JouTo

Toimittajan työväline

Suunnitteludokumentti

Versio 1.11

16.3.2001

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Ohjelmistotuotantoprojekti, kevät 2001, ryhmä 3

Projektiryhmä:	Petri Hellemaa (PH)
	Antti Koskinen (AK)
	Lari Laine (LL)
	Jarno Saarto (JS)
	Eija Teitto (ET)
	Harri Tuuloskoski (HT)
Ohjaaja:	Kim Viljanen
Vastuuhenkilö:	Raine Kauppinen
Asiakkaat:	Helena Ahonen-Myka
	Greger Lindén

Dokumentin versiohistoria

0.1	19.02.2001	Ensimmäinen hahmotelma (AK)
0.2	23.02.2001	Lisätty arkkitehtuurikuvaus sekä tyyliohje (AK)
0.3	26.02.2001	Korjattu yleiskuvaus ja arkkitehtuurikuvausta (AK)
0.9	5.3.2001	Kasattu kokonaisuudeksi (AK)
1.0	9.3.2001	Korjattu virheitä ja yhdistelty irrallisia kappaleita (AK)
1.1	12.3.2001	Yhdistelty kokonaisuuksia
1.11	16.3.2001	Kirjoitusvirheiden korjauksia

Tästä dokumentista vastaa Antti Koskinen (AK).

Sisältö

1 Johdanto (AK) 1

1.1 Määritelmät, termit, lyhenteet 1

1.2 Viitteet 2

1.3 Dokumentin rakenne 2

2 Yleiskuvaus (AK,ET,HT) 3

2.1 Sovellusalueen kuvaus 3

2.2 Laitteistoympäristö (ET,HT) 3

2.3 Ohjelmistoympäristö (ET,HT) 3

2.4 Toteutuksen keskeiset reunaehdot 4

3 Ohjelmiston arkkitehtuurikuvaus (PH, ET ja HT) 4

3.1 Yleinen arkkitehtuurikuvaus 4

3.2 Tarkempi arkkitehtuurikuvaus 5

3.3 Palvelimen yksityiskohtainen arkkitehtuurikuvaus 6

3.4 Luokkien väliset suhteet 7

3.5 Tietokannan kuvaus (JS) 9

3.5.1 Taulu alias 9

3.5.2 Taulu document 11

3.5.3 Taulu dtd 11

3.5.4 Taulu journalist 12

3.6 Leikepöytätoiminnot 12

3.7 Peruutustoiminto 12

4 Luokkakuvaukset 14

4.1 UIManager (JS) 14

4.1.1 Rajapinnat (JS) 15

4.1.2 Attribuutit 15

4.1.3 Metodit (JS) 16

4.2 TreeManager (PH) 21

4.2.1 Attribuutit 22

4.2.2 Metodit 23

4.3 TreeNode (PH) 27
4.3.1 Attribuutit 28
4.3.2 Metodit 28
4.4 PCData (PH) 28
4.4.1 Attribuutit 29
4.4.2 Metodit 29
4.5 Element (PH) 29
4.5.1 Attribuutit 29
4.5.2 Metodit 30
4.6 Block (PH) 30
4.6.1 Attribuutit 30
4.6.2 Metodit 30
4.7 CommentBlock (PH) 31
4.8 Attribute (PH) 31
4.8.1 Attribuutit 31
4.8.2 Metodit 31
4.9 XMLAssistant (PH) 31
4.9.1 Metodit 32
4.10 ANode (PH) 32
4.10.1 Attribuutit 32
4.10.2 Metodit 33
4.11 BodyPanel (PH) 34
4.11.1 Attribuutit 34
4.11.2 Metodit 34
4.12 Body (PH) 34
4.12.1 Attribuutit 35
4.12.2 Metodit 36
4.13 ElementPopup (PH) 38
4.13.1 Metodit 39
4.14 AttributesDialog (PH) 39
4.14.1 Attribuutit 39
4.14.2 Metodit 39

