TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Eki Loippo

RAKENTEISEN INFORMAATION HALLINTAKOMPONENTTI

Työn ohjaaja DI Tony Torp
Työn teettäjä DokuMentori Oy, valvojana Insinööri (AMK) Erno Nieminen
Tampere 2006

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka
Loippo, Eki
Rakenteisen informaation hallintakomponentti
Tutkintotyö 39 sivua + 1 liitesivu
Työn ohjaaja DI Tony Torp
Työn teettäjä Insinööri (AMK) Erno Nieminen
Helmikuu 2006
Hakusanat JavaBean, ohjelmointi, rakenteinen informaatio

TIIVISTELMÄ

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa rakenteisen informaation hallintakomponentti DokuMentori Oy:lle. Työssä keskityttiin komponentin suunnitteluun ja toteutukseen ja se vastaa hyvin ohjelmistoinsinöörin tyypillistä työtehtävää.

Työn tavoitteena oli luoda komponentti, jonka avulla voidaan esittää ja hallita rakenteista informaatiota. Toteutettava komponentti auttaisi sekä ohjelmoijaa että loppukäyttäjää hahmottamaan ja hallitsemaan suuriakin määriä informaatiota. Komponenttia voitaisiin käyttää useassa ohjelmistotuotteessa monin eri tavoin ja sen tulisi olla myös helposti laajennettavissa tulevaisuuden tarpeita silmällä pitäen.

Työn toteutus tuli ajankohtaiseksi DokuMentori Oy:n siirtyessä tuottamaan ohjelmistotuotteita, jotka ovat tekemisissä rakenteisen dokumentaation kanssa. Kunnollista informaation hallinta- ja esityskomponenttia ei Java-ajoympäristöstä löytynyt, jolloin sellainen täytyi tehdä. Toimiva hallintakomponentti on välttämätön loppukäyttäjän kannalta, ja siten työn toteutuksesta tulikin yksi suurimman prioriteetin omaavista projekteista DokuMentori Oy:ssä. Sen vuoksi toteutuksen aikaraja oli tärkeä.

Komponentin suunnittelun jokaisessa vaiheessa tuli ottaa huomioon kaksi käyttäjäkuntaa, jotka ovat loppukäyttäjät ja jatkokehittäjät. Suunnittelussa pyrittiin kiinnittämään erityisen paljon huomiota kummankin käyttäjäkunnan tarpeisiin, laajennettavuuteen ja ylläpidon helppouteen. Komponentti suunniteltiin toimimaan informaation standardista riippumatta, jolloin komponentin käyttömahdollisuudet ovat laajat.

Suunnittelussa käytettiin laajalti hyväksi MVC-ratkaisumallia. Komponentti toteutettiin Java-ohjelmointikielellä Java-ajoaikaympäristön tarjoamia kehityskomponentteja hyväksi käyttäen ja JavaBean-standardin mukaisesti.

Työn tulos on käyttökelpoinen rakenteisen informaation hallintaan tarkoitettu komponentti, jolla on laajat käyttömahdollisuudet. Komponenttia käytetään DokuMentori Oy:n DokuPort™ -sovelluksessa rakenteisen informaation hallintatyökaluna. Komponentti olisi myös tulevaisuudessa tarkoitus tuotteistaa. Komponentille on myös jatkokehitysideoita, eli sen kehitys ei ole loppumassa pitkään aikaan.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering
Loippo, Eki A management component for structured information
Engineering Thesis 39 pages + 1 appendices
Thesis Supervisor Tony Torp (MSc)
Commissioning Company DokuMentori Oy. Supervisor: Erno Nieminen (BSc)
February 2006
Keywords JavaBean, programming, structured information

ABSTRACT

The goal of this bachelor's thesis was to design and implement a management component for structured information, which was placed in order by the joint stock company DokuMentori. The thesis focuses mainly on designing and implementing the component and therefore it is a quite typical task for a software engineer.

The purpose of the component application is to assist in presenting and controlling large amounts of structured information. The component would help both, the end users and the implementing software developers in the task. The component could be used in several software products in many different ways and it should be easily extendible considering the needs of the future.

Once DokuMentori Oy started to produce software products handling structured documentation, a need for the management component arose. Such component did not exist in the default Java Runtime/Development Environment, so the implementation of a new component was necessary. A working management component is essential to the end users and therefore the project became a top priority in DokuMentori Oy's software developing section. Based on those facts, the time limit for the implementation was important.

The two user groups of the component were to be taken into account in every area of the component design. Those two user groups are the end users and the implementing software developers. The design task strived for focusing on the both user group's needs, ease of maintenance and modularity. The component was designed to work regardless of the standard of the information and therefore it has extensive usage possibilities.

The MVC design pattern was used widely in the component's design. The component was programmed using the Java language and implementing some of the Java Development/Runtime Environment's development components. The component was also programmed according to the JavaBean-standard.

The result of this thesis is a useful component application in managing structured information with extensive usage possibilities. At the time, the component is used in DokuMentori Oy's DokuPort™-application as a management component for structured information. The component will possibly be merchandised in the near future. There are also some ideas for continuation development, which means that the component's development will not end in a while.

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

ALKUSANAT

Tätä työtä kirjoitettaessa on oletettu, että lukija on ohjelmistotuotannon tietomäärältään ohjelmistoinsinöörin tasolla ja tuntee entuudestaan Java-ympäristön ja -ohjelmoinnin. Siksi tiettyjä termejä, kuten MVC-ratkaisumalli, ei selitetä tarkemmin.

Työ on kirjoitettu laajasta aihepiiristä, joka on yritetty tarkoin rajata. Tarkoituksena ei ole kertoa toteutetusta spesifikaatioista ja ohjelmakoodista, vaan itse projektista – sen lähtökohdista, ongelmista ja ratkaisuista. Suurimman osan tekstistä onkin tarkoitettu olevan analyyttistä tai sitä tukevaa. Työn tarkoitus on kuvata tyypillistä ohjelmistoinsinöörin työtehtävää ja sen edistymistä.

Haluan kiittää työni ohjaajaa, Tony Torpia, rakentavista kommenteista ja nopeasta toiminnasta. Kiitän myös DokuMentori Oy:n ohjelmistotuotantopäällikkö Erno Niemistä avusta ja vastauksista kysymyksiini sekä mahdollisuudesta toteuttaa tämä työ.

Tampereella 24. huhtikuuta 2006

_____

Eki Loippo

SISÄLLYSLUETTELO

1 JOHDANTO

Insinöörityön tarkoituksena on suunnitella ja toteuttaa rakenteisen informaation hallintakomponentti DokuMentori Oy:lle. DokuMentori Oy on joulukuussa 1998 perustettu dokumentointituotantoon erikoistunut asiantuntijayritys, joka tarjoaa asiakkailleen dokumentoinnin sisällöntuotantoa, palveluntuotantoa ja ohjelmistotuotantoa /2/. DokuMentori Oy:n kehitysosasto tuottaa sekä räätälöityjä että tuotteistettuja ohjelmistoja.

Työn toteutus tuli ajankohtaiseksi, kun DokuMentori Oy siirtyi tuottamaan ohjelmistotuotteita, jotka ovat tekemisissä rakenteisen dokumentaation kanssa. Toimiva hallintakomponentti on välttämätön loppukäyttäjän kannalta, ja siten työn toteutuksesta tulikin yksi suurimman prioriteetin omaavista projekteista.

Työssä toteutettavan sovelluskomponentin avulla voidaan esittää ja hallita rakenteista informaatiota. Komponentti auttaa sekä ohjelmoijaa että loppukäyttäjää hahmottamaan ja hallitsemaan suuriakin määriä informaatiota. Toteutettavaa komponenttia tulee voida käyttää mahdollisimman useassa ohjelmistotuotteessa monin eri tavoin. Komponentin tulisi olla myös helposti laajennettavissa tulevaisuuden tarpeita silmällä pitäen. Sovelluskomponentin toteutus käsittää esitutkimuksen, suunnittelun ja ohjelmoinnin.

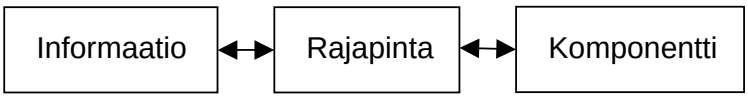
Komponentin ohjelmointi on toteutettu Sun Microsystemsin Java-kielellä. Toteutuksessa käytetään JavaBean-tekniikkaa.

2 RAKENTEINEN INFORMAATIO

Rakenteinen informaatio on laaja käsite, joka on työn kuvaamisen kannalta hyvin tärkeä. Rakenteinen informaatio on tiedon standardista täysin riippumatonta. Se on mitä tahansa informaatiota, jolla on jokin rakenne. Esimerkkejä rakenteisesta informaatiosta ovat muun muassa seuraavat:

- Kirjan sisällysluettelo, joka sisältää kirjan sisällön rakenteisesti esitettynä. Tämä tekee siitä rakenteista informaatiota.
- Tiedostojärjestelmän hakemistorakenne, joka esittää tallennetun informaation rakenteisessa muodossa. Siten sitä usein hallinnoidaankin graafisesti puumuotoisena.
- Rakenteinen dokumentaatio, joka noudattaa jotain tiettyä rakennekuvausta. Tämä rakennekuvaus on merkintäkielestä riippuvainen. Rakenteisia merkintäkieliä ovat esimerkiksi XML ja HTML.

Työssä toteutettu komponentti kehitettiin ensisijaisesti rakenteisen dokumentaation hallinnan apuvälineeksi. Silti komponentti suunniteltiin siten, että sillä voitaisiin hallita mitä tahansa rakenteista informaatiota, tiedon varsinaisesta olemuksesta riippumatta. Tämä standardista vapaa esitysmuoto mahdollisti komponentin käytön moniin erilaisiin tarkoituksiin. Esittääkseen ja hallitakseen tietoa komponentin avulla jatkokehittäjän täytyisi ainoastaan ohjelmoida informaation ja komponentin välille rajapinta. Tämän rajapinnan avulla komponentin toiminnot sidotaan informaatioon. Rajapinnan toiminta on esitettynä alla olevassa kuvassa (kuva 1).



Kuva 1 Komponentin toimintojen sitominen informaatioon rajapinnan avulla.

Koska komponentin kehitys perustui alun perin rakenteisen dokumentaation hallinnointiin ja sen tarpeisiin, on käsitteiden selventäminen tarpeen. Dokumenttien tyypit voidaan karkeasti jakaa kahteen eri ryhmään. Nämä ryhmät ovat

proseduaaliset ja rakenteiset dokumentit. Ohessa kerrotaan, miten nämä kaksi dokumenttityyppiä eroavat toisistaan.

