnot on yhdistetty samaan Swing komponenttiin. (Zukowski, 2000, Luku 1.)

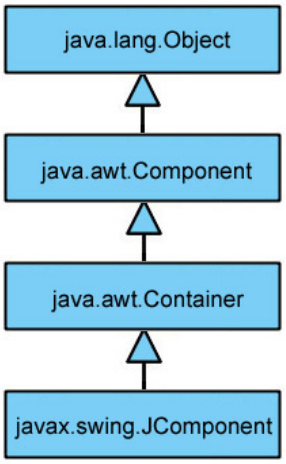
Swingiin perustuvat komponentit latautuvat näytölle hiukan AWT:n komponentteja hitaammin, mutta nykyisillä tietokoneilla tätä eroa on lähes mahdotonta havaita. Suurimmat edut AWT:hen verrattuna ovat, että:

- Swingissä on paljon laajempi ja kätevämpi valikoima erilaisia käyttöliittymäkomponentteja.
- Swing ei pohjaudu yhtä paljon sitä suorittavan ympäristön ominaisuuksiin ja on näin vähemmän altis ympäristökohtaisille virheille.
- Swingillä toteutetut ohjelmat näyttävät käyttäjälle samoilta kaikissa ympäristöissä.

Kolmantena esitelty etu voi joissakin tilanteissa olla myös haitta sillä, jos käyttöliittymäkomponentit näyttävät ulkoisesti samoilta ympäristöstä riippumatta, ne poikkeavat varmasti joiltain osin ympäristön omista komponenteista ja tästä johtuen käyttäjät voivat oudoksua niitä. (Horsmann, Cornell, 2000, 260–261.)

## 5.3 Swing-komponentit

Swing-komponenttien suunnittelun perustana voidaan pitää MVC-suunnittelumallia. Suunnittelumallissa data, näkymä ja ohjelmalogiikka ovat erillään ja riippumattomia toisistaan. Mallin antaa mahdollisuuden muokata swing-komponenttien ulkoasua sekä sen luonnissa käytettyä tietorakennetta tarpeiden mukaan.




KUVIO 9. JComponentin luokka-hierarkia. (Zukowski, 2000, Luku 2)

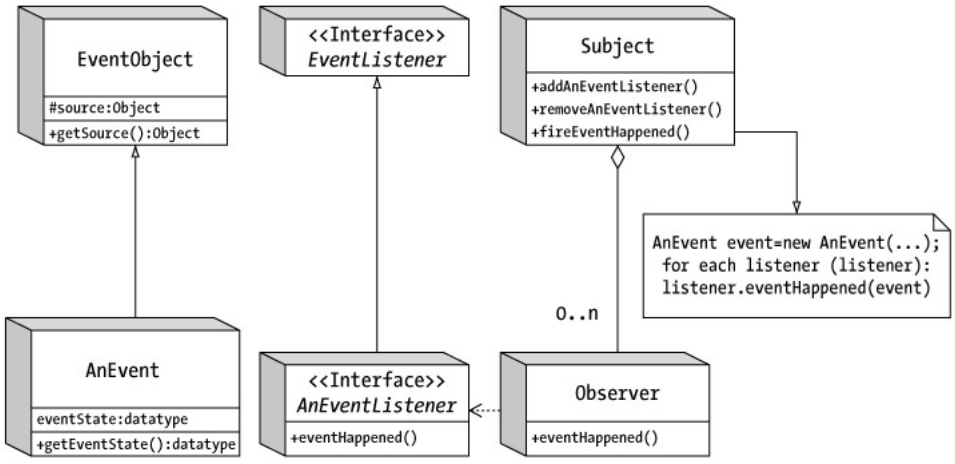
Kaikki Swing-käyttöliittymäkomponentit on peritty kuvion 9 mukaisesti abstraktista JComponent-luokasta, ja se tarjoaa yksinkertaisia ominaisuuksia ja toimintoja lapsiluokilleen. Luokka tarjoaa muun muassa:

- käytettävyysrajapinnan vammaisille
- ulkoiset reuna- ja raja-alueasetukset
- näppäinpainalluksien käsittelyn
- mahdollisuuden lisätä tapahtuville muutoksille kuuntelijoita
- mahdollisuuden kertoa tapahtuvista muutoksista kuuntelijoille
- komponentin tasauksen piirtoalueelle sekä automaattisen vierityksen
- grafiikkapuskuroinnin
- vihjeiden (tooltips) näytön
- kohdistuksen (focus) vastaanottamisen
- komponentin mittasuhteiden muutokset.

(Pantham, 2000, Luku 1)

### 5.3.1 Tapahtumien käsittely

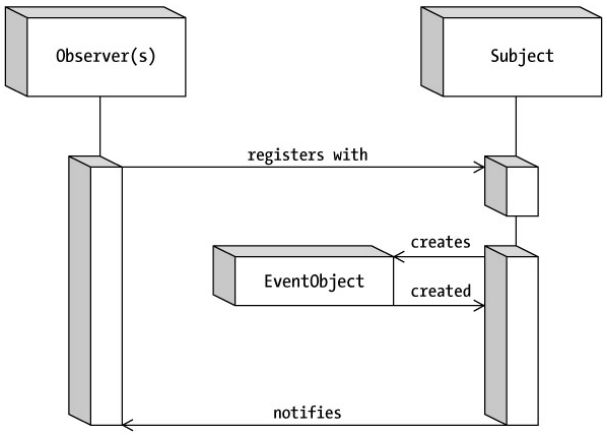
Kaikkien graafista käyttöliittymää tukevien käyttöjärjestelmien on jatkuvasti tarkkailtava ympäristöstä tulevia tapahtumia kuten näppäinpainalluksia tai hiiren eri toimintoja. Käyttöjärjestelmä ilmoittaa tapahtumista käsittelyssä oleville ohjelmille, ja kukin näistä ohjelmista käyttäytyy toteutuksensa mukaisesti.




KUVIO 10. Muunneltu tarkkailija malli (Zukowski, 2000, Luku 2)

Javan käyttöliittymäkomponenttien tapahtumienkäsittely perustuu kuviossa 10 esitettyyn muunneltuun tarkkailijamalliin. Mallissa kaikille tapahtumia tuottaville olioille täytyy asettaa kuuntelija, jotta tapahtumaan voidaan reagoida jotenkin. Kuuntelija on sellaisen luokan ilmentymä, joka toteuttaa EventListener-rajapinnan ja tämän rajapinnan kautta tapahtuman tuottaja ilmoittaa tapahtumasta kaikille rekisteröityneille kuuntelijoilleen. (Horstmann, Cornell, 2000, 317.)

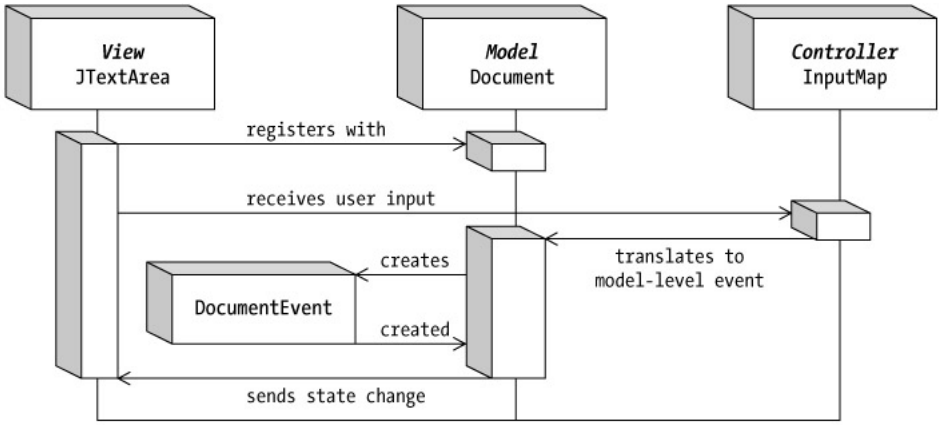
Ilmoittaessa kuuntelijoille tapahtumasta tapahtuman tuottaja luo ensiksi EventObject tapahtuma olion (Kuvio 11) ja välittää sen kaikille kuuntelijoilleen tapahtumailmoituksen yhteydessä.




KUVIO 11. Kuuntelija mallin tapahtumakaavio
(Zukowski, 2000, Luku 2)

### 5.3.2 Swingin MVC-kommunikointi

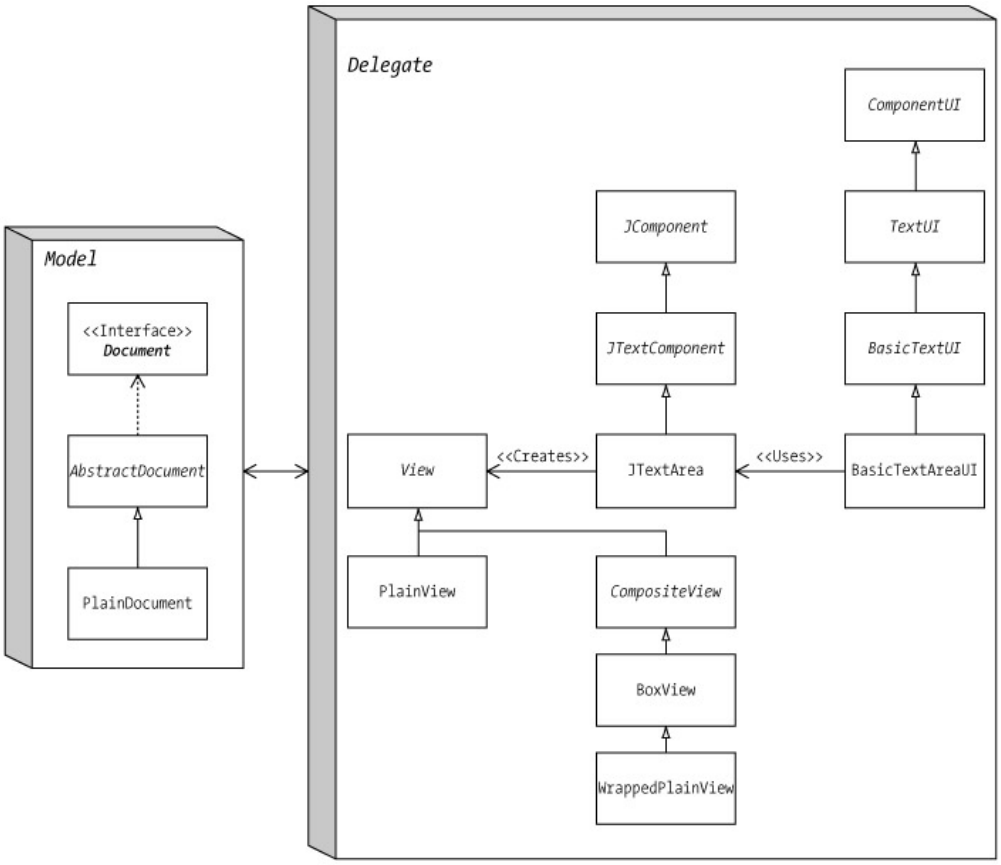
Swing-komponenttien toiminnan perustana toimiva MVC-suunnittelumalli esiteltiin Smalltalk-ohjelmointikielessä 1980-luvun lopussa. MVC-malli koostuu kolmesta erillisestä malli- (view), näkymä- (model) ja kontrolliosasta (controller). Malli huolehtii tiedon tallettamisesta, ylläpidosta ja käsittelystä. Näkymä määrittelee ulkoasun sekä mallin tietojen esitystavan ja kontrolli vastaanottaa käyttäjältä tulevat käskyt ja muuttaa mallia ja näkymää niiden mukaan.