4.15 AreaSelection (PH) 39
4.15.1 Attribuutit 40
4.15.2 Metodit 40
4.16 TreeSelection (PH) 40
4.16.1 Attribuutit 40
4.16.2 Metodit 41
4.17 ClipboardManager (PH) 41
4.17.1 Attribuutit 41
4.17.2 Metodit 42
4.18 Communications 42
4.19 CommunicationsException 44
4.20 loadArticle 44
4.21 loadDTD 44
4.22 saveArticle 45
4.23 listArticles 45
4.24 listDTD 45
4.25 Database 45

5 Käyttöliittymä(HT ja ET) 47
5.1 Käytettävät tekniikat(HT) 47
5.2 Käyttöliittymän elementit 47
5.2.1 Valikot 47
5.2.2 Tool Bar (ET,HT) 50
5.3 Metatieto 52
5.3.1 Artikkeli 53
5.3.2 Status bar 55
5.4 Käyttöliittymän dialogit (HT,ET) 55
5.4.1 Dialogien kuvaukset (ET) 56
5.4.2 Dialogien tekninen toteutus (HT) 58

A Liite: Osajärjestelmien rakennekuva 63

1 Johdanto (AK)

Tässä suunnitteludokumentissa esitellaan JouTo-ohjelmistotuotantoprojektin tuotteena syntyvä uutistoimittajan työväline. JouTo on lyhenne englannin kielen sanoista *"A Journalist's Tool"*.

1.1 Määritelmät, termit, lyhenteet

CSS Cascading Style Sheets, W3C:n kehittämä, suosittu HTML- ja XML-dokumenttien ulkoasun määrittelyyn tarkoitettu kieli.
Lisätietoja: `http://www.w3.org/Style/CSS/`

DoReMi Document Management, Information Retrieval, Text and Data Mining. TKTL:n tutkimusryhmä, joka vetää TYTTI-projektia.
Lisätietoja: `http://www.cs.helsinki.fi/research/doremi/`

DTD Document Type Definition, rakenteisten dokumenttien rakenteen määrittely.

EPS Encapsulated PostScript, kehystetty versio PostScript-kielisestä ohjelmasta. Ohjelmatiedostoon lisätty kehys määrittää rajat tiedoston sisällölle ja mahdollistaa sen sijoittamisen haluttuun kohtaan sivua.

HTML Hypertext Markup Language, WWW-sivujen sisällön kuvauksessa käytetty koodaus.
Lisätietoja: `http://www.w3.org/pub/WWW/MarkUp/`

IPTC International Press Telecommunications Council, kansainvälinen yhteistyöelin, joka organisoi uutisten määrittelyä ja ryhmittelyä.
Lisätietoja: `http://www.iptc.org/`

Java Sunin kehittämä ohjelmointikieli ja järjestelmäriippumaton ajoympäristö.
Lisätietoja: `http://www.sun.com/java/`

Javadoc Sunin kehittämä java-koodin dokumentointikieli
Lisätietoja: `http://www.sun.com/java/`

Java SDK Sunin kehittämä Java-ohjelmointiympäristö.
Lisätietoja: `http://www.javasoft.com/products/`

JDBC Java Database Connectivity, Java-kielen rajapinta tietokantojen käyttöön.

NITF News Industry Text Format, toimittajien kehittämä ja standardoima XML-pohjainen uutisartikkelien määrittelykieli. Lisätietoja: `http://www.nitf.org`

PostScript Adoben kehittämä suosittu sivunkuvauskieli. Tulostettava sivu voidaan lähettää PostScript-kirjoittimelle tiiviinä PostScript-ohjelmana tilaavievän raakadatan sijaan.