Proseduaaliset dokumentit

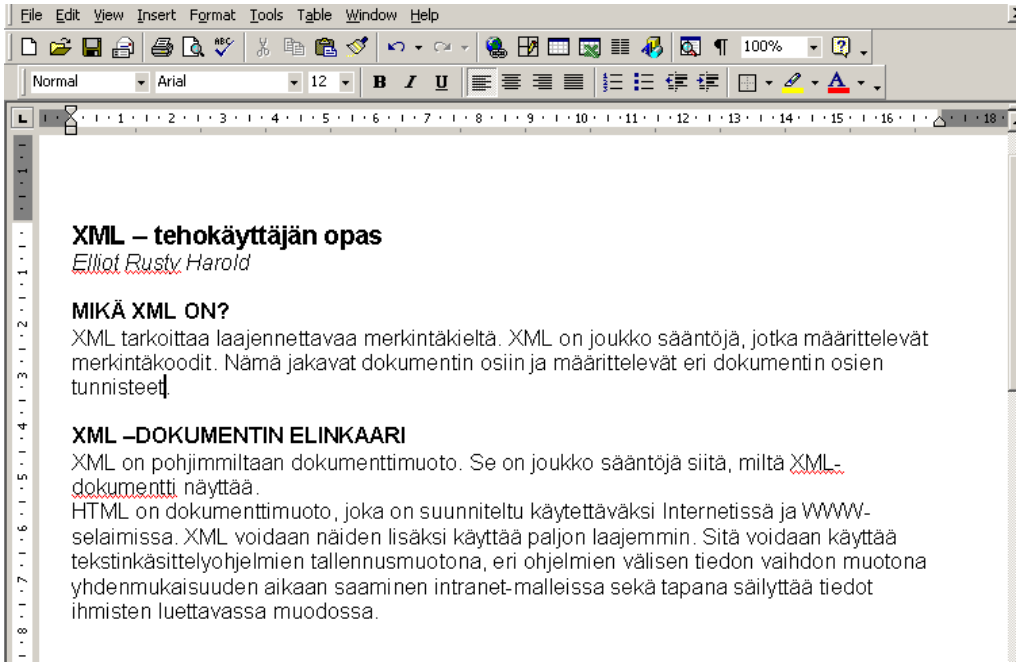
Proseduaaliset dokumentit tallennetaan siten, että niiden ulkoasu vastaa mahdollisimman paljon paperitulostetta. Nykyiset tekstinkäsittelyohjelmat tukevat tätä lähestymistapaa, koska ne tallentavat dokumentin ulkoasuun vaikuttavia tietoja tekstin lomaan. Täten käyttäjä voi muotoilla dokumentin ulkoasua samalla, kun hän kirjoittaa sitä ja havaita välittömästi, miltä dokumentti näyttää paperille tulostettuna. Tällaista tekstinkäsittelyä, jossa käyttäjä kertoo ohjelmalle, miten dokumentti esitetään, kutsutaan proseduaaliseksi. /3/

Proseduaalisissa dokumenteissa

- tallennetaan dokumentin ulkoasumääritykset, kuten esimerkiksi lihavointi, kursivointi ja fonttimääritykset tekstin joukkoon.
- dokumenttien tietosisällöllä ei ole merkitystä.
- tallennusmuodon perusajatus on turvata tiedon esittäminen mahdollisimman nopeasti samassa muodossa, kuin se on talletettu.
- tallennusmuoto määräytyy tallennusvälineen sekä ohjelman mukaan, joilla se tallennetaan. Tämä tekee proseduaalisista dokumenteista välineestä ja laitteesta riippuvaisia.

/3/

Alla olevassa kuvassa on esimerkki proseduaalisesta dokumentista, joka on tehty Microsoft Word -tekstinkäsittelyohjelmalla (kuva 2).



Kuva 2 Esimerkki proseduaalisesta dokumentista /3/

Rakenteiset dokumentit

Proseduaaliset dokumentit ovat alttiita väline-, laite- ja ohjelmistojen muutoksille. Tällä on suuri vaikutus informaation säilymiseen ajan kuluessa, kun ohjelmistot, laitteet ja välineet vaihtuvat. Rakenteisessa dokumenteissa tätä ongelmaa ei ole. Rakenteisissa dokumenteissa kuvataan vain dokumentin rakenne ja sen sisältämä varsinainen informaatio – ulkoasumäärityksiä ei dokumenttiin sisälly. Kun rakenteinen dokumentti halutaan esittää tietyn muotoisena, siihen liitetään erillinen ulkoasun kuvaus. Rakenteinen dokumentti tallennetaan alustasta riippumattomaan ASCII-muotoon.

Rakenteisen dokumentin täytyy noudattaa tietokoneen tulkittavissa olevaa rakennemäärittelyä. Rakenteinen dokumentti koostuu rakenneosista ja niiden sisältämästä informaatiosta. Dokumentin rakenne noudattaa jotakin rakennekuvausta ja dokumentti tulee laatia rakennetta vastaavaksi. /3/ Rakennekuvaus voi olla joko jo valmista standardia noudattava tai itse määritelty. Koska rakenteisen dokumentin tulostusasu voidaan määritellä myöhemmin, käyttäjän ei tarvitse kirjoitusvaiheessa huolehtia dokumentin ulkoasuun liittyvistä

seikoista (esimerkiksi kirjainkoon vaihtamisesta otsikon ja leipätekstin välillä), vaan hän voi keskittyä dokumentin sisällön oikeellisuuteen. Rakenteisten dokumenttien käsittelyyn tarkoitettua järjestelmää kutsutaan deklaratiiviseksi. /3/

Esimerkkejä rakenteisen dokumentaation merkintäkielistä ovat XML, HTML ja SGML. XML (Extensible Markup Language) on erinomainen esimerkki rakenteisen dokumentoinnin merkintäkielestä. XML:llä voidaan luoda puhtaita rakenteellisia dokumentteja, joissa merkintä sisältää vain rakenteen ja sisällön eikä lainkaan muotoiluja. Toisin kuin esimerkiksi HTML-merkinnässä, XML-merkinnässä käytetään sellaisia tunnisteita ja elementtejä, jotka kuvaavat elementin sisältöä. XML:n tunnisteet ja elementit ovat siis täysin käyttäjän määriteltävissä, kun taas HTML:ssä voidaan käyttää vain ennalta määriteltyjä tunnisteita. /3/
Alla olevassa listauksessa on esitettynä kuvan 2 dokumentti XML-muodossa (listaus 1).

Listaus 1 Kuvan 2 dokumentti XML-muodossa. /3/

```
<kirja>
        <nimi>XML – tehokäyttäjän opas</nimi>
        <tekija>Elliot Rusty Harold</tekija>
        <luku>
                <otsikko>
                MIKÄ XML ON?
                </otsikko>

                <kappale>
                XML tarkoittaa laajennettavaa merkintäkieltä.
                XML on joukko sääntöjä, jotka määrittelevät
                merkintäkoodit. Nämä jakavat dokumentin osiin
                ja määrittelevät dokumentin eri osien tunnisteet.
                </kappale>
        </luku>
        <luku>
                <otsikko>
                XML-DOKUMENTIN ELINKAARI
                </otsikko>

                <kappale>
                XML on pohjimmiltaan dokumenttimuoto.
                Se on joukko sääntöjä siitä,
                miltä XML-dokumentti näyttää.
                HTML on dokumenttimuoto, joka on suunniteltu
                käytettäväksi Internetissä ja WWW-selaimissa.
                XML:ää voidaan näiden lisäksi käyttää paljon
                laajemmin.
```

```
                                    Sitä voidaan käyttää tekstinkäsittelyohjelmien
                                    tallennusmuotona, eri ohjelmien välisen tiedon
                                    vaihdon muotona yhdenmukaisuuden aikaan
                                    saamiseksi intranet-malleissa sekä tapana
                                    säilyttää tiedot ihmisten luettavassa muodossa.
                                    </kappale>
            </luku>
</kirja>
```

Listauksesta 1 käyvät hyvin ilmi XML:lle tyypilliset kuvaavat merkintäkoodit, kuten esimerkiksi <nimi>, <tekijä> ja <luku>. Nämä merkintäkoodit ovat täysin käyttäjän määriteltävissä ja kertovat vain dokumentin sisällön. Sisällön muotoilu sivulla tehdään erillisen tyylipohjan avulla. Tämä tyylipohja muotoilee tekstin asetettujen sääntöjen mukaisesti.

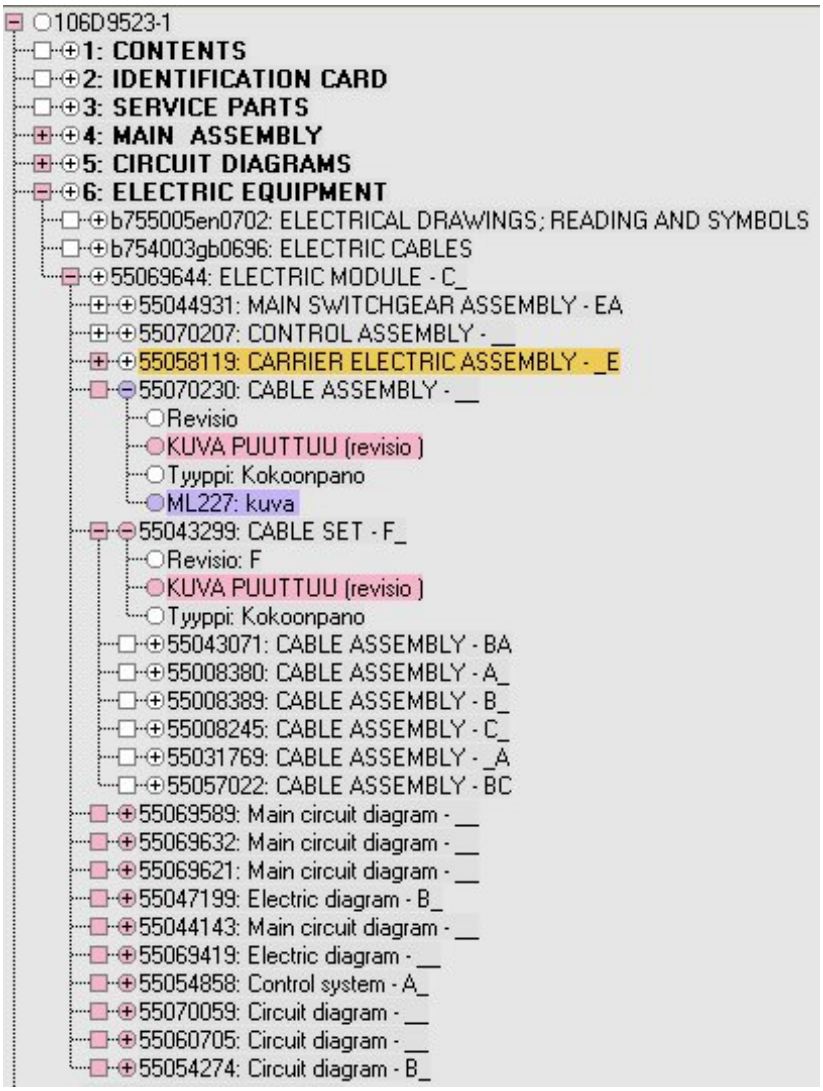
XML on ihanteellinen laajoihin ja monitasoisiin dokumentteihin. Koska sen tieto on rakenteellista, dokumenteista voidaan hakea tehokkaasti haluttuja tietoja tunnisteita käyttämällä: jos edellisessä esimerkissä olisi pitkä luettelo kirjojen tietoja, voitaisiin siitä hakea esim. tietyn kirjailijan teokset. HTML-dokumentista taas tietoja ei voi hakea, koska rakenteen merkkaukseen käytetään samoja tunnisteita. XML tarjoaa myös työasemapuolelle tiedon säilyttämismekanismin, joka yhdistää tietoja useista lähteistä ja näyttää ne yhtenä dokumenttina. Tiedot voidaan järjestää uudelleen nopeasti. /3/