KUVIO 12. MVC-kommunikaatio mekanismi
(Zukowski, 2000, Luku 3)

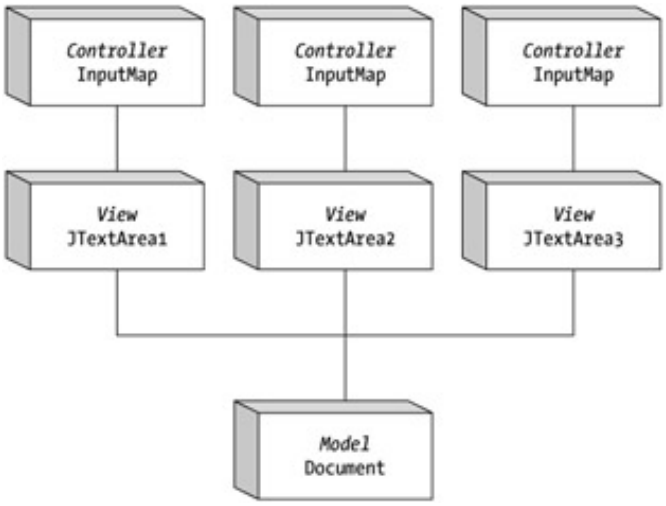
Kuviossa 12 esitetään, kuinka MVC-mallin eri osat kommunikoivat keskenään. Kuvion tapauksessa Swingin monirivinen tekstikomponentti JTextArea toimii MVC-mallin näkymäosana. Document sisältää komponentin tilaan liittyvät ominaisuudet sekä itse tekstisisällön. Tekstikomponentin tapauksessa kontrolliosana toimii InputMap-luokka, joka ottaa vastaan käyttäjän syöttämät komennot näppäimistöltä ja välittää ne Document-luokalle. Näppäintapahtuman tapahduttua Document luo DocumentEvent tapahtuma-olion ja lähettää sen JTextArea-komponentille. (Zukowski, 2000, Luku 3.)

Esimerkki havainnollistaa MVC-mallin tärkeää roolia Swing-kirjaston sisällä. Jokainen erillinen tapahtuma laukaisee monimutkaisen tapahtumaketjun näkymän ja kontrolliosan välillä. Swing-suunnittelussa nämä kaksi osaa yhdistetään edustus-objektiksi (Kuvio 13) helpottamaan kokonaissuunnittelua. Lopputuloksena jokaisella Swing-komponentilla on UI-edustaja, joka huolehtii komponentin graafisesta esittämisestä ja käyttäjän toimista syntyneistä tapahtumista, sekä malli, joka huolehtii komponentin tietosisällöstä. (Zukowski, 2000, Luku 3.)




KUVIO 13. JTextArea MVC-arkkitehtuuri. (Zukowski, 2000, Luku 3)

Monimutkaiselta tuntuvan mallin hyödyt tulevat esille, kun samaa dataa käyttävät eri Swing-komponentit. Datan ollessa yhden ilmentymän sisällä voivat useat komponentit hyödyntää sitä samanaikaisesti (Kuvio 13). Jokaisen komponentit näkymäosat voivat esittää datasta oman esityksensä, ja yhdessä komponentissa tehdyt datan muutokset tulevat näkyviin myös toisissa. Muutoksia tekevän komponentin ei tarvitse olla tietoinen muista, vaan päivitys hoituu automaattisesti yhteisen Document-olion välityksellä (Kuvio 14).




KUVIO 14. Jaettu MVC-data malli. (Zukowski, 2000, Luku3)

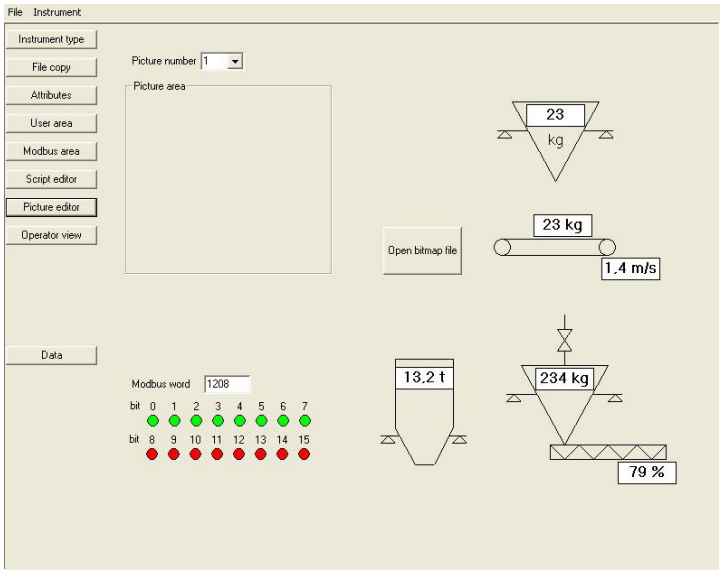
Monilla Swing-komponentilla, kuten napeilla, listoilla, taulukoilla ja puilla on valmiina rajapinnat ja toteutukset data-mallin hyödyntämiselle. Komponentille voidaan antaa jo luonnin yhteydessä käytettävä malli tai se voidaan asettaa jälkeenpäin.

# 6 SOVELLUSKEHITTIMEN SUUNNITTELU JA TOTEUTUS

## 6.1 Vaatimukset ja työn lähtökohta

Vaakaohjaimen sovelluskehittimelle määritettyjä vaatimuksia olivat käyttöliittymien räätälöintimahdollisuus, erilaisten tuotevarianttien nopea toteutus, alustariippumattomuus sekä ohjainlaitteen alustamisen helpottaminen ja mahdollinen automatisointi. Ennen työn aloittamista kaikkia vaatimuksia ja tarpeita ei pystytty toteamaan, ja tästä johtuen uusia ominaisuuksista syntyi kehitysprojektin aikana.

Ennen opinnäytetyön aloittamista sovelluskehittimestä oli jo tehty alustava vaatimusmäärittely. Tämän lisäksi oli toteutettu prototyyppiohjelma herättämään keskustelua suunnittelijoiden avuksi. Työn käytännön osuudessa ei jatkettu prototyyppiohjelman kehittämistä, vaan ohjelman toteutus aloitettiin alusta. Prototyyppiohjelmasta (Kuvio 15) oli kuitenkin suuri apu suunnan näyttäjänä, kun erilaisia vaatimuksia ja toiminnallisuuksia mietittiin.




KUVIO 15. Sovelluskehittimen prototyyppi

Prototyyppiohjelma oli tuotettu käyttäen Visual Basic ohjelmointikieltä, johon lopullinen sovelluskehitin ei kuitenkaan perustunut. Kehittimen toteutuskieleksi valittiin Java. Valintaan vaikuttivat pyrkimys alustariippumattomuuteen ja jo Ja-

valla toteutettu ohjainlaitteen käyttöliittymäohjelma. Osa syynä voidaan myös pitää toteuttajan mielenkiintoa ja oppimishalua valittua kieltä kohtaan.

## 6.2 Sovelluskehittimellä tuotettavat projektit ja laitteet

Sovelluskehitinlähtöisen projektinhallinnan taustalla on tuoteperheajattelu, jossa samalle tekniselle alustalle on mahdollista toteuttaa nopealla tahdilla erilaisia tuotevariantteja. Erilaisista laitteista on olemassa tiettyjä standardimalleja, joita pyritään suosimaan ja joiden kautta yksittäiselle asiakkaalle lähdetään toteuttamaan haluttua ratkaisua. Standardimallit eivät kuitenkaan aina tarjoa kaikkia vaadittuja ominaisuuksia tai oikeanlaista käyttöliittymää ja tästä johtuen joudutaan laitteita räätälöimään asiakkaan vaatimusten ja toiveiden mukaisiksi.

Sovelluskehittimellä toteutettava projekti sisältää kaikki samaan toimitukseen sisältyvät laitteet. Jokaisella projektille on annettu yleisentason määritykset, joiden avulla ne tunnistetaan ja erotetaan toisista. Yksittäiset laitteet (Kuvio 16) eivät välttämättä ole kaikki samanlaisia, vaan niihin on voitu tehdä pienempiä muutoksia tai ne voivat olla täysin erilaisia. Laitteen asetukset koostuvat useista eri osa-alueista, kuten fyysisen ohjainlaitteen rekisterialueiden määrityksistä, käyttöliittymäsivujen kuvauksesta ja muista laitteen toimintaan vaikuttavista tekijöistä.


KUVIO 16. Fyysinen ohjainlaite

Laitteen kaksi tärkeintä osa-aluetta ovat rekisterialue- ja käyttöliittymäasetukset. Rekisterialuemääritysten avulla samasta fyysisestä ohjainlaitteesta (Kuvio 16) on mahdollista tuottaa toiminnaltaan erilaisia vaakaohjaimia. Käyttöliittymäasetuksien avulla on mahdollista suunnitella projekti ja asiakaskohtaisesti yksilöllinen käyttöliittymä (Kuvio 17) kulloinkin esiintyvien tarpeiden mukaisesti.


KUVIO 17. Paneeli PC ja siinä ajettava ohjainlaitteen käyttöliittymä

## 6.3 Sovelluskehittimen toiminnallinen rakenne

Sovelluskehittimen toiminta voidaan jakaa kahteen osaan: suunnittelu ja projektinhallinta. Suunnitteluosan tarkoituksena on tukea projektinhallintaa, ja siellä toteutetaan kaikki rekisterialuemallit, joista yksittäisen laitteen rekisterialue koostuu. Näiden kahden osan toiminta on eriytetty toisistaan, eikä niitä voida käyttää yhtä aikaa.