TKTL Helsingin yliopiston tietojenkäsittelytieteen laitos.
Lisätietoja: `http://www.cs.helsinki.fi`

TYTTI Tietotyöläisen työasema. Vuosina 2000-2002 käynnissä oleva tutkimusprojekti Helsingin yliopiston tietojenkäsittelytieteen laitoksella, rahoittajina TEKES sekä viisi suomalaisyritystä: Nokia, Alma Media, Sanoma-WSOY, Vaisala ja Lingsoft. TYTTI-projektin vastuuhenkilö Helena Ahonen-Myka ja projektipäällikkö Greger Lindén toimivat JouTo-projektin asiakkaina.
Lisätietoja: `http://www.cs.helsinki.fi/research/doremi/TYTTI/index.html`

W3C World Wide Web Consortium, kansainvälinen yhteistyöelin, joka koordinoi WWW-tekniikan ja -standardien kehitystä.
Lisätietoja: `http://www.w3.org/pub/WWW/Consortium/`

WWW World-Wide Web, Internetissä toimiva palvelu, joka mahdollistaa tekstin ja kuvien jakamisen helposti ja edullisesti verkon lukemattomille käyttäjille. WWW:ssä aineisto jaetaan sivuihin, jotka sijoitetaan WWW-palvelimelle tavallisina tiedostoina ja hakemistoina.

XML Extensible Markup Language, W3C:n kehittämä rakenteisten dokumenttien määrittelykieli.
Lisätietoja: `http://www.w3.org/XML/`

XSL Extensible Stylesheet Language, W3C:n kehittämä XML-dokumenttien ulkoasun määritykseen tarkoitettu kieli.
Lisätietoja: `http://www.w3.org/Style/XSL/`

1.2 Viitteet

NITFDTD News Industry Text Format DTD version 2.5 8/2000,
`http://www.nitf.org`

HTTP-protokolla Hypertext Transfer Protocol – HTTP/1.1,
`ftp://ftp.isi.edu/in-notes/rfc2616.txt`

1.3 Dokumentin rakenne

Ensimmäinen luku, Johdanto, esittelee tämän dokumentin rakenteen. Luvussa käydään läpi myös dokumentissä käytettävät termit, lyhenteet ja määritelmät.

Toinen luku kertoo ohjelmiston yleiskuvauksen. Luvussa käydään läpi ohjelman laitteisto ja ohjelmistoympäristöjä.

Kolmannessa luvussa on käsitelty ohjelman arkkitehtuurikuvaus. Kuvaus käydään läpi usealla eri tasolla. Tässä luvussa käydään läpi myös ohjelman eri luokkien väliset suhteet.

Neljäs luku, Luokkakuvaukset, käy läpi yksityiskohtaisesti ohjelman eri luokat ja niiden kuvaukset.

Viidennessä luvussa esitellään käyttöliittymän suunnittelu. Luvussa käydään läpi käyttöliittymän eri osat ja kuvataan ne.

2 Yleiskuvaus (AK,ET,HT)

2.1 Sovellusalueen kuvaus

JouTo-ohjelmistotuotantoprojekti on keväällä 2001 Helsingin yliopiston tietojenkäsittelytieteen laitoksella järjestettävän Ohjelmistotuotantoprojekti-kurssin näytetyö.

Työ liittyy osittain myös laitoksella käynnissä olevaan DoReMi-tutkimusryhmän "TYTTI- Tietotyöläisen työasema"-projektiin, jossa on mukana rahoittajina joukko suomalaisia teknologiayrityksiä ja Teknologian kehittämiskeskus. JouTo-projektin tarkoituksena ei kuitenkaan ole valmistaa valmista tuotetta kaupalliseen tarkoitukseen. Tuotettava ohjelmisto toimii prototyyppinä, jota voidaan kehittää myöhemmin lisää seuraavissa Ohjelmistotuotanto-projekteissa.