3 ESITUTKIMUS

Komponentin toteutuksen esitutkimus oli työssä tarpeen, sillä ohjelmistoprojektille tyypillisesti aikataulu oli tiukka ja eri lähestymistavat komponentin toteutukseen erosivat paljon toisistaan. Kävi ilmi, että yhden toteuttamistavan valinta olisi lopullinen, eikä aikaa perusteellisiin muutoksiin myöhemmin olisi.

3.1 Lähtökohdat ja vaatimusmäärittely

DokuMentori Oy:n aiemman ohjelmistotuotantokokemuksen avulla selvisi hyvin tarkasti, minkälaista komponenttia toteutuksella haetaan. Puumuotoinen informaation esitys oli havaittu ylivoimaisesti parhaimmaksi ja tehokkaimmaksi tavaksi hahmottaa informaation rakenne ja sen hallinta. DokuMentori Oy esitti Spairport2™ -sovelluksensa rakenteenhallintakomponentin, jonka toimintojen ja käyttöliittymän haluttiin toimivan lähtökohtana uuden rakenteisen informaation hallintakomponentin kehitykselle. Alla esitettynä on kuva Spairport2™ -sovelluksen rakenteenhallintakomponentista (kuva 3).



Kuva 3 Spairport2™ -ohjelmiston rakenteenhallintakomponentti

Kuten kuvassa näkyy, myös SpairPort2™ -ohjelmiston komponentti on puumuotoinen. Kyseistä komponenttia ei kuitenkaan voitu kehittää edelleen, sillä DokuMentori Oy:n päätös vaihtaa ohjelmointikieli Javaan sulki pois tämän vaihtoehdon. Lähdettiin siis kehittämään kokonaan uutta ohjelmistokomponenttia, jota varten tehty vaatimusmäärittely perustui Spairport2™ -ohjelmiston rakenteenhallintakomponenttiin.

Komponentille asetetut vaatimukset voidaan jakaa kolmeen yleiskategoriaan. Nämä kategoriat ovat:

1. Ohjelmoijan – eli jatkokehittäjän ja implementoijan – vaatimukset.
2. Loppukäyttäjän vaatimukset.
3. Ohjelmalliset – eli toiminnalliset – vaatimukset.

Ohjelmoijan vaatimuskategoriaan kuului muun muassa seuraavia vaatimuksia:

- Komponentin rajapinnan tulisi olla selkeä ja helposti käytettävä sekä mahdollisimman monipuolinen erilaisia sovelluksia varten.
- Komponentin sisäisen ohjelmistoarkkitehtuurin, koodin yleisen selkeyden ja kommenttien tulisi myös olla selkeitä. Tämä helpottaa ylläpidettävyyttä ja mahdollistaa edelleen kehityksen vaivatta.
- Komponentin ulkonäön tulisi olla helposti muokattavissa.

Loppukäyttäjän vaatimuskategoriaan kuului esimerkiksi seuraavia vaatimuksia:

- Komponentin käyttöliittymän tulisi olla selkeä ja johdonmukainen. Myös niiden toimintojen, joita loppukäyttäjä käyttää, tulisi olla oletuksena hyvin informatiivisia ja käyttäjäystävällisiä.
- Komponentin ulkoasun ja toimintojen tulisi tukea loppukäyttäjää suurien informaatiomäärien hahmottamisessa ja hallinnoinnissa.

Toiminnalliseen vaatimuskategoriaan kuului muun muassa seuraavanlaisia vaatimuksia:

- Suurien informaatiomäärien hallinnointia ja käyttäjäystävällisyyttä helpottaisivat niin sanotut ”vedä ja pudota” (drag and drop), sekä ”kumoa / tee uudelleen” (undo / redo) -ominaisuudet.
- Komponentissa tulisi olla mahdollista esittää tietoa eri tavoin, kuten esimerkiksi lihavoimalla, kursivoimalla sekä alleviivaamalla ja muuttamalla esitettävän tekstin väriä.
- Komponentissa tulisi olla informaation rakenteisen muodon vuoksi tapa esittää virheitä informaatiossa ja virheiden periytymistä rakenteissa. Tämä hoidettaisiin värityksen ja ikonien avulla.
- Komponenttiin tulisi olla mahdollista lisätä muita komponentteja, kuten esimerkiksi valintaruutuja ja alasvetovalikoita. Tämä mahdollistaa komponentin käytön moniin erilaisiin käyttötarkoituksiin hyvin pienellä koodimäärällä vastedes. Informaation metatietojen esitystapa tultaisiin hoitamaan tämän ominaisuuden avulla.
- Komponentin sisältämää informaatiota tulisi voida muuttaa sekä ohjelmallisesti että graafisesti käyttöliittymän kautta.

Edellä esitetty vaatimuslista koostuu komponentin päävaatimuksista. Pienempiä sivuvaatimuksia kaikista kolmesta yleiskategoriasta löytyi monia. Nämä vaatimukset vaikuttivat paljon komponentin toteuttamistavan valintaan, sillä mitä enemmän valmiita kehityskomponentteja ja koodia käytettäisiin hyväksi, sitä vaikeampi vaatimuksia olisi toteuttaa käytettyjen kehityskomponenttien sisäisten rajoitteiden vuoksi.

3.2 Komponentin kehitysympäristö

Kehitysympäristönä toimi Javan kehitysympäristön (JDK, Java Development Kit) versio 1.5.0, johon oli asennettu päivitys 06. Kehitysympäristön avuksi oli myös asennettu NetBeans 5.0 integroitu ohjelmointiympäristö (IDE, Integrated Development Environment). NetBeans oli monien hyödyllisten työkalujensa lisäksi

liitetty myös DokuMentori Oy:n CVS-palvelimeen (Concurrent Versions System), jonka avulla versionhallinta toteutettiin.

Ohjelman kehitysvaiheessa tarpeelliseksi tuli myös tapa testata komponenttia. Tätä varten oli aiemmin ohjelmoitu pieni testiympäristö, jota ajettiin DokuMentori Oy:n web-palvelimella. Komponenttia testattiin web-ympäristössä applet-ohjelman yhteydessä ja paikallisesti ajettavalla ohjelmalla. Web-testauksen yhteydessä käytettiin Microsoft Internet Explorer -selaimen versiota 6.0 sekä Mozilla FireFox -selaimen versiota 1.0.7.

Ohjelmoinnissa käytettiin HP Compaq nx9030 -kannettavaa tietokonetta, jossa oli käyttöjärjestelmänä Windows XP. Testauksessa käytettiin sekä HP:n että Dellin kannettavia tietokoneita, Machintosh iBook -kannettavia tietokoneita ja erinäisiä pöytäkonemalleja. Komponenttia testattiin Windows XP, Windows 2000, Mac OS X sekä Linux (RedHat) -käyttöjärjestelmillä Java-ajoympäristön versiolla 1.5.0_r4.

3.3 Komponentin toteuttamistavan valinta

Kuten edellä mainittiin, DokuMentori Oy:n ohjelmistokehitysosasto vaihtoi tuottavaksi ohjelmointikielekseen Javan. Tämä sulki pois mahdollisuuden käyttää ja kehittää edelleen edellisten ohjelmistoprojektien komponentteja ja tekniikoita. Vaihtoehdoiksi jäi uuden komponentin luominen täysin alusta tai Java-ohjelmistokehitysympäristön JTree-puukomponentista periytyvän komponentin luominen. Luominen käsittää tässä yhteydessä suunnittelun, toteutuksen ja testauksen.

Molempia vaihtoehtoja harkittiin ja niiden hyviä sekä huonoja puolia mitattiin. Kumpaakin toteutustapaa myös testattiin pienillä sovellusesimerkeillä, jolloin saatiin käsitys toteutukseen kuluvasta ajasta. Kävi ilmi, että periyttämisellä luotu komponentti olisi moninkertaisesti nopeammin ja siten myös halvemmin saatavilla tuotannolliseen käyttöön kuin kokonaan alusta luotava komponentti. Kun JTree-komponentista periytyvän komponentin muokkaus vaatimusmäärittelyiden mukaisesti kesti arvioiden mukaan noin kolme kuukautta, olisi kokonaan uuden

komponentin luominen kestänyt kuudesta yhdeksään kuukautta. Vastapainona nopealle valmistumiselle periytetyllä komponentilla oli periytymisestä johtuvat rajoitteet ominaisuuksien toteutuksessa. Siinä missä täysin uusi komponentti on mukautumiskykyisin, on periytetty komponentti yliluokan rajoitteiden alainen.

Suurin haaste mukautumiskyvyssä oli informaation metatietojen esitys puumallissa. Vaatimuksen mukainen esitys- ja hallintaratkaisu vaati komponentilta käytännössä sen, että puukomponentin sisälle voitaisiin sijoittaa toinen puukomponentti.

Ohjelmistoprojekteissa usein ilmenevä kiireellisyys vei lopulta voiton, kun JTree-komponentista periytetyn komponentin taipuvuus uusin ominaisuuksiin koettiin riittäväksi. Riskit tiettyjen vaatimusten toteutumiskyvystä hyväksyttiin ja alustavat suunnitelmat kyseisten vaatimusten korvaamiseksi toisilla toiminnoilla tehtiin.