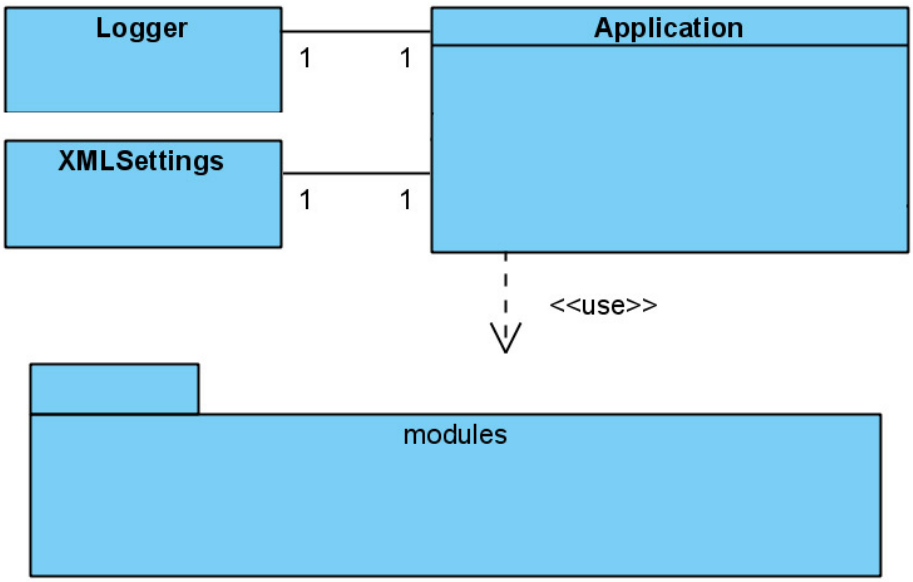
Suunnitteluosan päätarkoituksena on luoda ja muokata erilaisia rekisterialuemalleja, joita hyväksikäytetään suunniteltaessa laitteiden ominaisuuksia ja kokoonpanoa. Kaikista standardeista laitteista toteutetaan yleiset mallit, joita hyödyntäen projektin suunnittelija voivat nopeasti luoda tarvitsemiaan laitteita ilman yksityiskohtaista määrittelyä tai yksityiskohtaisempaa tietämystä laitteen teknisistä yksityiskohdista.

Projektinhallinnan käyttö alkaa aina uuden projektin luomisella tai vanhan avaamisella. Tämän jälkeen on mahdollista luoda uusia laitteita tai muokata jo olemassa olevia. Laitteiden ominaisuuksien muokkaamiseen on käytettävissä joukko hallintatyökaluja. Tärkeimmät näistä työkaluista ovat rekisteri- ja käyttöliittymäeditori. Rekisterieditorilla hallitaan ohjainlaitteen rekisterialueita, ja sen avulla asettaan ja luetaan ohjainlaitelaitekohtaisia rekisteriarvoja suoraan laitteelta tai talletetaan ne myöhempää käyttöä varten. Käyttöliittymäeditori mahdollistaa omien käyttöliittymien suunnittelun graafisella työkalulla, jolla asetetaan kaikki tarvittavat komponentit kohdalleen sekä sidotaan komponenttien toiminta ohjainlaitteen rekisterialueisiin. Muita työkaluja ovat esimerkiksi asetustiedostojen lähettämisen ja vastaanottamisen mahdollistava työkalu sekä ohjainlaitteen tietoliikenneyhteyksien asettamiseen tarkoitettu työkalu.

## 6.4 Sovelluskehittimen tekninen rakenne

### 6.4.1 Päätason rakenne

Sovelluskehittimen perusrakenne koostuu pääohjelmana toimivasta Application-luokasta ja sen sisällä olevista moduuleista, sekä XMLSettings- ja Logger-luokista (Kuvio 18). Kehittimen käynnistyksessä pääohjelma lataa ennalta määritellyt moduulit ja suorittaa ohjelman asetusten alustamisen. Kaikki sovelluskehittimessä käytetyt, mahdollisesti muuttuvat, asetukset on tallennettu XML-tiedostoihin, joita käsitellään XMLSettings-luokan avulla. Ohjelman käytön yhteydessä ilmaantuvat poikkeukset, viestit ja virheet käsitellään Logger-luokassa, joka ilmoittaa niistä käyttäjälle käyttöliittymän kautta.



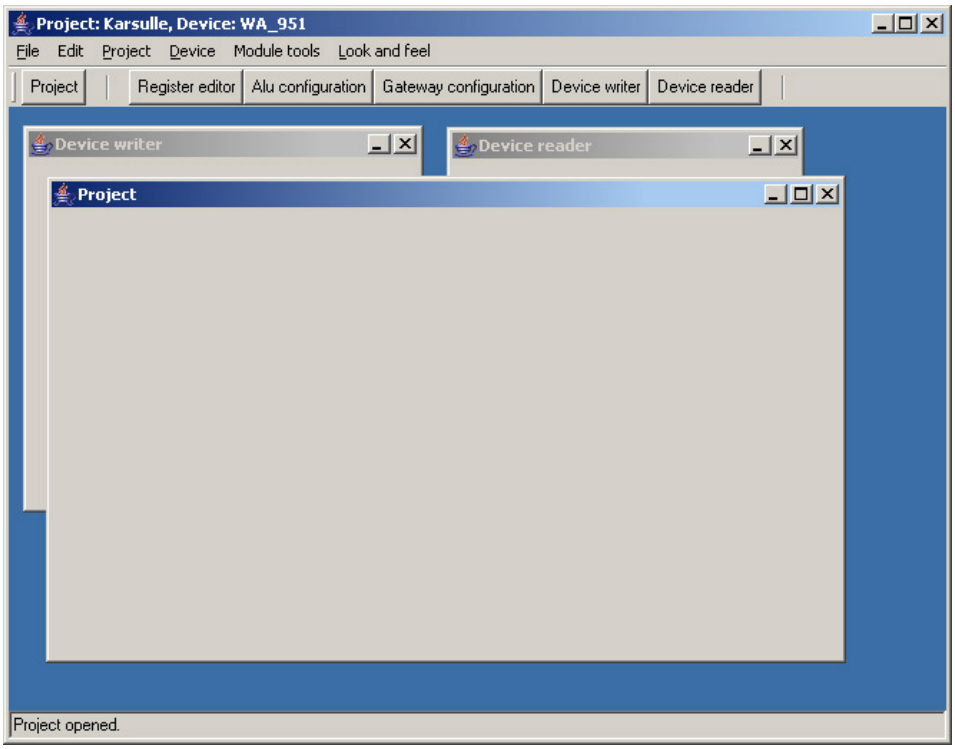

KUVIO 18. Sovelluskehittimen päätason rakenne

Application, Logger, ja XMLSettings luokat ovat kaikki toteutettu singleton-suunnittelumallin mukaisesti. Mallissa luokasta ei luoda kuin yksi ilmentymä ja tämä tallennetaan staattiseksi muuttujaksi ohjelman sisälle. Luokan toimintoja tarvitseva ohjelmanosa pyytää luokalta ilmentymää, joka ensimmäisellä kerralla luodaan ja seuraavilla kerroilla palautetaan. Singleton-malli mahdollistaa kaikkien kolmen luokan ilmentymien käytön mistä tahansa sovelluskehittimen osasta ilman niiden välittämistä.

### 6.4.2 Pääohjelma ja moduulit

Application-luokka on periytetty Swing-komponenttikirjaston JFrame-ikkunaluokasta, joka tarjoaa kaikki graafisen sovelluksen toteuttamiseen vaadittavat perusominaisuudet. Luokka antaa sovellukselle näkyvät raamit, otsikon sekä yläpaneelin, jossa on ohjelman pienentämisen, suurentamisen ja sulkemisen toteuttavat painikkeet. JFramessa olevalle näkymättömälle tasolle on mahdollista lisätä mitä tahansa Javan käyttöliittymäkomponentteja, kuten kuviossa 19 ylhäällä olevat painikkeet ja alareunassa sijaitseva tilarivi. Näiden lisäksi JFramen perusominaisuuksiin kuuluu ylävalikosta (JMenuBar) huolehtiminen. Näkyvien graafisten ominaisuuksien lisäksi JFrame-luokka toimii kuuntelijana käyttöjärjestelmästä

virtuaalikoneen kautta tuleville sekä omille ohjelman lopetus-, sulku- ja pienennystapahtumille.




KUVIO 19. JFrame luokasta periytetty Application-luokan ilmentymä

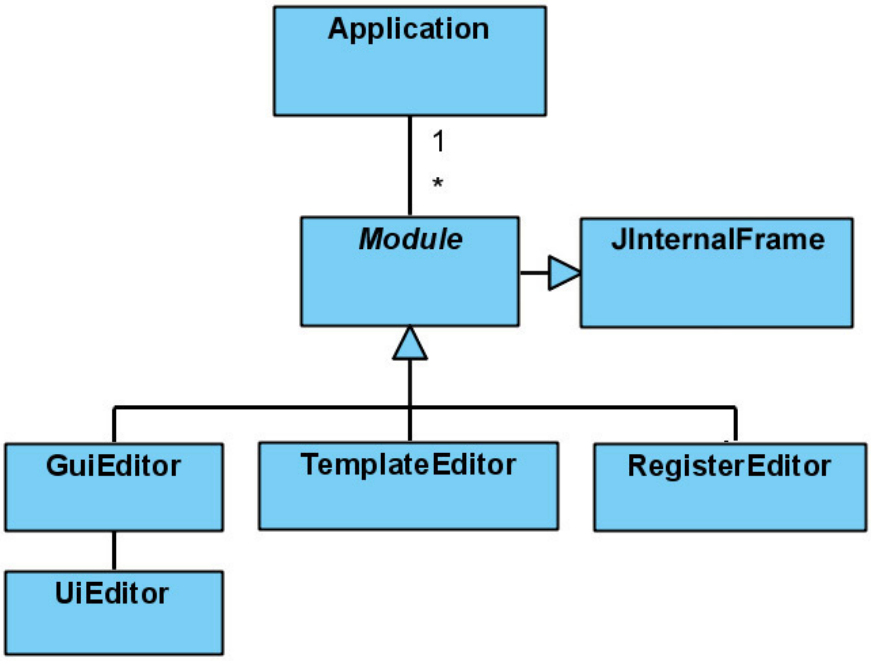
JFrame-luokan ominaisuudet mahdollistaisivat käyttöliittymäkomponenttien asettamisen suoraan Application-luokan päälle, mutta käytännön työssä päädyttiin ratkaisuun, jossa käyttöliittymä jaettiin kahteen osaan. Ensimmäiseen osaan kuuluvat kuviossa 19 näkyvät ylävalikko ja sen alla sijaitseva moduulien käynnistyspainikerivi sekä alareunan tilarivi. Toisen osan muodostaa kuviossa 19 keskellä sijaitseva tila moduuleita varten.


KUVIO 20. Moduulien välilehtitoteutus

Pyrkimys selkeään käyttöliittymään johti sen kahtiajakamiseen. Alusta lähtien moduulien toiminnot haluttiin eriyttää toisistaan niin ohjelmakoodien kuin käyttö-

liittymien osalta. Toteutusvaiheessa ensimmäisenä käyttöliittymävaihtoehtona oli moduulien esittäminen kuviossa 20 näkyvillä välilehdillä. Välilehtien hyvänä puolena on selkeä ulkonäkö ja helppokäyttöisyys. Huonona puolena on, ettei välilehtisivuja voi olla auki kuin yksi kerrallaan. Tästä johtuen toteutettiin oma ratkaisu, jossa otettiin mallia välilehtien tavasta toimia. Ratkaisussa välilehtien valintalehdet korvattiin kuvion 19 yläosassa näkyvillä painikkeilla ja sisältöosa ohjelman sisäisillä ikkunoilla (JInternalFrame).