2.2 Laitteistoympäristö (ET,HT)

Ohjelmiston arkkitehtuuri on asiakas-palvelin pohjainen. Asiakaskoneissa pyöritettävä sovellus vaatii Pentium-teholuokan koneen, jossa on toimiva tietoliikenneyhteys. Palvelimen puolen tehovaatimukset on esitetty määrittelydokumentissa kappaleessa 7.2. *Palvelimen suorituskyky.* Toimintaynpäristöksi oletetaan toimituksen sisäinen verkko.

2.3 Ohjelmistoympäristö (ET,HT)

Ohjelmistoa käytetään WWW-selaimen kautta. Ohjelmiston asiakkaan puolen käyttöliittymä, editori, toteutetaan WWW-selaimessa toimivana graafisena Java applettina. Appletissa käytetään Javan version 1.2 ominaisuuksia ja valmiita komponentteja, kuten Swing-kirjastoa. Lisäksi käytetään JouTo-projektin ulkopuolella toteutettuja Java-kirjastoja, Xerces XML-parseria ja Mark Wutkan tekemää DTD-parseria. Näiden lisäksi ohjelmistossa luodaan omia tietorakenteita sekä XML-puun käsittelyä että varsinaisen artikkelin muokkausta varten.
Artikkelit ja artikkelipohjat tallennetaan tietokantaan XML-muodossa, josta ne pystytään myös hakemaan jatkomuokkausta varten. Tietokanta-operaatioista ja tietokannan käsittelystä huolehtivat JouTo-projektissa luodut Java Servlet-komponentit. Nämä komponentit käyttävät JDBC-rajapintaa tietokannan käsittelyssä. JDBC-tietokantarajapinta kuuluu osana Java 1.2 - standardiin.

Palvelinpuolen servlettien ja asiakkaanpuolen appletin välillä tietoja välittää tietoliikenne-komponentti. Tämä komponentti luodaan JouTo-projektin toimesta.

2.4 Toteutuksen keskeiset reunaehdot

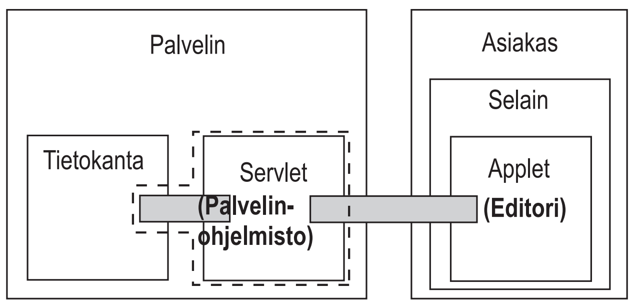
Ohjelmiston tulee toimiva Helsingin yliopiston tietokonejärjestelmässä sekä Linux- että Windows-ympäristöissä. Ohjelmiston on oltava valmis viimeistään 24.4.2001, jolloin se palautetaan asiakkaille ja ohjaajille.

3 Ohjelmiston arkkitehtuurikuvaus (PH, ET ja HT)

Tässä kappaleessa selvitetään ohjelmiston arkkitehtuuria. Kappale 3.1 kuvaa ohjelmiston arkkitehtuuria yleisellä tasolla. Kappale 3.2 kuvaa arkkitehtuuria tarkemmalla tasolla tietovuokaavioiden avulla. Kappaleessa 3.3 kuvataan palvelimen puolen arkkitehtuuri yksityiskohtaisesti. Kappaleessa 3.4 kuvataan järjestelmän luokat ja niiden väliset suhteet. Tässä kappaleessa kuvataan myös asiakaspuolen yksityskohtainen arkkitehtuuri.

3.1 Yleinen arkkitehtuurikuvaus

Ohjelmiston arkkitehtuuri on asiakas-palvelin pohjainen. Asiakkaan puolen ohjelmisto on WWW-selaimessa toimiva Java-sovelma (applet), ja palvelimen puolen ohjelmisto (servlet) huolehtii tietojen tallennuksesta sekä latauksesta tietokannasta.