4 KOMPONENTIN SUUNNITTELU

Kun komponentin toteutuksen lähestymistapa oli valittu, pystyttiin sen suunnittelu aloittamaan. Sovelluskomponentin suunnittelu käsittää komponentin käyttöliittymäsuunnittelun, järjestelmäarkkitehtuurin suunnittelun ja moduulisuunnittelun. Suunnittelun kaikissa vaiheissa tärkeää oli ottaa huomioon komponentin kahden käyttäjäkunnan – jatkokehittäjien ja loppukäyttäjien – tarpeet. Projektin kaikista vaiheista suunnittelulla oli näkyvin vaikutus komponentin käyttäjäkunnille. Lisäksi suunnittelu vaikutti eniten siihen, kuinka laajennettavissa komponentti olisi myöhempien projektien tarpeisiin.

4.1 Käyttöliittymäsuunnittelu

Käyttöliittymäsuunnittelun lähtökohdaksi haluttiin DokuMentori Oy:n Spairport2™ -ohjelmiston rakenteenhallintakomponentti. Tämä tarkoitti käytännössä puumuotoista ratkaisua rakenteisen informaation esitykseen.

Komponentti tulisi kuitenkin sisältämään esikuvaansa enemmän toiminnallisuuksia, jotka vaativat toimivan ja käyttäjäystävällisen käyttöliittymän.

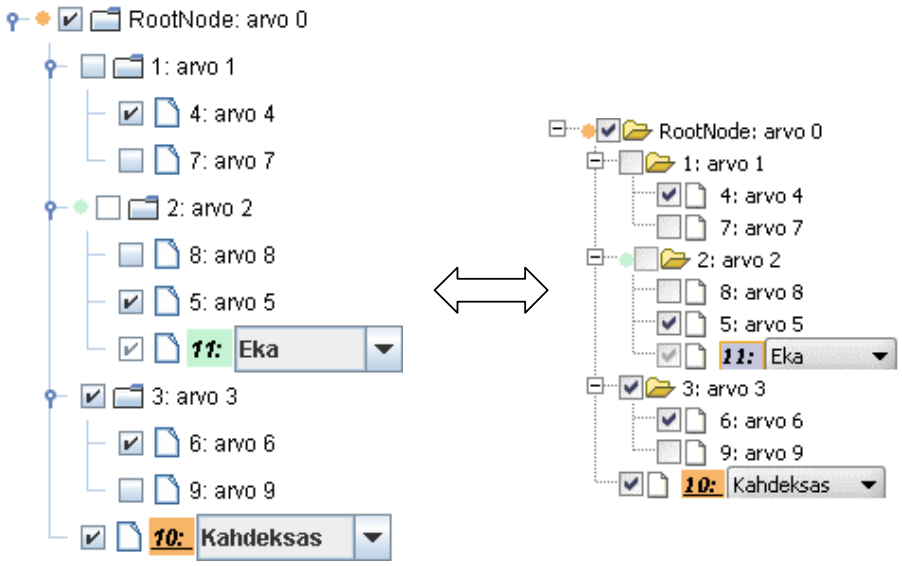
Koska komponenttia tultaisiin myöhemmin käyttämään moniin erilaisiin tarkoituksiin, oli käyttöliittymän ulkoasusta tärkeä saada mahdollisimman helposti muokattava. Komponenttia voitaisiin yksilöitävän ulkonäön avulla käyttää hyväksi myös täysin eri yrityksissä, mikä mahdollistaisi komponentin tuotteistamisen myöhemmässä vaiheessa kehitystä. Ulkonäön yksilöiminen onnistui osittain helposti Java-ajoympäristön (Java Runtime Environment) käyttötuntumaominaisuuden vuoksi (Look & Feel). Tämän ominaisuuden avulla koko käyttöliittymän ulkonäkö voidaan vaihtaa helposti, vain yhdellä rivillä koodia.

Java-ajoympäristön käyttötuntumaominaisuus tarkoittaa yleisesti käyttöliittymän ulkonäköön liittyvien seikkojen lisäksi myös käyttäjäystävällisyyteen ja käyttöliittymään itseensä liittyviä suunnitteluperiaatteita. Sun Microsystems on julkaissut aiheesta kirjan, joka löytyy myös ilmaiseksi Internetistä seuraavasta osoitteesta:

- http://java.sun.com/products/jlf/ed1/dg/index.htm

Tämän työn yhteydessä on kuitenkin merkittävää rajata käsite käyttötuntuma tarkoittamaan Java-ajoympäristön ominaisuutta, jolla käyttöliittymän ulkonäkö voidaan yksilöidä. Tämä on mahdollista Javan Swing-käyttöliittymäkirjaston avulla, jonka sisältämien komponenttien arkkitehtuurissa toiminnallisuus ja ulkonäkö on erotettu toisistaan.

Työssä oli siis myös tärkeä perehtyä tähän Javan ominaisuuteen tarkasti. Alla olevassa kuvassa on esitettynä käyttötuntuman vaihdon vaikutus komponentin ulkonäköön (kuva 4).



Kuva 4 Käyttötuntuman vaihdon vaikutus komponentin ulkonäköön

Koska Java on suunniteltu toimimaan sekä monessa eri laitteistossa että monissa eri laitteistojen käyttöjärjestelmissä, on sen ajoympäristössä käyttövalmiina seuraavat käyttötuntumat:

- Sekaympäristö-käyttötuntuma.
- Windows-käyttötuntuma.
- Macintosh-käyttötuntuma.
- CDE/Motif-käyttötuntuma.

/4/

Näiden Java-ajoympäristön sisältämien käyttötuntumien lisäksi käyttötuntumia voi luoda itse. Sen vuoksi käyttötuntumia saa myös paketteina Internetistä ja suurin osa niistä on täysin ilmaisia. Esimerkkinä Internetistä saatavista käyttötuntumista on Napkin-käyttötuntuma, jonka löytää seuraavasta osoitteesta:

- http://napkinlaf.sourceforge.net/

Komponentin ulkonäön yksilöintiin liittyi myös sen toiminnallisuuksista aiheutuvia asetuksia. Näistä asetuksista suurin osa liittyi värivalintoihin, jolloin päädyttiin toteuttamaan tiettyjen värien asetukseen rajapintoja. Rajapintojen avulla

vältettäisiin koko käyttötuntuman vaihtaminen vain värien vuoksi. Tämä osoittautui toimivaksi ja työtä säästäväksi ratkaisuksi.

Ohjelmoijille suunnattuun käyttömukavuuteen komponentissa vaikutti hyvin suunniteltu ja toteutettu ohjelmistoarkkitehtuuri, moduulisuunnittelu ja dokumentointi. Myös tämä käyttömukavuus otettiin huomioon kyseisissä työvaiheissa.

4.3 Järjestelmäarkkitehtuurin suunnittelu

Komponentin järjestelmäarkkitehtuurin suunnittelun lähtökohtana toimi vaatimus, jonka mukaan komponentin tulisi olla JavaBean-arkkitehtuurin mukainen.

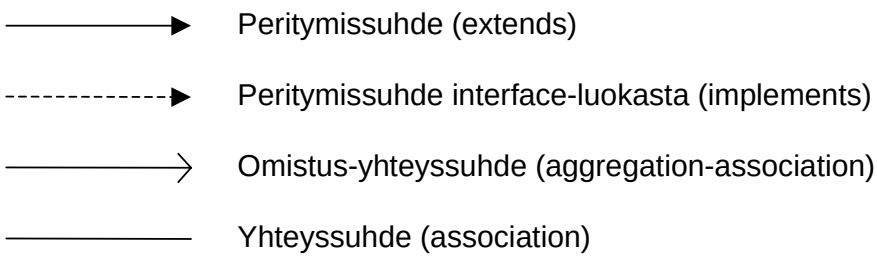
JavaBean-arkkitehtuuri on Javan komponenttimalli, joka mahdollistaa Javan komponenttikehitysmallin. Tämän kehitysmallin avulla voidaan rakentaa ohjelmia komponenttiolioista, joita voi asettaa paikalleen joko graafisesti tai ohjelmallisesti. JavaBean-arkkitehtuuri tarkoittaa ohjelmistoteknisesti monia asioita, joita ovat muun muassa:

- tuki visuaaliselle ohjelmoinnille.
- yhteensopivuus muiden käyttöliittymäkomponenttien kanssa.
- olio-ohjelmoinnin periaatteiden mukaisesti luotu ohjelmakoodi.
- tietoturvariskien hallinta.
- yhteensopivuus monisäikeisissä ohjelmissa.
- olion yksilöitävyys sekä visuaalisesti että ohjelmallisesti.
- vuorovaikutteisuus eli yhteensopivuus Javan tapahtumankäsittelymallin kanssa.

/1/

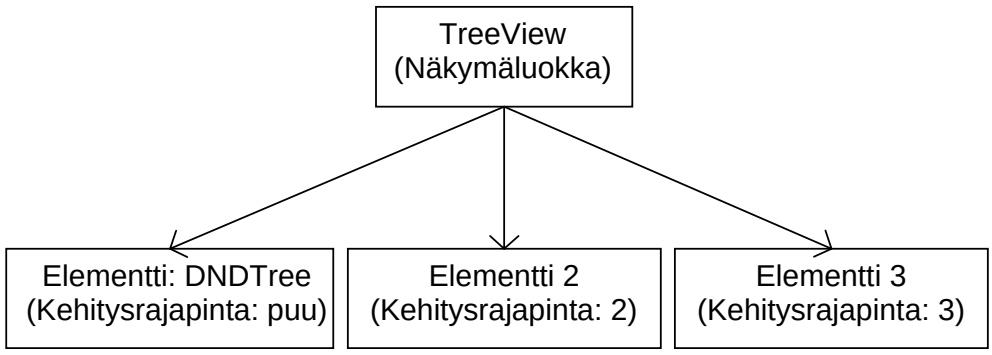
NetBeans-ohjelmointiympäristö toteutti osan yllä olevista JavaBean-arkkitehtuurin vaatimuksista automaattisesti. Osa vaatimuksista täyttyi myös, kun päätettiin käyttää Java-ajoympäristön JTree-kehityskomponentista periytettyä puuelementtiä. Muut vaatimukset täytyi toteuttaa käsin. Osa arkkitehtuurin vaatimista asioista on komponentteittaisia ja siten sovellettavissa, joten esimerkiksi olion yksilöitävyys voidaan tämän hallintakomponentin yhteydessä tehdä vain ohjelmallisesti.