Käyttöliittymämoduulit ovat itsenäisiä ohjelman osia, joiden kautta kaikki sovelluskehittimen toiminnot suoritetaan. Jokainen moduuli on periytetty kuvion 21 mukaisesti abstraktista Module-luokasta, joka on periytetty JInternalFrame-luokasta. Module-luokkaan on ylikirjoitettu ikkunan sulkemiseen ja kokoon vaikuttavat metodit, koska oletustoteutuksessa ikkunan sulkeminen lopettaa moduulin toiminnan kokonaan ja muutettaessa pääohjelman kokoa täytyy myös moduuleiden koon muuttua samassa suhteessa.



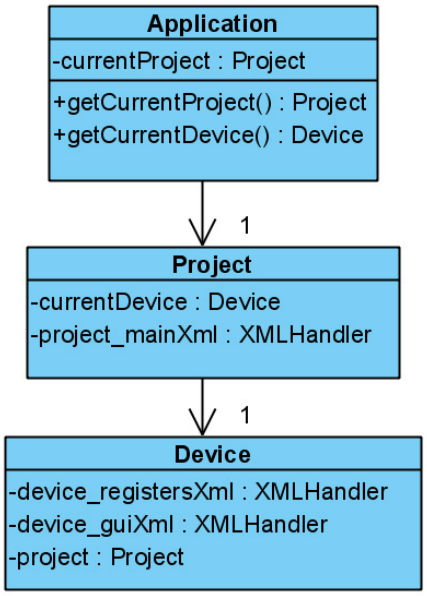

KUVIO 21. Moduulien rakenne ja yhteydet

Moduuli-ikkunat (Kuviossa 19 project, Device writer ja Device reader) avataan automaattisesti Application-luokassa niille varatulle tasolle. Taso on toteutettu JDesktopPane-luokalla, joka on erikoistettu muoto oletuksena JFrame-luokassa

sijaitsevasta käyttöliittymäkomponenttien alustana toimivasta tasosta. JDesktop-Pane-tason erikoisuutena on, että sen sisälle voidaan lisätä komponenttien lisäksi haluttu määrä sisäisiä ikkunoita (JInternalFrame). Sisäiset ikkunat näyttävät JFrame-luokan ilmentymiltä ja niiden toiminta on hyvin samankaltainen. Niiden käyttö on mahdollista JFrame-luokan ominaisuudella, jonka avulla oletus taso voidaan korvata JDesktopPane-tasolla.

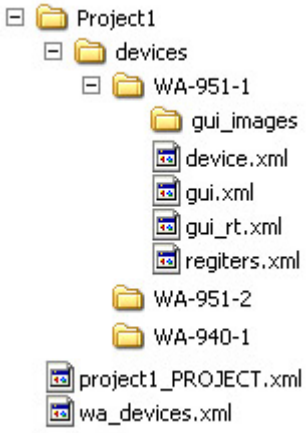
## 6.5 Projekti ja sen laitteet

Sovelluskehittimen toteutus käynnistettiin, kun sillä toteutettavan ohjainlaitteen käyttöliittymästä oli olemassa jotenkin toimiva versio. Tämän takia kehittimen suunnittelussa otettiin huomioon vain yksittäisen laitteen rakentamisessa vaadittavat toiminnallisuudet. Kehityksen kuluessa kävi kuitenkin ilmi, että ainoastaan yksittäisen laitteen hallintaan keskittyvällä sovelluskehittimellä voi olla hankalaa toteuttaa useamman laitteen sisältäviä projekteja. Tästä syystä ohjelman rakennetta päätettiin muuttaa. Lopputuloksena syntyi kuviossa 22 esitelty rakenne, jossa projekteja voi olla auki vain yksi kerrallaan. Yksi projekti voi sisältää monta laitetta, mutta vastaavasti laitteita voi olla auki vain yksi.




KUVIO 22. Projekti, laite ja niiden yhteydet

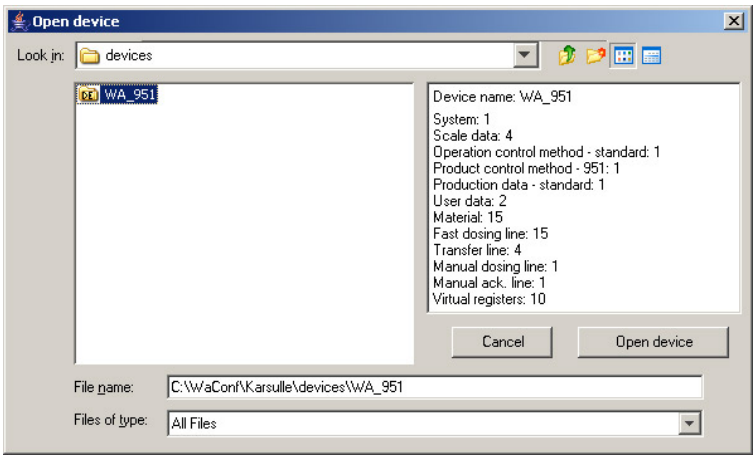
Käytännön projektia vastaa ohjelmakoodissa Project-luokka. Se sisältää tiedot projektin nimestä ja hakemistosta, jonne projektikohtaiset XML-tiedostot ja laitteet tallennetaan. Luokalle on toteutettu toiminnot projektin luomiselle, avaukselle, tallennukselle sekä sulkemiselle, ja se sisältää myös niiden yhteydessä käytetyt käyttöliittymädialogit.




KUVIO 23. Projektin ja laitteiden tiedostorakenne

Projektista luodaan kuvion 23 mukaisesti kaksi XML-tiedostoa ja yksi hakemisto laitteita varten. Tiedostot sisältävät projektin yksilöimisen kannalta tärkeitä tietoja sekä kaikki projektille kuuluvat laitteet.

Fyysistä laitetta kuvaava Device-luokka on aina Project-luokan, sisällä ja sitä voidaan käsitellä ainoastaan rajapintametodin kautta. Device-luokalle on toteutettu lähes samat perusominaisuudet kuin Project-luokalla mutta sillä erolla, että se käsittelee useampia XML-rakenteita (Kuvio 23, WA-951-1 hakemiston alla olevat tiedostot). Tärkeimmät näistä rakenteista sisältävät laitteen käyttöliittymän ja sen rekisterialueiden kokoonpanon.




KUVIO 24. Laitteen aukaisu dialogi

Project- ja Device-luokkien sisältämät dialogit toteutettiin suoraan luokkien sisälle, jotta ne voivat käsitellä tarvittavia XML-tietorakenteita ilman rajapintametodeita. Javan perusdialogien toimintoja muokattiin siten, että ainoastaan projekti tai laite XML-tiedostoja sisältäviä hakemistoja voidaan avata. Javan JDialog-luokan ominaisuudet antavat mahdollisuuden muuttaa näytettävien hakemistojen ikoneja asetettujen määritysten mukaisesti. Toinen dialogeissa käytettävä ominaisuus on kuviossa 24 esitetty projekti ja laitekohtaisten tietojen esikatselu ennen avaustoimenpidettä. Toiminnot helpottavat projekti- ja laitehakemistojen löytämistä sekä niiden sisältöjen tutkimista.

## 6.6 Tietojen tallennus

### 6.6.1 Tietojen varastoiminen XML-rakenteeseen

Sovelluskehittimen toteutuksen alkuvaiheessa suunniteltiin ja testattiin erilaisia vaihtoehtoja tietojen varastoimiseen. Ensimmäiseksi testattiin Javan tarjoamaa ResourceBundle-luokan tapaa määritellä ja tallettaa hierarkista dataa tiedostoihin. Luokkaa käyttäen tallennetut tiedot saadaan nopeasti käsittelyyn kuviossa 25 esiteltyjen avain arvo parien perusteella. Toteutus on erittäin hyvä ja selkeä, kun käsiteltävä tieto on yksinkertaista, mutta monimutkaisien tietojen tallennukseen rakenne ei ole riittävän monipuolinen.

```
project.name = project1
project.devices = 5
project.devices.device1 = wa-951
project.devices.device2 = wa-940
```

KUVIO 25. Javan ResourceBundle-luokan ymmärtämää dataa

Hylätyn ratkaisuvaihtoehdon korvaajaksi valittiin XML-rakenne. Se on tarkoitettu datan ja varsinkin sen rakenteen tallennukseen standardilla tavalla. XML on erinomainen tapa tallettaa tietoa, koska sitä ymmärretään ja voidaan tarvittaessa muokata ilman sen tuottamiseen käytettyä alkuperäistä ohjelmaa.

Valintaan vaikuttivat myös Javalle saatavilla olevat XML-ohjelmistorajapinnat, joita käyttämällä datan rakenteen lukemiseen ja kirjoittamiseen tarvittavia ohjelmanosia ei tarvinnut itse toteuttaa. Ohjelmistorajapinnoista valittiin käyttöön JDOM-kirjasto, koska sen toiminta perustuu koko XML-rakenteen tallettamiseen muistiin. Toteutustapa on hyödyllinen, koska tietojen täytyy olla käytettävissä ja muokattavissa koko ohjelman käynnissä oloajan.

JDOM-rakenne toimii perustana kaikelle tietojen käsittelylle sovelluskehittimen sisäisessä toiminnassa. Sen avulla välitetään kokonaisia XML-dokumentteja (Document-luokka) tai osia niistä (Element-luokka) ohjelman eri osien välillä.

JDOM:n vahvuutena on, että puurakenteen yksittäisen elementin sisältöön tai rakenteeseen tehdyt muutokset ovat heti voimassa koko XML-dokumentissa.

JDOM:n käytössä ei esiintynyt lainkaan teknisiä ongelmia. Rakenteen toteutustavan ymmärtäminen on kuitenkin tärkeää varsinkin onnistuneen kopioinnin ja poiston yhteydessä. Elementtiä kopioidessa täytyy ottaa huomioon siinä olevat viittaukset alkuperäisen puurakenteen isäntä- ja lapsielementteihin. Ilman Javan kloonaus toimintoa (clone), elementin vanhat viittaukset jäävät voimaan eikä puurakenne ole eheä. Pahimmassa tapauksessa kopioinnin lähde ja kohde sijaitsevat samassa XML-rakenteessa ja tällöin puuhun syntyy silmukoita. Elementin poistossa on myös huolehdittava vastaavista viittauksista. Edelleen tulee huolehtia siitä, että kaikki viittaukset kyseiseen elementtiin katkaistaan poiston yhteydessä.