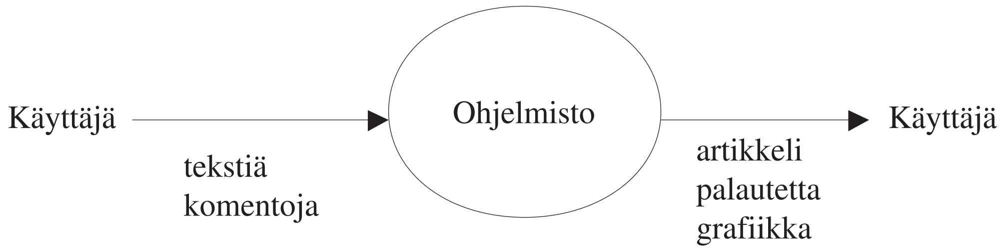


Kuva 1: Ohjelmiston arkkitehtuuri

Kuvassa 1 nähdään ohjelmiston arkkitehtuuri yleisellä tasolla.

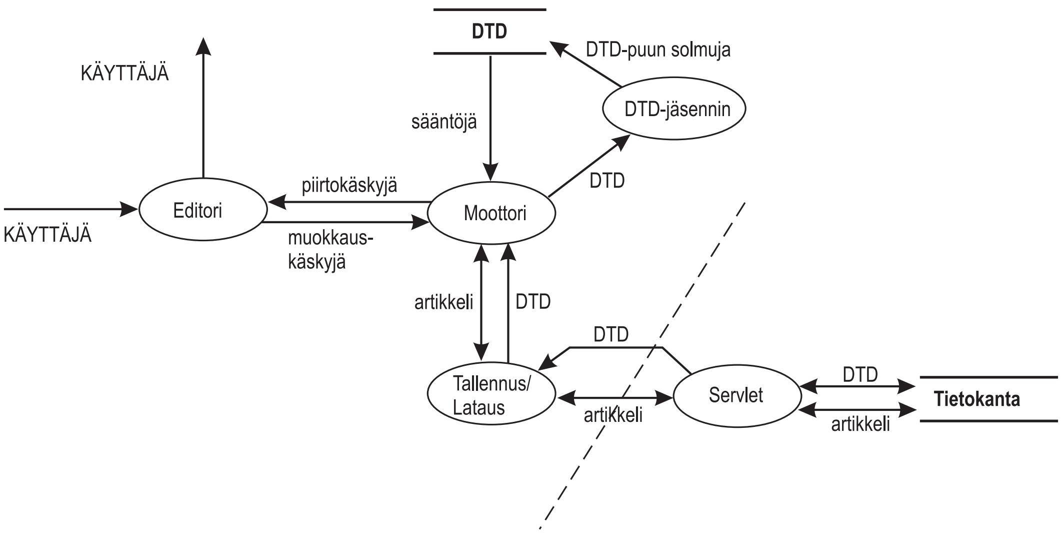
3.2 Tarkempi arkkitehtuurikuvaus

Tässä kappaleessa kuvataan ohjelmiston arkkitehtuuria hieman tarkemmin kuin kappaleessa 3.1



Kuva 2: Ohjelmiston tietovuokaavio - 1. taso

Kuvassa 2 nähdään ohjelmiston toiminta yleisellä tasolla (tietovuokaavion 1. taso).



Kuva 3: Ohjelmiston tietovuokaavio - 2. taso

Kuvassa 3 nähdään ohjelmiston komponenttien ja erilaisten tietovarastojen sijoittelu ja niiden keskinäinen toimintalogiikka.
Käyttäjä käyttää editoria artikkelien käsittelyyn. Editori on toteutettu Java-sovelmalla (applet), joka toimii käyttäjän WWW-selaimessa. Käyttäjän ohjelmisto on jakautunut neljään osa-alueeseen, moottoriin, editoriin, DTD-jäsentimeen sekä tallennus/lataus-osaan.
Editori-osa huolehtii käyttöliittymän luomisesta käyttäjälle, sekä käyttäjän syötteiden välittämisestä eteenpäin moottorille. Editori-osa muokkaa käyttöliittymän valitun artikkelin mukaisesti ja huolehtii että käyttäjä voi luoda vain valitun artikkelin mukaisia rakenteita. Editori-osa toimii rajapintana käyttäjän ja varsinaisen