Komponentin järjestelmäarkkitehtuuri on kokonaisuudessaan esitettynä liitteessä 1. Arkkitehtuurin läpikäynti on esitetty vastedes vaiheittain selkeyden vuoksi. Arkkitehtuuri sisältää komponentin tärkeimmät luokat ja niiden periytymishierarkian Java-ajoympäristön natiiviluokkiin asti, jotta työn hahmottaminen olisi helpohkoa. Alla olevassa kuvassa on esitettynä arkkitehtuurien luokkien suhdemerkinnät (kuva 5):



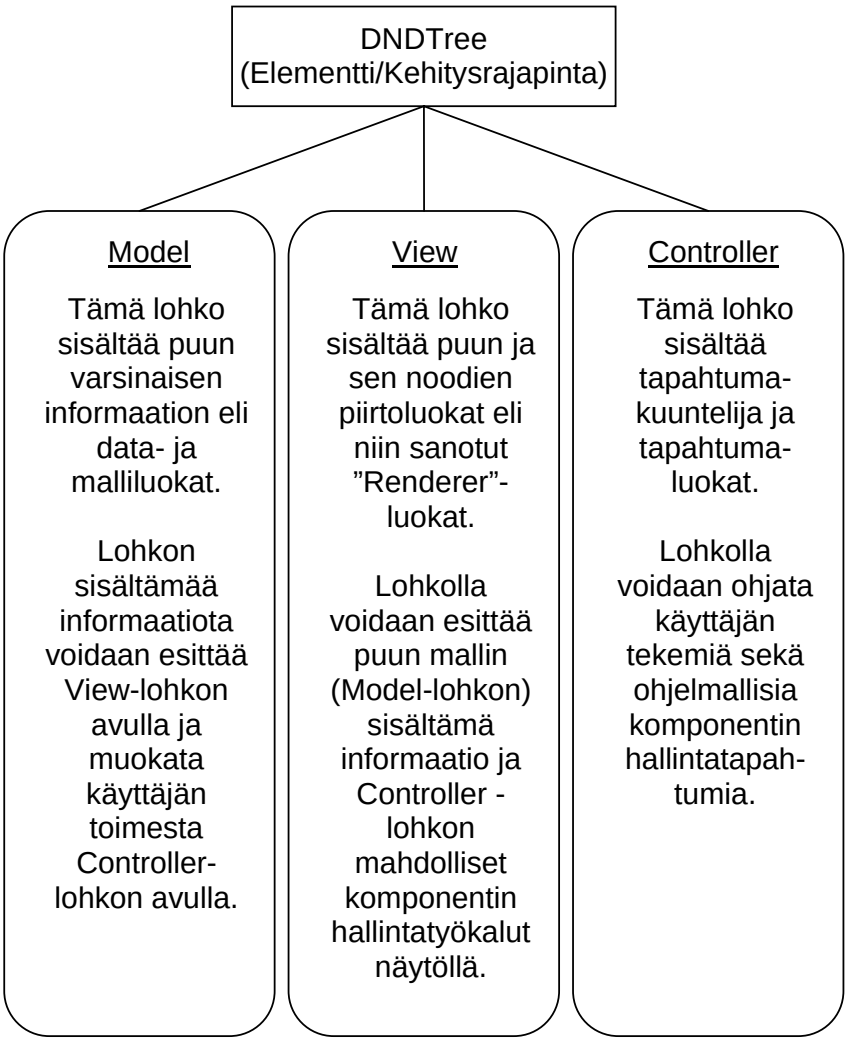
Kuva 5 Arkkitehtuurikaavioiden luokkien suhdemerkinnät

Komponentin pääluokaksi luotiin näkymäluokka (TreeView), jotta komponentti olisi mahdollisimman helposti laajennettavissa myös tulevaisuudessa. Näkymäluokkaan voitaisiin siten liittää myös muita elementtejä tulevan puu-elementin (DNDTree) lisäksi. Näitä elementtejä voidaankin kutsua kehitysrajapinnoiksi. Oheisessa kuvassa on esitettynä näkymäluokka, johon on liitetty kolme elementtiä (kuva 6).

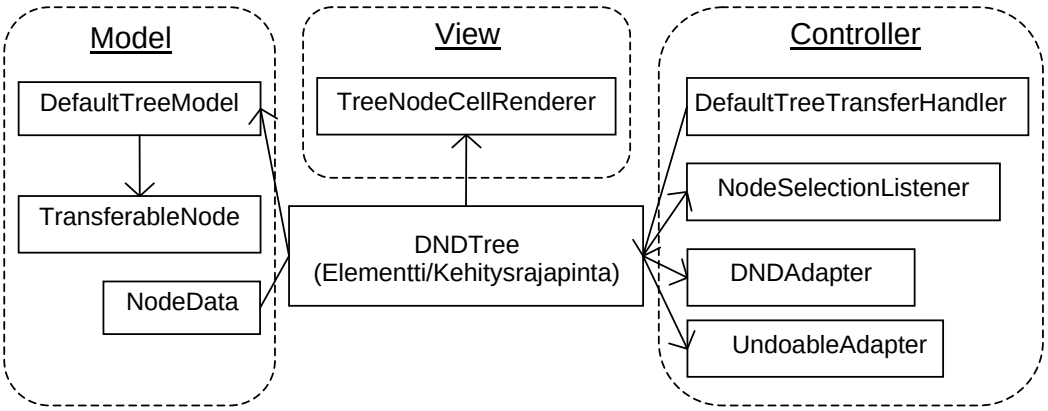


Kuva 6 Komponentin näkymäluokan arkkitehtuurin yleiskuvaus

Elementtien eli kehitysrajapintojen järjestelmä- ja moduuliarkkitehtuuri noudattaa myöskin JavaBean-standardin vaatimuksia. Siksi jopa elementit ovat sekä graafisesti että ohjelmallisesti asetettavissa näkymäluokkaan. Näin pienistä rakennusosista voidaan koota suuria kokonaisuuksia. Javan komponenttiarkkitehtuuri sisältää implementaation MVC-ratkaisumallista, joka näkyy myös elementtien järjestelmäarkkitehtuurissa. MVC-ratkaisumallin periaatetta tulisi pyrkiä aina noudattamaan myös uusien elementtien luomisessa. Koska DNDTree-puuelementti on ensimmäinen komponenttiin toteutettu elementti, käytetään sen järjestelmäarkkitehtuurin yleiskuvausta esimerkkinä. MVC-mallin toiminta DNDTree-puuelementissä on esitettynä kuvassa 7. DNDTree-puuelementin järjestelmäarkkitehtuurin yleiskuvaus ja MVC-mallin vaikutus on esitettynä kuvassa 8.



Kuva 7 MVC-ratkaisumallin toiminta DNDTree-elementissä.



Kuva 8 DNDTree-puuelementin järjestelmäarkkitehtuurin yleiskuvaus.

Kuvan 8 järjestelmäarkkitehtuuri on vain yksinkertaistettu läpileikkaus yksittäisen elementin rakenteesta. Ainoastaan DNDTree-elementtiin suoraan liittyvät luokat ovat esitettyinä selkeyden vuoksi. Todellisuudessa elementin järjestelmäarkkitehtuuriin kuuluu vielä lisäksi runsaasti luokkia, mutta ne ovat esitettyinä komponentin täydellisessä järjestelmäarkkitehtuurissa (liite 1).

Kuvasta 8 voidaan nähdä, kuinka kaikki yksittäiset elementin ohjelmamoduulit voidaaan lajitella osa-alueittain kuulumaan joko Model-, View- tai Controller-kategoriaan. Näiden yksittäisien moduulien toiminta ja tarkoitusperä selitetään tarkemmin luvussa 4.4 (Komponentin moduulisuunnittelu).

4.4 Komponentin moduulisuunnittelu

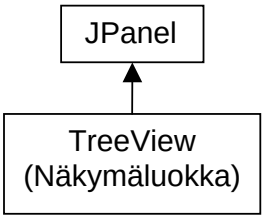
Komponentti koostuu monista yksittäisistä moduuleista eli luokista. Näiden luokkien tarkoitusperät ja suunnittelu käydään tässä luvussa läpi, jotta saataisiin selkeämpi kuva liitteen 1 järjestelmäarkkitehtuurista ja siitä, kuinka komponentti toimii. Suurin osa luokista luotiin JTree-luokan pakottaman arkkitehtuurin mukaisesti, joka varmisti JavaBean-yhteensopivuuden ja hyväksi havaittujen ratkaisumallien käytön. Tämä oli siis vain hyvä asia ja suositeltavaa myös kaikkien tulevien luokkien toteutuksessa.

Koska MVC-ratkaisumalli on niin selvästi esillä kehitysrajapintaluokkien moduuliarkkitehtuurissa, on DNDTree-luokan alla olevien yksittäisten luokkien esittely jaoteltu kategorioittain ratkaisumallin mukaisesti (Model, View ja Controller).

TreeView

TreeView suunniteltiin ohjelman näkymäluokaksi, jonka ensisijaisena tehtävänä on hallinnoida kaikkia komponenttiin liittyviä kehitysrajapintoja eli elementtejä ja toteuttaa niiden rajapinta jatkokehittäjälle. TreeView-luokka toteuttaa myös tiettyjä muita rajapintoja jatkokehittäjälle, kuten esimerkiksi asetusrajapinnan, tarvittavat rakentajat, apumetodit ja tapahtumankäsittelijöiden välitysmetodit. Lisäksi

TreeView-luokan tehtävänä on varmistaa JavaBean-yhteensopivuus. Alla olevassa kuvassa on esitettynä näkymäluokan moduulisuunnittelun arkkitehtuuri (kuva 9).

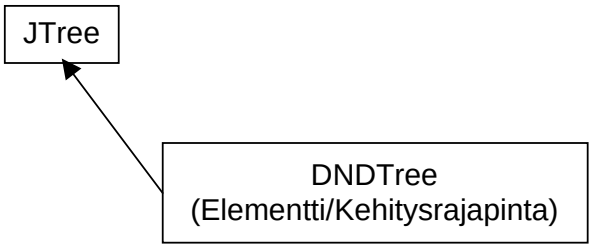


Kuva 9 TreeView-luokan moduuliarkkitehtuuri.

Kuten kuvasta 9 nähdään, TreeView-luokka periytyy Java-ajoympäristön JPanel-luokasta. Myös tämä varmistaa komponentin JavaBean-yhteensopivuutta.