### 6.6.2 XML-datan hallinta

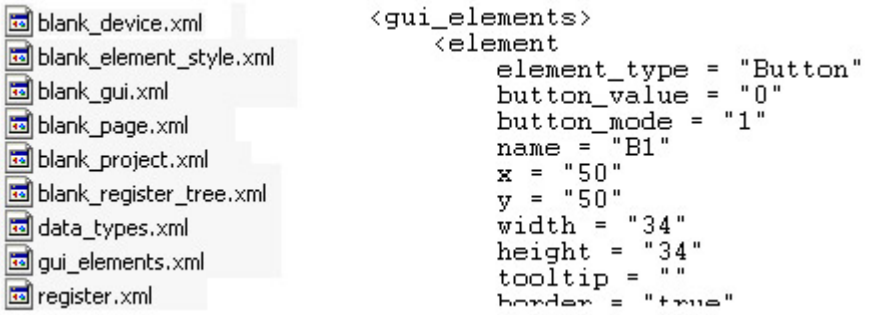
Sovelluskehittimessä käytettyjen tietojen hallinta voidaan jakaa kahteen osaan. Ensimmäiseen kuuluvat kaikki kehittimellä tuotettavat tiedot ja toiseen ohjelman toimintaan ja sillä tuotettavien projektien ja laitteiden mallintamiseen vaadittavat määritykset. Eri osien XML-tiedostojen hallinta on jaettu XMLSettings- ja XMLHandler-luokille.

Kuviossa 23 esitettyjen projekti- ja laitetiedostoista huolehtii XMLHandler-luokka. Jokaista tiedostoa kohden on yksi luokan ilmentymä ja tämän avulla tiedot luetaan ja kirjoitetaan levylle. Näiden operaatioiden lisäksi XMLHandler-luokalla on tärkeä tehtävä tiedon varastoimisessa ja sen eheyden tarkkailijana ohjelman käytön aikana. Kaikki kyseisestä XML-tiedostosta kiinnostuneet sovellusosat ovat siihen yhteydessä vain XMLHandler-luokan rajapintojen kautta.

XMLSettings-luokka huolehtii kaikkien sovelluskehittimen toimintaan liittyvien XML-tiedostojen lataamisen ja tarjoaa rajapinnat tietojen käyttöön. Tiedostoissa on määritelty:

- ohjelman sisäiset asetukset kuten projektien tallennus hakemisto,
- ohjelman käynnistyksessä ladattavat moduulit ja
- projekti ja laite määrityksien rakennemallit.

Sovelluskehitin on toteutettu siten, että kaikki sillä tuotettu XML-data on peräisin rakennemallitiedostoista (Kuvio 26). Tästä johtuen dataan voidaan tehdä muutoksia ja lisäyksiä ilman, että sovelluskehitinohjelmaa joudutaan kääntämään uudelleen. Mallitiedostot mahdollistavat myös sovelluskehittimen joustavamman kehityksen tulevaisuudessa.




KUVIO 26. Muutamia rakennemalleja sekä ote gui_elements XML tiedostosta

## 6.7 Sovelluskehittimen käyttöliittymäkomponentit

### 6.7.1 Käyttöliittymien rakenne ja tehtävä

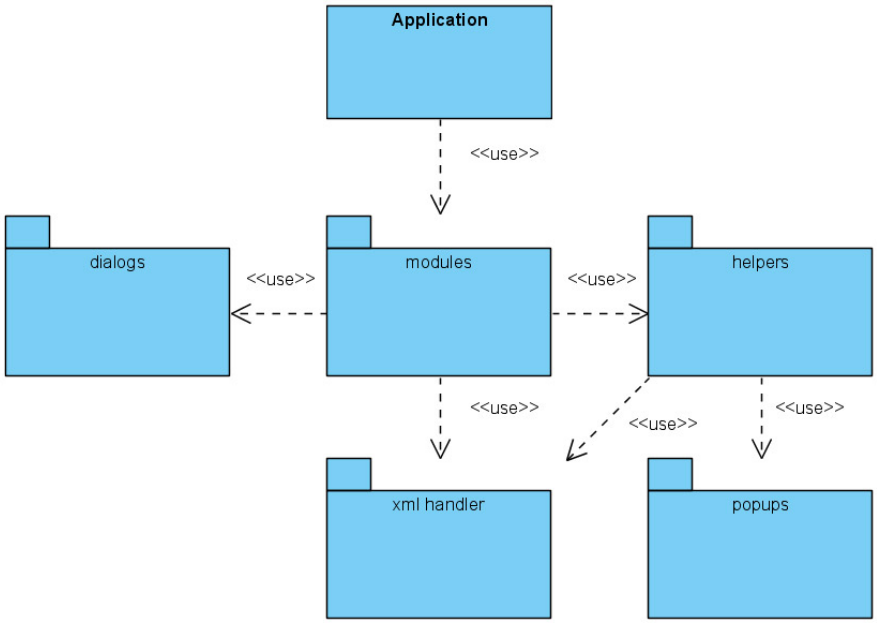
Sovelluskehittimen tehtävänä on tuottaa ennalta määriteltyjä XML-rakenteita vaakaohjaimen alustamista ja käyttöä varten. Toteutetun kehittimen periaatteena on, ettei XML-dataa tarvitse käsin kirjoittaa, vaan kaikki rakenteet luodaan käyttöliittymäkomponenttien avulla.

Komponenttien pohjaksi valittiin Swing-grafiikkakirjasto AWT:n sijaan, koska Swing tarjoaa laajemman komponenttivalikoiman sekä MVC-mallin mukaisen

toiminnan. Käyttöliittymissä käytetään runsaasti Swingin peruskomponentteja, kuten nappeja, valikoita ja tekstilaatikoita, käynnistämään, esittämään ja ohjaamaan suoritettavia tapahtumia. Näiden peruskomponenttien lisäksi sovelluskehittimeen on toteutettu joukko niistä erikoistettuja komponentteja.

Erikoistettujen komponenttien päätehtävänä on esittää haluttu XML-data graafisesti ja luoda käyttöliittymä sen muokkaamiseen. Komponentit sisältyvät kuvion 27 helpers-pakettiin, ja niitä hyväksikäyttävät ainoastaan sovelluskehittimen eri moduulien käyttöliittymät. Joissain tapauksissa erikoistetut komponentit käyttävät yhteiskäyttöisiä ponnahdusvalikoita, jotka löytyvät popups-paketista (Kuvio 27).

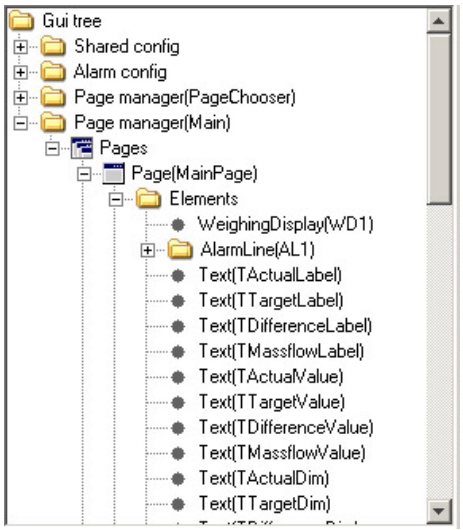
Helpers-paketin lisäksi moduuleilla on käytössä joukko käyttöliittymädialogeja, joita ne käyttävät erilaisten toimintojen avaamiseen ja valintojen suorittamiseen.




KUVIO 27. Sovelluskehittimen rakenne käyttöliittymien näkökulmasta

### 6.7.2 Erikoistettu puukomponentti

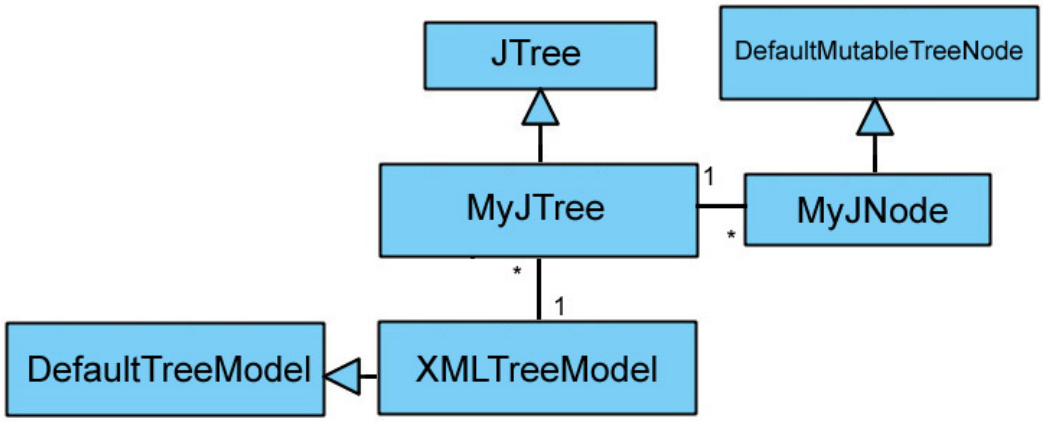
XML:n perusrakennetta voidaan kätevimmin kuvata puurakenteena, ja tästä johtuen ensimmäinen monimutkaisempaan käyttöön toteutettu komponentti periytettiin (Kuvio 29) Swingin JTree-luokasta. Kuviossa 28 esitetty MyJTree-luokan ilmentymä, myöhemmin MyJTree, näyttää sille asetetusta XML-rakenteesta ainoastaan elementtitason rakenteen ja jättää arvot ja attribuutit näkymättömiksi.




KUVIO 28. Puukomponentin käyttöliittymänäkymä

MyJTree käyttää hyväkseen kuviossa 29 näkyvää MyJNode-luokkaa, jonka tehtävänä on yksittäisten XML-elementtien kuvaaminen. Kuviossa 28 näkyvä puurakenne syntyy, kun MyJTree:lle annetaan MVC-periaatteen mukaisesti datan kuvaava malli.

Mallina toimii XMLTreeModel-luokka (Kuvio 29) toteuttaen rajapinnat, joiden avulla puukomponentti pystyy hyödyntämään sille osoitetun XML-datan. Malli käyttää XML-rakennetta JDOM-ohjelmistorajapinnan avulla, jonka se saa käyttöönsä luonnin yhteydessä välitettävänä JDOM-Elementtinä.




KUVIO 29. Puukomponentti luokkakaavio

XMLTreeModel-luokan toiminta perustuu siihen, että se välittää tietoa MyJTree-komponentille paketoimalla käsittelyssä olevan JDOM-Elementin MyJNode-luokan sisälle. Paketointi on välttämätöntä, koska muussa tapauksessa JTree-luokasta periytetty MyJTree ei osaa käsitellä ja näyttää graafisia elementtejä oikein.

Ilman MyJNode-luokan toteuttamista ja JDOM-Elementin sisällyttämistä sen sisälle, XML-datassa tapahtuvien muutosten esittäminen MyJTree-komponentilla ei olisi mahdollista. Tässä tapauksessa MyJTree saisi ainoastaan luontihetkellä voimassa olevan XML-rakenteen haltuunsa. Käyttöliittymäpuun sisältäessä alkuperäiset JDOM-Elementit, taustalla olevaa XML-rakennetta on mahdollista muuttaa puuta muokkaamalla ja rakenteessa tapahtuvat muutokset voidaan lukea käytössä olevista JDOM-Elementeistä.

MyJTree-käyttöliittymäkomponentille toteutettiin yksinkertaiset rajapinnat, joita hyödyntäen on mahdollista:

- poistaa haluttu tai valittuna oleva elementti
- siirtää elementti uuteen paikkaan lähde ja kohde elementtien avulla
- saada käyttöön valittuna oleva elementti
- asettaa valituksi tietty elementti
- etsiä puusta haluttua elementtiä.

MyJTree-komponentin graafisen ulkoasun muuttaminen ja tiedon rajaaminen on myös mahdollista. Filter-luokan avulla määritellään puusta näytettävät elementit nimen perusteella. Ominaisuus on hyödyllinen tilanteissa, joissa kaikkia puun elementtejä ei haluta näyttää. TreeRenderer-luokan avulla käyttöliittymässä näkyviä elementtien nimiä tai niiden edessä olevia kuvakkeita voidaan muuttaa XML-datassa olevien tietojen perusteella (Kuvio28). Kuvatut ominaisuudet eivät ole täysin välttämättömiä, mutta niiden avulla saadaan parannettua sovelluskehittimen käytettävyyttä.

### 6.7.3 Erikoistettu taulukomponentti

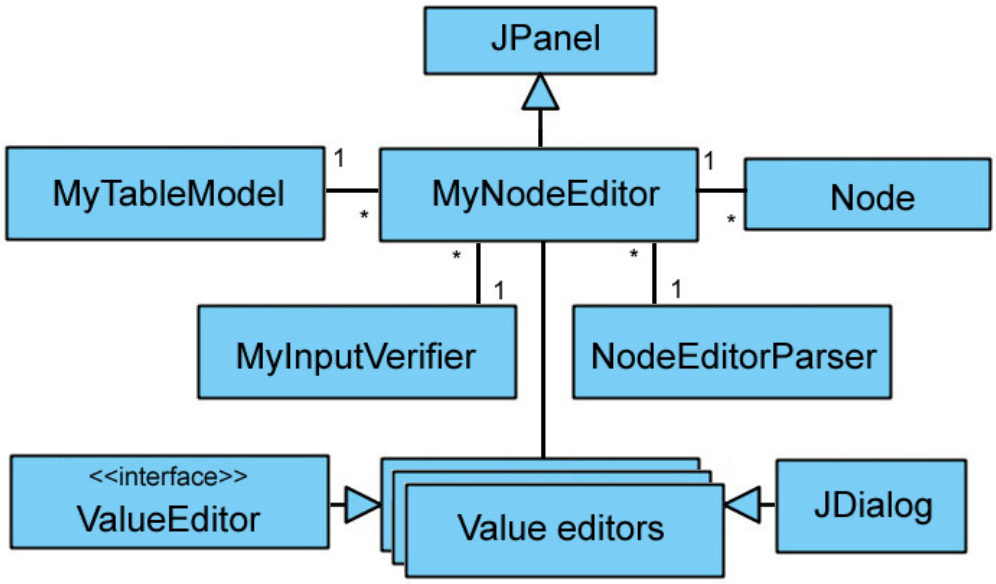
Puukomponentin keskittyessä ainoastaan XML-rakenteen esittämiseen ja muokkaamisen, taulukomponentilla hallitaan yksittäisen elementin ja sen attribuuttien arvoja. Komponentin avulla nähdään komponentille asetetun elementin kaikki arvot yhdellä silmäyksellä (Kuvio 30) ja niitä on myös mahdollista muokata.

| Title | Value |
|---|---|
| name | l2_feed_cut_mode |
| address | 8754 |
| data_type | H_USHORT |
| data_length | 1 |
| value | Internal |
| access | RW |
| description | line data: Feed cut operat |

KUVIO 30. Taulukomponentin käyttöliittymänäkymä

Taulukomponentti koostuu otsikkorivistä ja sen jälkeen tulevista datariveistä. Jokaisella datarivillä on nimi ja arvo kentät sekä painike, jota painamalla avautuu ValueEditor-rajapinnan toteuttama dialogi (Kuvio 30). Arvoa muokataan oletuksena tekstikenttäkomponentilla (JTextField), mutta se voidaan myös määritellä alasvetovalikoksi (JComboBox).

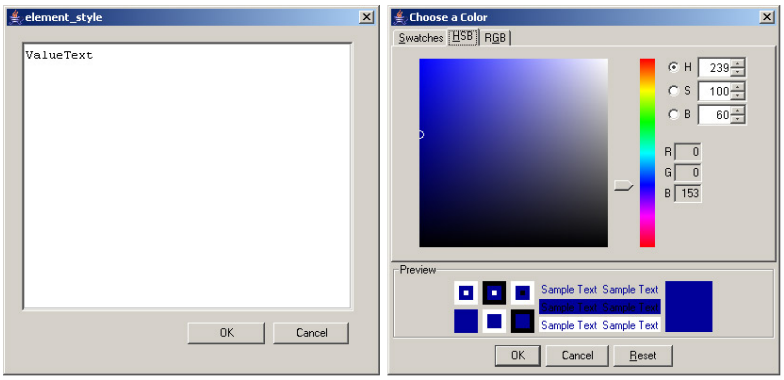
Taulukomponentti (Kuvio 31, MyNodeEditor) ei ole puunkomponentin tapaan periytetty Swingin JTable:sta vaan JPanel-luokasta. JTable on luokan sisällä yksittäisenä muuttujana, joka on tämän jälkeen lisätty taulukkokomponentin päälle.




KUVIO 31. Taulukomponentti luokkakaavio

JTable:ssa esitettävästä data sisällöstä vastaa MyTableModel-luokka, joka on suoraan kytköksissä MyNodeEditor-luokkaan. Taulukomponentin saadessa JDOM-Elementti käytettäväkseen suoritetaan kaksivaiheinen tarkastelu. Ensimmäisessä vaiheessa käydään läpi kaikki elementin attribuutit, ja toisessa käsitellään sen arvo. Käsittelyn tuloksena luodaan malliin sijoitettavia Node-luokan ilmentymiä (Kuvio 30), jotka sisältävät JDOM-Elementin tai –Attribuutin sekä arvoa muokkaavan käyttöliittymäkomponentin.

Taulukomponentin perusluokkien lisäksi kokonaisuuteen kuuluu joukko sitä tukevia luokkia. NodeEditorParser tarjoaa mahdollisuudet rajoittaa taulukomponentissa näkyviä JDOM-Elementin attribuutteja ja arvoa niiden nimien perusteella. Samoin attribuutin tai elementin nimen perusteella voidaan määritellä, onko kyseistä arvoa mahdollista muokata (Kuvio 30, harmaat kentät). Samojen nimitietojen perusteella määritellään arvoa muokkaava komponentti (Kuvio 30, JTextField tai JComboBox).

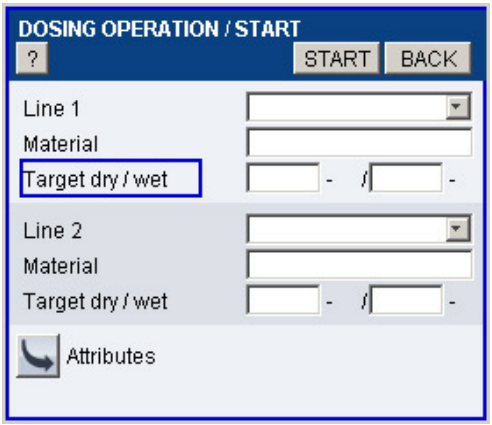



KUVIO 32. Oletus- ja värieditoridialogi

Attribuutin tai elementin nimen perusteella määritellään myös kuviossa 30 jokaisella datarivillä näkyvän painikkeen takaa avautuva arvon muokkauseditori. Muokkauseditorin ideana on, että asetettavia arvoja voidaan helposti asettaa käyttöliittymäkomponenttien avulla, kuten kuviossa 32 värivalinta dialogi. Värikoodien tulee olla oikean muotoisia, joten taulukomponentilla on arvojen tarkastukseen erikoistunut MyInputVerifier-luokka. Sen avulla voidaan määritellä attribuutti- ja elementtikohtaisesti, millaiset arvot ovat hyväksyttyjä.

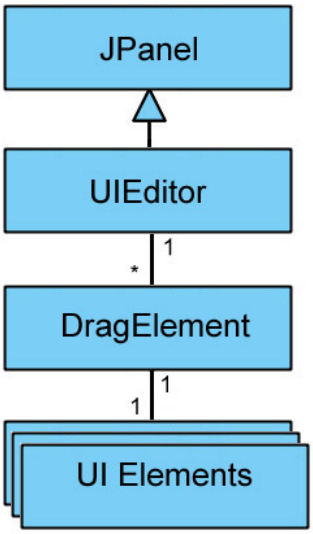
### 6.7.4 Erikoistettu käyttöliittymien suunnittelukomponentti

Suunnittelukomponentti tehtävänä on luoda alusta, jota käyttäen voidaan määritellä käytössä olevien käyttöliittymäelementtien sijaintia ja kokoa ruudulla. Suunnittelukomponentilla toteutettava kokonaisuus pyrkii mahdollisimman tarkasti muistuttamaan todellista käyttöliittymää näkymää ja näin helpottaa käytännön suunnittelutyötä (Kuvio 33).




KUVIO 33. Suunnittelukomponentin käyttöliittymänäkymä

Käyttöliittymien suunnittelukomponenttia ei voida verrata puu- ja taulukomponentin tapaan mihinkään olemassa olevaan Swing-komponenttiin. Komponentin perustana on JPanel-luokka (Kuvio 34), mutta se toimii vain alustana, jolle muut suunnittelussa käytetyt käyttöliittymäelementtien ladotaan.




KUVIO 34. Suunnittelukomponentin luokkakaavio

Suunnittelukomponentilla työstettävien painikkeiden, listojen, paneelien, tekstien, kuvien, linkkien ja tekstikenttien isäntäluokkana toimii kuviossa 34 näkyvä DragElement-luokka. Jokaisesta näkymäelementistä (Kuvio 34, UI Elements) on luotu todellisessa käyttöliittymässä esiintyvän kaltainen toteutus perustuen Swing-komponentteihin. DragElementille välitetään luonnin yhteydessä JDOM-Element, jonka perusteella se valitsee näytettävän toteutuksen ja lisää sen JPanel-luokasta peritylle pinnalle.