DNDTree

DNDTree oli työn vaatimusmäärittelyjen mukaisesti toteutettu komponentin ensimmäinen kehitysrajapinta eli elementti. DNDTree-luokan tehtävänä oli toteuttaa suurin osa luoduista vaatimusmäärittelyn vaatimuksista. Sen vuoksi suurin osa komponentin toiminnoista keskittyy DNDTree-luokkaan, jolloin luokan kehitysrajapinnan tehtävänä on tarjota toiminnot TreeView-moduulille. Vaatimusmäärittelyssä esitetyt tärkeimmät vaatimukset ovat esitettyinä luvussa 3.1. Alla olevassa kuvassa on esitettynä DNDTree-luokan moduulisuunnittelun arkkitehtuuri (kuva 10).



Kuva 10 DNDTree-luokan moduuliarkkitehtuuri.

Koska puumuotoinen informaation esitys oli asiakasvaatimus ja paras tapa hallinnoida rakenteista informaatiota, päädyttiin periyttämään luokka Java-

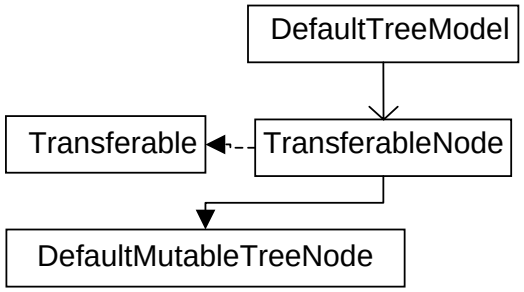
ajoympäristön JTree-kehityskomponentista. Valintaan liittyi myös edellä esitettyjä muita kriteerejä.

4.4.1 Model – Tietomalli

Komponentin tietomalli käsittää komponentin sisältämän informaation. Koska informaatio on esitettynä puumallin mukaisesti, sisältävät DNDTree-luokan dataluokat kyseisen informaation.

DefaultTreeModel, TransferableNode
DefaultTreeModel on Java-ajoympäristön puukomponenttien oletusdatamalli, jonka tehtävänä on sisältää yksittäisen puun noodit. Lisäksi sen tehtävänä on toimia rajapintana kaikille toiminnoille, joita puun sisältämälle informaatiolle tehdään. Tämä moduuli on Java-ajoympäristön natiiviluokka.

TransferableNode on DNDTree-puuelementin oma noodiluokka. TransferableNode-luokan tehtävänä on sisältää DND-puuelementin yksittäisen noodin informaatio, sen lapsinoodit, sisarusnoodit ja äitinoodi. Näiden tietojen avulla voidaan määrittää noodin paikka puussa ja siten rakenteellisen informaation rakenne. Noodin tehtävänä on myös toteuttaa rajapinta näiden tietojen muuttamiselle. Alla olevassa kuvassa on esitettynä DefaultTreeModel- ja TransferableNode-luokkien moduulisuunnittelun arkkitehtuuri (kuva 11).

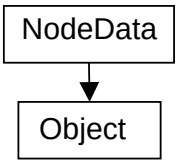


Kuva 11 DefaultTreeModel- ja TransferableNode -luokkien moduuliarkkitehtuuri.

Kuten kuvasta 11 nähdään, DefaultTreeModel-luokka sisältää TransferableNode-luokkia. TransferableNode-luokka periytyy DefaultMutableTreeNode-luokasta, joka on Java-ajoympäristön oma noodiluokka. Tämä periytymishierarkia varmisti yhteensopivuuden DefaultTreeModel-luokan kanssa. Lisäksi TransferableNode toteuttaa Transferable-luokan rajapinnan eli periytyy myös tästä. Transferable-rajapinta on Java-ajoympäristön oma rajapinta ”vedä ja pudota”-tapaan siirrettäville moduuleille. Tämä kuuluu osaksi komponentin ”vedä ja pudota”-ominaisuuden toteutusta.

NodeData

NodeData-luokka on apuluokka, joka luotiin yksinkertaistamaan rakentajia ja yksittäisen noodin sisältämän informaation välitystä. NodeData-luokka onkin niin sanottu ”data wrapper”-luokka, joka sisältää suuren määrän informaatiota ja niiden get()- ja set()-metodeita. Luokka toteuttaa rajapinnan sisältämälleen informaatiolle. NodeData-luokan moduulisuunnittelun arkkitehtuuri on esitettynä alla olevassa kuvassa (kuva 12).



Kuva 12 NodeData-luokan moduuliarkkitehtuuri.

4.4.2 View – Näkymä

Näkymä käsittää DNDTree-luokan ulkonäön, informaation ja luokan hallintakomponenttien piirtämisen näytölle.

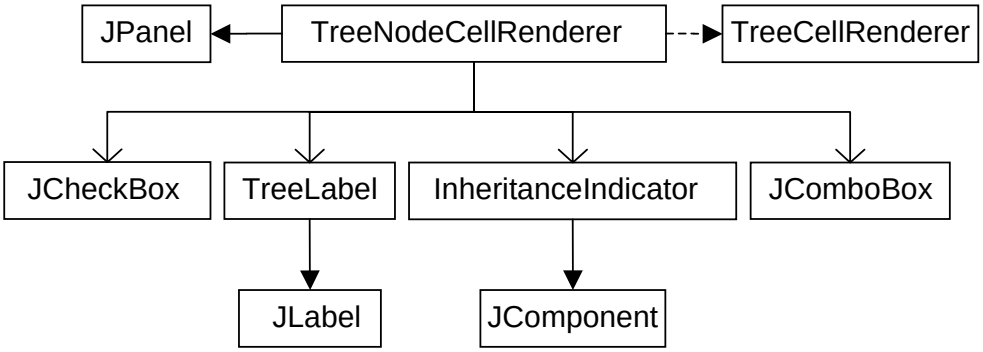
TreeNodeCellRenderer, TreeLabel, InheritanceIndicator

TreeNodeCellRenderer on piirtoluokka, jonka tehtävänä on piirtää näytölle puun yksittäinen noodi. Luokka toteuttaa Java-ajoympäristön vaatiman rajapinnan piirtämiselle.

TreeLabel on luokka, jonka tehtävänä on piirtää tekstiä näytölle erilaisin värein ja tyylein. Luokka toteuttaa Java-ajoympäristön vaatiman komponenttirajapinnan, jolloin se pystyy toimimaan yksittäisenä komponenttina.

InheritanceIndicator on luokka, jonka tehtävänä on piirtää rakenteisen informaation esityksessä virheiden periytymiseen käytetty ikoni monin erilaisin värein. Myös tämä luokka toteuttaa Java-ajoympäristön vaatiman komponenttirajapinnan, jolloin se pystyy toimimaan yksittäisenä komponenttina.

Alla olevassa kuvassa ovat esitettyinä TreeNodeCellRenderer-, TreeLabel- ja InheritanceIndicator-luokkien moduulisuunnittelun arkkitehtuuri ja kyseisten luokkien suhteet toisiinsa (kuva 13).



Kuva 13 TreeNodeCellRenderer-, TreeLabel- ja InheritanceIndicator-luokkien moduuliarkkitehtuuri ja keskinäiset suhteet.

Kuten kuvasta 13 nähdään, TreeNodeCellRenderer-luokka periytyy Java-ajoympäristön JPanel-luokasta ja toteuttaa TreeCellRenderer-rajapinnan. Tämä hierarkia mahdollistaa TreeNodeCellRenderer-luokan toimimisen yksittäisenä komponenttina ja sen yhteensopivuuden DNDTree-luokan kanssa. Kuten edellä mainittiin, DNDTree on periytetty Java-ajoympäristön JTree-kehityskomponentista ja käyttää siten samoja piirtoluokkarajapintoja. Jotta TreeNodeCellRenderer voisi piirtää kaikki puun noodilta vaaditut ominaisuudet, täytyy sen omistaa TreeLabel-, InheritanceIndicator-, JcomboBox- ja JCheckBox-luokkien instanssit. TreeLabel-instanssi mahdollistaa noodin sisältämän tekstisisällön piirtämisen näytölle, InheritanceIndicator-instanssi mahdollistaa periytyneiden virheiden esittämisen

näytöllä, JCheckBox-instanssi mahdollistaa käyttöliittymän mahdollisille noodivalinnoille ja JComboBox-instanssi mahdollistaa käyttöliittymän noodin mahdolliselle tietosisällön monivalinnalle.

Kuvasta 13 nähdään myös, että TreeLabel-luokka periytyy Java-ajoympäristön JLabel-luokasta. Koska näiden kahden toiminnallisuus on pääosin sama, oli periyttäminen paras vaihtoehto TreeLabel-tyylisen luokan luomiseen. Jotta tekstiä osattaisiin piirtää eri tavoin, on komponentin piirtometodi ylikirjoitettu ja piirtotyylin valintaan on lisätty rajapinta.

Kuvasta 13 nähdään myös, että InheritanceIndicator-luokka periytyy Java-ajoympäristön JComponent-luokasta. Tämä mahdollistaa luokan toimimisen yksittäisenä komponenttina.

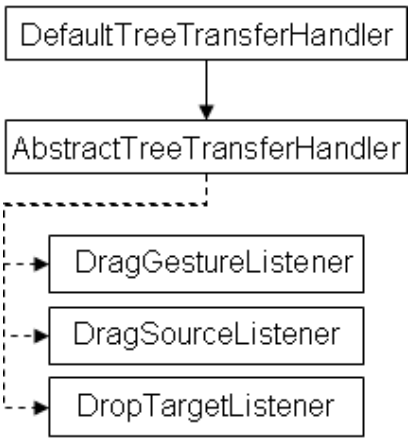
4.4.3 Controller – Hallinta

Hallinta käsittää DNDTree-luokan varsinaisen toiminnallisuuden loppukäyttäjän syötteiden mukaisesti.

AbstractTreeTransferHandler, DefaultTreeTransferHandler
AbstractTreeTransferHandler on abstrakti luokka, joka toteuttaa rajapinnat Java-ajoympäristön DragGestureListener-, DragSourceListener- ja DropTargetListener-luokkien mukaan. Nämä rajapinnat mahdollistavat ”vedä ja pudota”-toiminnallisuuden toteuttamisen kuuntelemalla kyseisiä käyttäjän tekemiä tapahtumia. AbstractTreeTransferHandler asettaa toteutettavaksi tiettyjä rajapintoja, jotta luokasta periytetyt luokat voivat määrätä yksilöllisesti ”vedä ja pudota”-toiminnallisuuden käyttäytymisen.

DefaultTreeTransferHandler on luokka, joka toteuttaa DNDTree-puuelementin ”vedä ja pudota”-toiminnallisuuden käyttäytymisen.