DragElement-luokka toteuttaa MouseListener, KeyListener ja FocusListener rajapinnat ja kuuntelee näin itse itseään. MouseListener-rajapinnan avulla toteutettiin elementtien liikuttelu ruudulla, mutta toteutuksessa täytyi ottaa huomioon DragElementin pinnalle lisättyjen näkymäelementtien olemassa olevat ominaisuudet. Hankaluuksia tuotti DragElementin valinta, koska kohdistus tapahtui aina sen päälle lisättyihin näkymäelementteihin. Ongelma saatiin ratkaistua niin sanotun lasipaneeli-(glass panel) tekniikan avulla. Siinä komponentin pinnalle lisätään koko sen alan peittävä paneeli (JPanel), joka asetetaan sitten läpinäkyväksi.

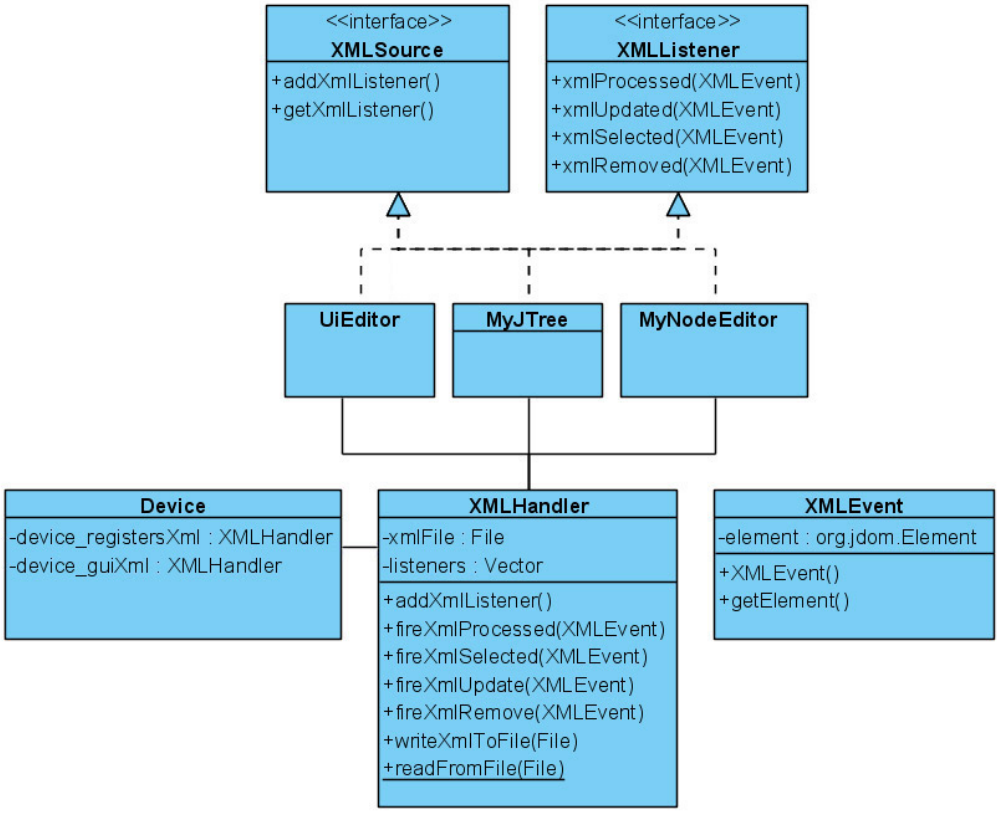
Tämän jälkeen DragElementin MouseListener-rajapinta asetettiin kuuntelemaan vain lasipaneeliin kohdistuvia hiiritapahtumia. Toteutuksen jälkeen kaikki hiiritapahtumat kohdistuivat vain lasipaneeliin ja DragElementin valinta, liikuttelu ja koon määritys voitiin toteuttaa.

KeyListener-rajapinnan avulla toteutettiin komponentin liikuttelu sekä poisto nuoli- ja del-painikkeiden kautta. FocusListener-rajapintaa käytetään toteamaan valinta tapahtumaa, jolloin valitun DragElementin ympärille piirretään reunukset kuvion 33 mukaisesti.

### 6.7.5 Käyttöliittymäkomponenttien ja XML:n välinen yhteys

Sovelluskehittimen toteutuksessa eteen tulleista ongelmista vaikein oli käyttöliittymäkomponenttien ja XML-rakenteen (JDOM) välinen yhteneväisyys. Kehitystyön alkuvaiheessa luoduissa moduuleissa käytettiin vain puu- ja taulukomponentteja. Komponentit tunsivat toinen toisensa ja informoivat muutoksista toisiaan. Huonona puolena toteutuksessa oli, että kaikkien samaa JDOM-rakennetta käyttävien komponenttien tuli tuntea toisensa ja ilmoittaa tapahtuneista muutoksista kaikille.

Kehitystyön edetessä tuli vastaan tilanne, jossa samaa JDOM-rakennetta käyttävien komponenttien määrä nousi niin suureksi, ettei enää ollut mielekästä eikä järkevää olettaa niiden tuntevan toisiaan. Joissain tapauksissa tämä oli jo lähes mahdotonta. Ongelman ratkaisun pohjaksi otettiin tarkkailija-suunnittelumalli. Ratkaisua laajennettiin siten, että alkuperäisessä mallissa käytössä olevan yhdensuuntaisen informaation välittämisen sijaan päädyttiin kahdensuuntaiseen ilmoittamiseen. Tämän lisäksi tapahtuman välityksen yhteydessä lähetetään aina tieto (Kuvio 35, XMLEvent) lähettäjästä sekä tapahtuman kohteena olevasta JDOM-Elementistä.




KUVIO 35. XML:n ja käyttöliittymäkomponenttien välisen yhteyden rakenne

Luvussa 6.6.2 (XML-datan hallinta) ja kuviossa 35 esitetyn XMLHandler-luokan tehtävänä on hallita yhden XML-tiedoston tietoja. Luokka tarjoaa tiedoston levytoiminnot ja muuntaa luetun XML-datan JDOM-puuksi. Näiden lisäksi se ylläpitää listaa kuuntelijoista, jotka ovat kiinnostuneita sen hallitsemasta XML-datasta ja informoi näitä tapahtumista.

Yksittäinen käyttöliittymäkomponentti saa XML:n käyttöönsä toteuttamalla XMLListener- ja XMLSource-rajapinnat. Tämän jälkeen komponentti täytyy rekisteröidä haluttuun XMLHandleriin, ja samalla XMLHandler rekisteröidään käyttöliittymäkomponenttiin, sillä se toteuttaa samat rajapinnat, kuin komponentin on toteutettava.

XMLListener määrittelee neljä kuuntelijoille ilmoitettavaa tapahtumaa ja niitä vastaavaa rajapintaa, jotka liittyvät XML-dokumentin prosessointiin tai siinä olevan elementin päivitykseen, valintaan tai poistoon (Kuvio 35). Jokaista rajapinta-

metodia kutsutaan tapahtuman sattuessa, ja jokainen kuuntelija voi toteuttaa omat toimintonsa kyseiselle tapahtumalle.

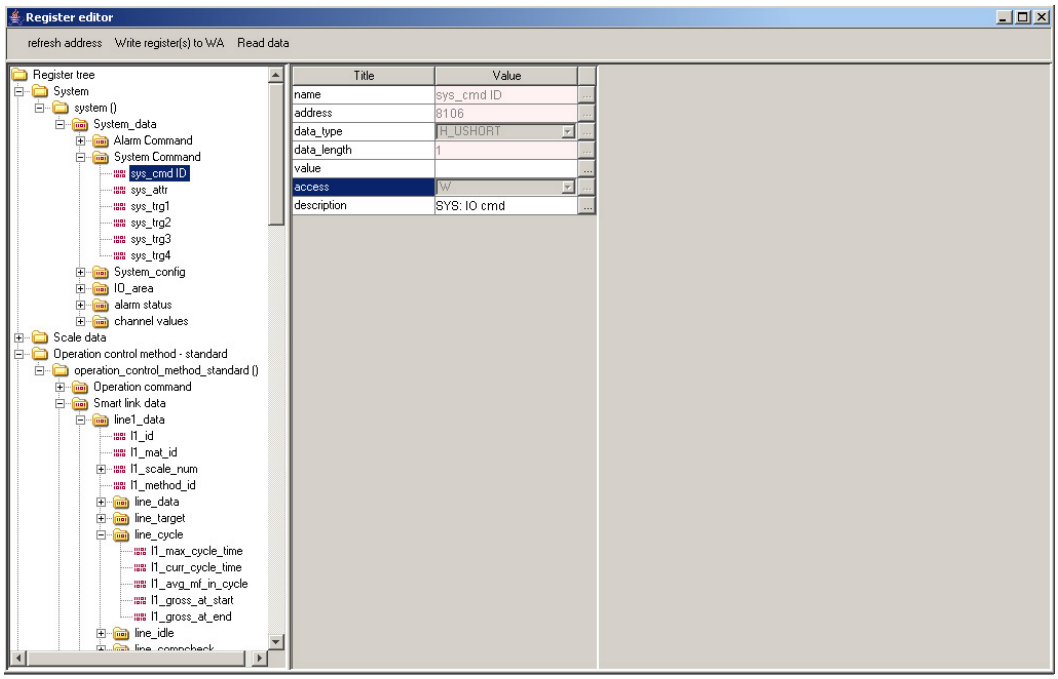
## 6.8 Sovelluskehittimen moduulit

### 6.8.1 Yleistä

Luvussa 6.4.2 esitelty moduulirakenne mahdollistaa sovelluskehittimen eri osia kehityksen pitkällä aikajänteellä. Rakenteen ansiosta tärkeimpien moduuleiden kehitys voitiin suorittaa loppuun ennen siirtymistä kokonaan uusien toteutukseen. Moduulien riippumattomuudella toisistaan saavutetaan myös se etu, että useat kehittäjät voivat luoda omia moduuleita ja näin jakaa kehitystyötä.

### 6.8.2 Rekisterimalli- ja rekisterieditori

Rekisterimalli- ja rekisterieditori ovat toteutukseltaan hyvin lähellä toisiaan. Niillä muokataan samoja rekisterialueita, mutta käyttöajankohta ja tapa ovat erilaisia. Rekisterimallieditorilla suunnitellaan rekisterialueiden rakenteita, kun taas rekisterieditorilla muokataan vain olemassa olevia arvoja.

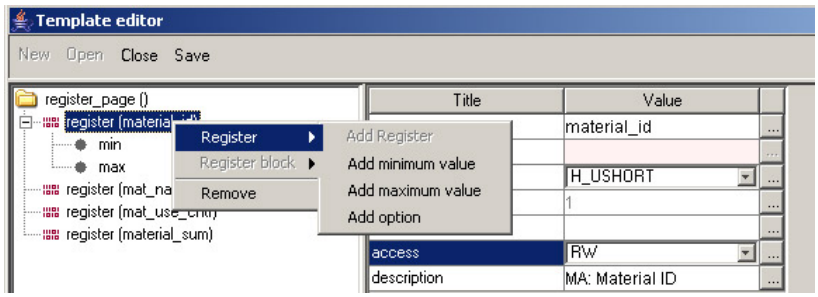



KUVIO 36. Rekisterieditorin käyttöliittymä

Molempien editoreiden käyttöliittymien perustana ovat erikoistetut puu- ja taulukomponentit (Kuvio 36). Puukomponentti toimii XML-rakenteen esittäjänä ja navigointikomponenttina taulukomponentille, joka aktivoituu aina, kun puusta valitaan jokin elementti. Komponenttien välisestä yhteydenpidosta huolehtii muokkauksen kohteena olevan XML-tiedoston XMLHandler-ilmentymä.

Rekisterimallieditorilla suunnittelun mahdollistaa puukomponentin kanssa toimiva ponnahdusvalikko (Kuvio 37) sekä aktiivisena olevat taulukomponentin arvokomponentit. Valikosta voidaan suorittaa haluttu toiminto kuten esimerkiksi rekisterin lisäys, jonka valmiiksi saattamiseen vaaditaan:

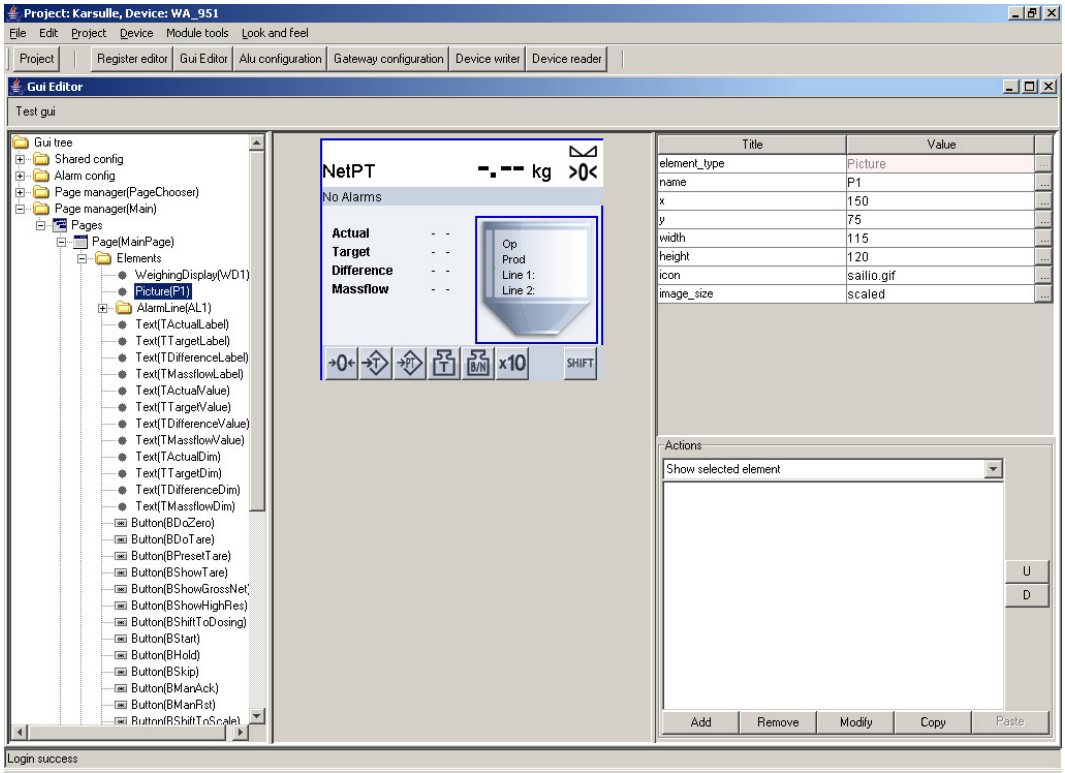
- rekisterimalli JDOM-Elementin haku XMLSettings-ilmentymältä
- rekisterimallin lisäys valittuna olevaan JDOM-Elementtiin
- muutoksesta ilmoittaminen XMLListener rajapinnan kautta
- uuden rekisterin valinnasta ilmoittaminen XMLListener rajapinnan kautta




KUVIO 37. Rekisterieditorin käyttöliittymä

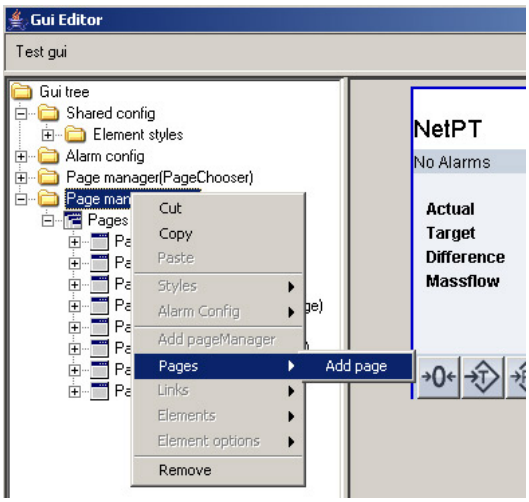
### 6.8.3 Käyttöliittymäeditori

Toteutettu käyttöliittymäeditori on laajin sovelluskehittimessä toimiva moduuli, ja siinä käytetään kaikkia erikoistettuja käyttöliittymäkomponentteja. Moduuli on jaettu pystysuunnassa kolmeen osaan (Kuvio 38). Vasemmalla on puukomponentti, joka toimii käyttöliittymärakenteen navigaattorina. Keskelle on asetettu käyttöliittymien suunnittelukomponentti, jolla määritellään elementtien sijainti sekä koko, ja oikealle taulukomponentti, jonka tehtävänä on esittää ja antaa muokkaus mahdollisuus puu- tai suunnittelukomponentista valituille elementeille.



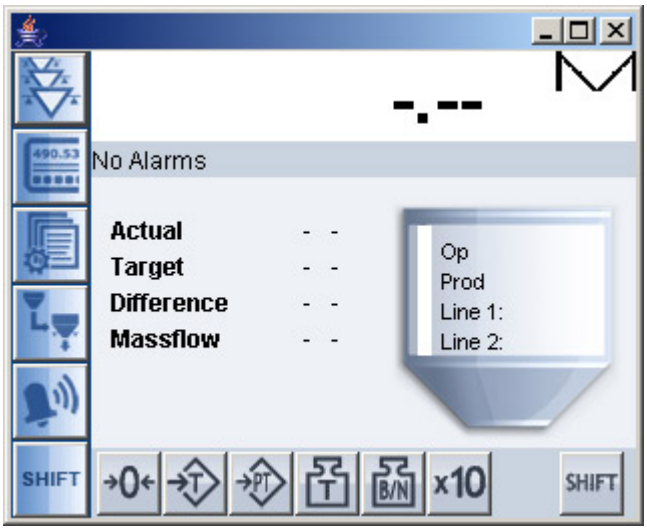

KUVIO 38. Käyttöliittymäeditorin käyttöliittymä

Käyttöliittymäeditorissa tapahtuvat XML-rakenteeseen vaikuttavat lisäys-, poisto- ja siirtotoiminnot suoritetaan puukomponentista esille tulevaa ponnahdusvalikkoa hyödyntäen (Kuvio 39). Valikon toiminnot ovat erilaisia riippuen valittuna olevasta XML-elementistä.




KUVIO 39. Puukomponentista tuleva ponnahdusvalikko

Käyttöliittymäeditoriin on toteutettu mahdollisuus käynnistää muokattavana oleva vaakaohjaimen käyttöliittymä suoraan sovelluskehittimestä. Kuviota 40 ja 38 vertaamalla voidaan sanoa, että käyttöliittymien suunnitteluun erikoistettu komponentti ja lopullinen käyttöliittymä ovat graafisesti hyvin lähellä toisiaan.




KUVIO 40. Vaakaohjaimen käyttöliittymä

# 7 YHTEENVETO

Opinnäytetyön rajaaminen oli vaikeaa, koska siinä käytetyt XML ja Javan grafiikkakirjastotekniikat ovat itsessään valtavan laajoja. Kuvaavaa on, että yli 500 sivua kattaneen lähteeni johdannossa pahoiteltiin kirjan pitäytymistä pelkissä XML:n perusteissa.

Tutkimustyön teoriaosassa käsiteltiin käytännön työssä käytettyjä tekniikoita ja ratkaisumalleja. Tekniikoista esiteltiin XML ja Javan grafiikkakirjastot sekä niiden käyttöä ohjelmistokehityksessä. Näiden lisäksi paneuduttiin sovelluskehitystä nopeuttavien ja helpottavien suunnittelumallien rakenteeseen ja toimintaan.

Teoriaosassa haettiin vastausta kysymykseen: kuinka XML:ää ja Javan grafiikkakirjastoja voidaan hyödyntää toteutettaessa vaakaohjaimen sovelluskehitintä. Ennalta asetettuun kysymykseen vastattiin kattavasti teoriaosassa esiteltyjen aihealueiden pohjalta. Vastaukseen saatiin myös täydennystä käytännön osuudessa eteen tulleiden ongelmien ratkaisuista ja käytetyistä toimintatavoista.

Sovelluskehittimen käytännön toteutuksessa päästiin asetettuihin tavoitteisiin ja kaikki asetetut vaatimukset täytettiin. Lisäksi kehitystyön aikana sovelluskehittimeen toteutettiin ennakolta tunnistamattomia toimintoja, kuten rekisterialueiden päivitys ja tarkkailu suoraan vaakaohjaimelta tai laitetiedostojen tallennus suoraan laitteelle. Kaikkia näitä toimintoja ei ole erikseen esitelty opinnäytetyön sisällössä.

Työssä toteutettu vaakaohjaimen sovelluskehitin on otettu käyttöön Raute Precision Oy:n tuotekehitysosastolla, ja sen jatkokehitys on täydessä käynnissä. Toteutettu toimintojen hajauttaminen moduuleihin osoittautui toimivaksi ja oikeaksi ratkaisuksi jatkokehityksen näkökulmasta. Se antaa mahdollisuuden uusien osien nopeammalle toteuttamiselle sekä helpottaa useiden ohjelmoijien samanaikaisesti tapahtuvan ohjelmistokehityksen.

# 8 LÄHTEET

Erich Gamma, Richard Helm, Ralpj Johanson, John Vlissides, P. 2000.
Design Patterns – Elements of Reusable Object-Oriented Software
Addison-Wesley Longman Publishing Co., Inc. Boston, USA

Brett McLaugh, P. 2001. Java & XML – Tehokäyttäjän opas
Talentum Media

Alexander Nakhimovsky, Tom Myers, P. 2002, Java & XML
IT Press, Helsinki.

Cay S. Horstmann, Gary Cornell, P. 2000. Inside Java2
IT Press, Helsinki.

Satyaraj Pantham, P. 1999. Pure JFC Swing
Ensimmäinen painos, Sams

John Zukowski, P. 2000. Definitive Guide to Swing for Java 2
Toinen pianos, Apress