AbstractTreeTransferHandler- ja DefaultTreeTransferHandler-luokkien moduulisuunnittelun arkkitehtuuri ja kyseisten luokkien keskinäiset suhteet ovat esitettyinä alla olevassa kuvassa (kuva 14).

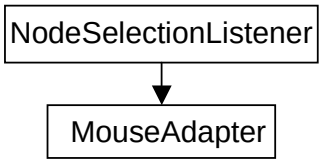


Kuva 14 AbstractTreeTransferHandler- ja DefaultTreeTransfer-luokkien moduuliarkkitehtuuri ja keskinäiset suhteet.

Kuten kuvasta 14 nähdään, AbstractTreeHandler-luokka periytyy DragGestureListener-, DragSourceListener- ja DropTargetListener-rajapintaluokista. DefaultTreeTransferHandler taas periytyy AbstractTreeTransferHandlerista, toteuttaen AbstractTreeTransferHandlerin asettaman rajapinnan ”vedä ja pudota”-toiminnoille. Näin toimintojen käyttäytyminen voidaan määrätä yksilöllisesti.

NodeSelectionListener

NodeSelectionListener-luokka toteutettiin kuuntelemaan loppukäyttäjän hiirellä tekemiä komponentin ohjaussyötteitä. Luokka toteuttaa ohjaussyötteiden mukaisesti operaatioita DNDTree-luokalle. Alla olevassa kuvassa on esitettynä NodeSelectionListener-luokan moduulisuunnittelun arkkitehtuuri (kuva 15).



Kuva 15 NodeSelectionListener-luokan moduuliarkkitehtuuri.

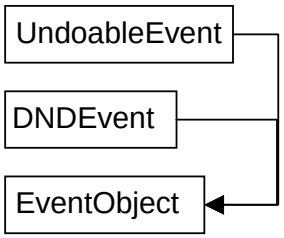
Kuvasta 15 nähdään, että NodeSelectionListener-luokka periytyy Java-ajoympäristön MouseAdapter-luokasta. Tämä periytymishierarkia mahdollistaa hiiritapahtumiin reagoinnin halutuilla toiminnoilla.

UndoableEvent, DNDEvent

UndoableEvent on tapahtumaluokka, jonka tarkoitus on välittää komponentin ”kumoa/tee uudelleen”-toiminnallisuuden mukaisia tapahtumia mahdollisesti asetetuille kuuntelija-olioille. Luokka toteuttaa metodit, joiden avulla saadaan tarvittava tieto tehdystä tapahtumasta.

DNDEvent on tapahtumaluokka, jonka tarkoitus on välittää komponentin ”vedä ja pudota”-toiminnallisuuden mukaisia tapahtumia mahdollisille kuuntelija-olioille. Jotta tarvittavat tiedot tehdystä tapahtumasta saadaan, myös tämä tapahtumaluokka toteuttaa rajapinnat näille tiedoille.

UndoableEvent- ja DNDEvent-luokkien moduulisuunnittelun arkkitehtuuri on nähtävillä seuraavassa kuvassa (kuva 16).



Kuva 16 UndoableEvent- ja DNDEvent-luokkien moduuliarkkitehtuuri.

Kuvasta 16 nähdään, että kumpikin tapahtumaluokista periytyy Java-ajoympäristön EventObject-luokasta. Tämä periytymishierarkia mahdollistaa tapahtumien lähettämisen ja vastaanottamisen Javan arkkitehtuurin mukaisesti.

UndoableListener, DNDListener, UndoableAdapter, DNDAdapter

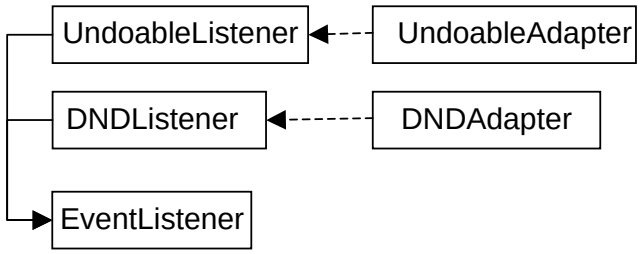
UndoableListener-luokka on tapahtumakuuntelijaluokka, jonka tarkoituksena on kuunnella UndoableEvent-tapahtumia ja toimia rajapintakehyksenä (interface). Rajapintakehyksenä UndoableListener-luokka asettaa siitä periytettävälle toiselle luokalle toteutettavaksi rajapintoja, joiden avulla tapahtumiin reagointi voidaan yksilöidä.

UndoableAdapter on adapteriluokka, joka toteuttaa UndoableListener-tapahtumakuuntelijaluokan tarjoaman rajapintakehyksen halutuin toiminnallisuuksin.

DNDListener-luokka on myös tapahtumakuuntelijaluokka, jonka tarkoitus on kuunnella DNDEvent-tapahtumia ja toimia rajapintakehyksenä. DNDListener-luokka asettaa rajapintakehyksen, jonka avulla tietynlaisiin tapahtumiin reagointi voidaan yksilöidä.

DNDAdapter on adapteriluokka, joka toteuttaa DNDListener-tapahtumakuuntelijaluokan tarjoaman rajapintakehyksen halutuin toiminnallisuuksin.

UndoableListener-, DNDListener-, UndoableAdapter- ja DNDAdapter-luokkien moduulisuunnittelun arkkitehtuuri ja kyseisten luokkien keskinäiset suhteet ovat esitettyinä alla olevassa kuvassa (kuva 17).



Kuva 17 UndoableListener-, DNDListener-, UndoableAdapter- ja DNDAdapter-luokkien moduuliarkkitehtuuri ja keskinäiset suhteet.

Kuten kuvasta 17 nähdään, molemmat tapahtumankuuntelijaluokat (UndoableListener ja DNDListener) periytyvät EventListener-luokasta. Tämä periytymishierarkia mahdollistaa yhteensopivuuden Java-ajoympäristön tapahtumakuuntelijastandardin kanssa, joka taas mahdollistaa yhteensopivuuden DNDTree-luokan kanssa.

Molemmat adapteriluokat (UndoableAdapter ja DNDAdapter) periytyvät esitetyistä tapahtumankuuntelijaluokista toteuttaen niiden rajapinnat halutuin toiminnallisuuksin.

5 KOMPONENTIN TOTEUTUS JA LOPPUTULOKSET

Suunnittelun jälkeen alkoi komponentin varsinainen toteutus, jonka jälkeen oli vuorossa integrointi suurempaan ohjelmaan ja komponentin testaus. Koska työhön kuului suunnittelu ja toteutus, ei varsinaisen integroinnin ja testauksen tuloksia esitetä myöhemmin kovinkaan tarkasti.

5.1 Toteutus

Komponentin toteutus eteni komponentin vaadittujen toiminnallisuuksien toteutuksella yksi kerrallaan. Toteutus keskittyi alkupäässä DNDTree-kehitysrajapintaan, jonka toiminnallisuudet varmistettiin ensin ja tämän jälkeen siirryttiin toteuttamaan ulkonäköä. Viimeiseksi päädyttiin toteuttamaan TreeView-

näkymäluokka, jolloin kehitysrajapinta saatiin kytkettyä jatkokehittäjälle tarkoitettuun rajapintaan. Koko toteutuksen ajan suoritettiin ohjelmakoodin testausta komponentin ominaisuuksien lisääntyessä.

Koska suunnittelu oli tehty perusteellisesti, tehdyt suunnitelmat eivät juuri eläneet komponentin toteutuksen aikana. Toteutuksen aikana suoritettujen kokeilujen ja tietomäärän lisääntymisen yhteydessä saatiin ideoita, joilla voitaisiin tehostaa komponentin käyttöä. Esimerkki tällaisesta suunnitelmasta on luvussa 5.5 esitetty jatkokehitysidea ”kumoa/tee uudelleen”-toiminnallisuudelle. Komponentin toteutus oli myös helppoa runsaan suunnittelun vuoksi.

5.2 Esiintyneet ongelmat ja niiden ratkaisut

Toteutusteknisiä ongelmia esiintyi ohjelmistoprojektille tyypilliseen tapaan tasaisesti, mutta mikään niistä ei ollut toteutusta hidastava. Vaatimuksiin liittymiin kysymyksiin ja toteutusteknisiin ongelmiin vastasi Erno Nieminen, joka oli työn tilaaja ja samalla DokuMentori Oy:n ohjelmistopäällikkö. Ainoa komponenttiin projektin aikataulun ulkopuolelle jäänyt ohjelmavirhe on komponentin piirtämiseen liittyvä. Tämä ohjelmavirhe esiintyy hiiren osoittimen lievänä välkkymisenä ”vedä ja pudota”-toiminnallisuutta käytettäessä. Ohjelmavirhe luokiteltiin työn tilaajan toimesta lopulta niin mitättömäksi, että sen korjaus päätettiin jättää projektin aikataulun ulkopuolelle.

Ainoa vaatimus, mitä komponenttiin ei saatu toteutettua, oli rakenteisen informaation metatietojen esitys. Tämäkin vaatimus oli kartoitettu riskien joukkoon päädyttäessä JTree-pohjaiseen toteutukseen komponentin suunnitteluvaiheessa, jolloin ongelman vahvistuessa suunniteltiin metatietojen esittämiseen toinen lähestymistapa.

Komponentille toteutettiin integrointi- ja käyttöönottotestaukset, joissa ei osoittimen välkkymisvirheen ja muutaman puuttuvan rajapintametodin lisäksi havaittu enempää ohjelmavirheitä.

5.3 Aikataulut

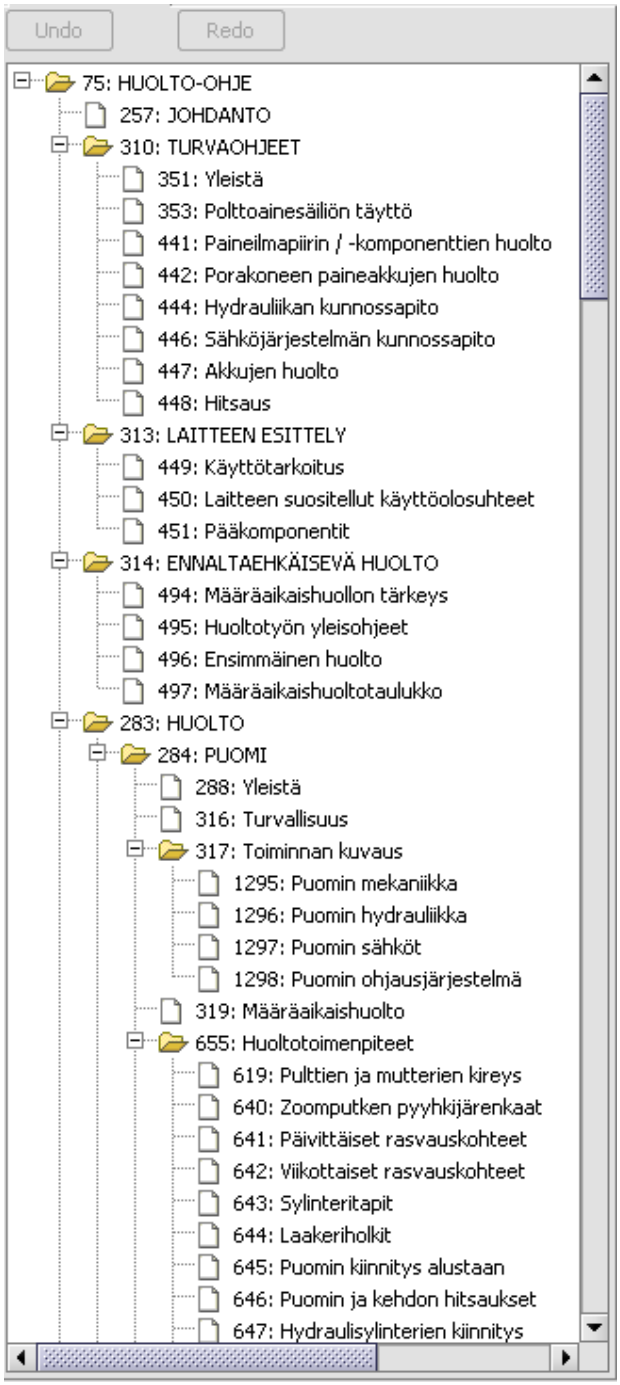
Aikataulu oli työssä hyvin tärkeä, sillä komponentti tuli pääasiallisesti DokuMentori Oy:n DokuPort™-ohjelmiston rakenteisen informaation hallintatyökaluksi. Ilman kunnollista hallintakomponenttia ohjelman toiminnot olisivat olleet rajoitetut ja tiedon esittäminen olisi ollut hankalampaa. Siten komponentti oli tärkeää saada integroiduksi määrättyyn aikaan.

Komponentin suunnittelu oli tosin kestänyt hieman odotettua pidempään, joten toteutusta aloitettaessa oli kiire. Tähän oli kuitenkin varauduttu, sillä kuten tyypillistä, ohjelmistoprojektien aikatauluilla on taipumusta venyä syystä tai toisesta. Komponentin toteutus kuitenkin vei odotettua vähemmän aikaa, jolloin komponentti valmistui edellä aikataulua ja viimeistelyyn jäi aikaa. Ohjelmakoodin perinpohjainen kommentointi ja testaus sai ylimääräistä aikaa, joka vaikutti suoraan varsinaisten testaajien työhön ja jatkokehittäjien mielekkyyteen integroida komponentti osaksi suurempaa ohjelmistoa.

Projekti komponentin parissa kesti yhteensä viisi kuukautta. Ensimmäiset kaksi kuukautta kului suunnitteluun ja viimeiset kolme kuukautta toteutukseen, testaukseen ja integrointiin.

5.4 Lopputulos

Työn tulos oli käyttökelpoinen rakenteisen informaation hallintaan tarkoitettu komponentti, jolla on laajat käyttömahdollisuudet. Komponentti valmistui ajallaan ja kaikki osapuolet olivat tyytyväisiä lopputulokseen. Komponenttia käytetään parhaillaan DokuMentori Oy:n DokuPort™ -sovelluksessa rakenteisen informaation hallintatyökaluna. Seuraavalla sivulla on kuvakaappaus valmiista komponentista esittämässä rakenteisen dokumentin rakennetta DokuPort™-sovelluksessa (kuva 18). Komponentille oli myös jatkokehitysideoita, eli sen kehitys jatkuu lähitulevaisuudessakin. Näistä jatkokehitysideoista kerrotaan lisää luvussa 5.5.



Kuva 18 Kuvakaappaus valmiista komponentista esittämässä rakenteisen dokumentaation rakennetta DokuPort™-sovelluksessa.

5.5 Jatkokehitys

Jatkokehitystä on suunnitteilla komponentin ”kumoa/tee uudelleen”-toiminnallisuutta koskien. Kyseisen toiminnallisuuden ratkaisu oli hieman tehoton, koska alkuperäinen vaatimus oli luoda yksitasoinen ”kumoa/tee uudelleen”-toiminnallisuus. Tämä tarkoittaa sitä, että vain yksi käyttäjän suorittama operaatio olisi kumottavissa tai uudelleentehtävissä. Vaatimuksen mukaan suunniteltiin yksinkertainen toiminnallisuus, jossa komponentin tietomalli talletettiin pinoon tiettyjen käyttäjän suorittamien operaatioiden ohessa ja haluttaessa palautettiin operaatiota edeltävä tietomalli. Komponentin toteutuksen edetessä päädyttiin tekemään ”kumoa/tee uudelleen”-toiminnallisuudesta monitasoinen, jolloin pinoon voitaisiin laittaa määräämätön määrä operaatioita.

Kuten edellä mainittiin, vaikka tietomallin tallennus pinoon oli helppo ja nopea toteuttaa, on se silti tehoton ja kömpelö ratkaisu. Koska toiminnallisuuden jatkokehitys ei mahtunut enää projektin alkuperäiseen aikatauluun, asetettiin se jatkokehitysideoihin. Uusi suunnitelma toiminnallisuudelle sisältää eräänlaisen implementaation Command-ratkaisumallista. Siinä jokaiselle komponentissa toteutettavalle toiminnalle on oma toimintaluokkansa. Jokainen toimintaluokka sisältää toiminnan ja käänteistoiminnan. Tiettyjen käyttäjän suorittamien toimintojen seurauksena pinoon asetetaan tietomallin sijasta toimintaluokkien olioita, joita voidaan haluttaessa myös poistaa sieltä. Toimintaluokan olio sisältää tiedon siitä, onko toteutettavissa toiminta vai käänteistoiminta. Näin komponentille suoritetaan aina operaatioita sen sijaan, että asetettaisiin koko tietomalli uudelleen. Tämä on nopeampaa ja lisäksi säästää keskusmuistia.

Toinen jatkokehitysidea on rakenteisen informaation metatietojen esitys, joka kartoitettiin riskien joukkoon JTree-toteutustapaa valittaessa. Toiminnallisuutta ei saatu toteutettua vaatimusten mukaisesti, mutta sen sijaan valittiin vaihtoehtoinen lähestymistapa. Tämä lähestymistapa on toteuttaa TreeView-luokkaan toinen kehitysrajapinta DNDTree-luokan lisäksi, jonka avulla metatiedot voidaan esittää. Koska tämän lähestymistavan toteutus meni projektin aikataulun ulkopuolelle, lisättiin se kehitysideoihin.

Myös komponentin tuotteistamisesta tuli jatkokehitysidea. Koska DokuMentori Oy:llä oli kiire saada komponentti integroitua omaan rakenteisen informaation sisällönhallintajärjestelmäänsä, jäi ajatus tuotteistamisesta jatkokehitysideaksi. Koska komponentin rajapinnat ovat huolella toteutettuja ja ulkonäkö yksilöitävissä, olisi mahdollisia ostajia komponentille kaikki ohjelmistoyritykset, joilla on tarve esittää ja hallita monipuolisesti rakenteista informaatiota.

Muita jatkokehitysideoita ei työn loppuessa ollut.

6 YHTEENVETO

Komponentin suunnittelu oli hyvin aikaa vievä prosessi, joka maksoi itsensä takaisin komponentin toteutusvaiheessa. Komponentin toteutus onnistui sovittujen aikataulujen mukaisesti ja siihen saatiin kaikki vaaditut ominaisuudet informaation metatietojen esitystä lukuun ottamatta. Voidaankin sanoa, että komponentin toteutuksen onnistuminen oli suurelta osin tarkan suunnittelun ansiota. Komponenttiin saatiin myös mielenkiintoisia lisäkehitysideoita, joita suunnitellaan toteutettaviksi lähitulevaisuudessa.

Toteutettu komponentti on tällä hetkellä käytössä DokuMentori Oy:n DokuPort™-ohjelmistossa. Kokonaisuudessaan projekti komponentin parissa onnistui hyvin ja vastasi sisällöltään hyvin tyypillistä ohjelmistoinsinöörin toteutusprojektia.

Työn aikana opin pitämään suunnittelun tasapainossa varsinaisen toteuttamisen kanssa. Mikäli aikaa olisi ollut vielä enemmän, olisi tiettyjä toiminnallisuuksia voitu suunnitella tarkemmin ja tehokkaammiksi. Näin olisi vältetty myöhemmät korjaukset, kuten esimerkiksi ”kumoa/tee uudelleen” –toiminnallisuuden tapauksessa. Vaikka komponentin toteutus Java-kehitysympäristön JTree-komponentista periyttämällä asettikin tiettyjä rajoituksia, oli päätös mielestäni silti oikea: JTree-komponentin sisäinen arkkitehtuuri pakotti suunnittelun ja toteutuksen tiettyjen kaavojen ja ratkaisumallien mukaiseksi, joka puolestaan lisäsi komponentin yhteensopivuutta ja laajennettavuutta runsaasti. Mikäli komponentti olisi toteutettu kokonaan alusta asti, olisi moni hyvä arkkitehtuuri- ja ratkaisumalli jäänyt oppimatta ja huomioimatta. Opin myös, että ratkaisumallien käyttäminen suurempien ohjelmistoprojektien yhteydessä on suositeltavaa, sillä ne helpottavat kokonaisuuksien hahmottamista sekä vähentää virheiden mahdollisuuksia ja niiden jäljitystä.

LÄHDELUETTELO

Painetut lähteet

1 Englander, Robert, Developing Java Beans. O'Reilly 1997. 1. luku: Introduction

Painamattomat lähteet

2 Koskentausta, Kivioja, DokuMentorin työhönotto-opas 2005. Sivu 2.

Sähköiset lähteet

3 Vanhala-Nurmi Vuokko, Koulutusaineistoa (Helia). [www-sivu]. [viitattu 24.4.2006] Saatavissa: http://myy.helia.fi/~vanvu/html/html.html

4 Pohlhau, William, William Pohlhau's Homepage [www-sivu]. [viitattu 24.4.2006] Saatavissa: http://www17.homepage.villanova.edu/william.pohlhaus/is/doc/Java%20Look%20and%20Feel.htm

Liite 1: Komponentin järjestelmäarkkitehtuuri