ohjelmiston toiminnan välillä.
DTD-jäsennin huolehtii valitun DTD:n tulkitsemisesta ja sääntöverkon luomisesta. Tämän verkon avulla valitun artikkelin oikeellisuutta pystytään tarkkailemaan ja käyttöliittymän avulla ohjaamaan.
Tallennus/lataus-osa huolehtii kommunikoinnista palvelimen puolen ohjelmiston (servlet) kanssa. Tallennus/lataus-osan rajapinnan kautta suoritetaan kaikki lataus- ja tallennusoperaatiot.
Moottori huolehtii ohjelman toimintalogiikasta ja muiden osien ohjaamisesta. Moottori huolehtii muiden osien keskinäisestä kommunikoinnista sekä suorittaa tarvittavat operaatiot ohjelman ajamiseksi.

Servlet huolehtii artikkelien ja DTD:n latauksesta, tallennuksesta ja poistamisesta tietokannasta. Servlet on toteuttu Java-sovelluksena, joka pyörii palvelimen Java-moottorin (Java-engine) avulla palvelimen prosessina. Servlet ottaa vastaan editorista tulleita syötteitä, tekee tarvittavat toimenpiteet palvelimen puolella ja palauttaa tulosteet takaisin editorille. Servlet on ainoa komponentti, joka pääsee käsittelemään tietokannan sisältöä. Servlet tarjoaa appletille rajapinnan tietokanta-operaatiohin. Näin tietokannan rakenne pysyy eheänä ja toteutus piilossa ulkopuolisilta elementeiltä.

Tietokantaan on tallennettuna eri DTD:t, DTD:n mukaiset artikkelit, eri käyttäjien tiedot sekä muut JouTo-ohjelmiston tarvitsema informaatio. Tietokanta on Oracle 8-pohjainen ja sijaitsee Helsingin Yliopiston atk-keskuksen *kontti*-tietokantapalvelimella. Tietokantaan on luotu tarvittavat taulut ja oikeudet, jotta JouTo-ohjelmistoa pystyy ajamaan.
Tarkempaa tietoa tietokannasta ja sen rakenteesta löytyy kappaleesta 3.5

3.3 Palvelimen yksityiskohtainen arkkitehtuurikuvaus

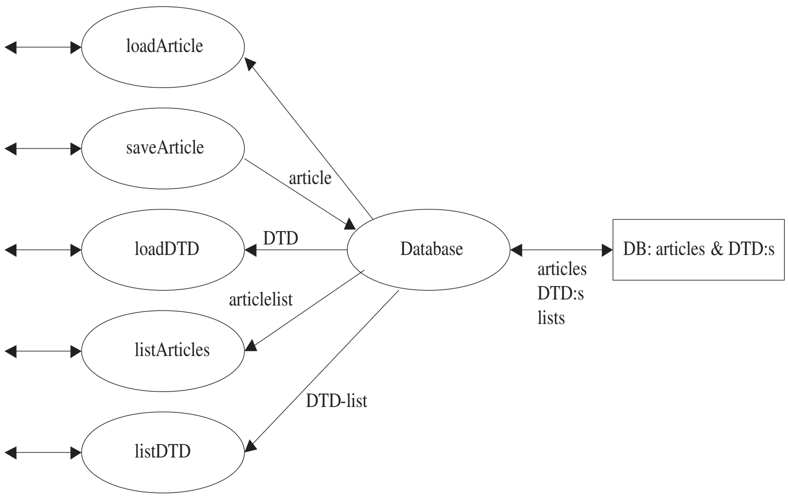
Kuvassa 4 nähdään palvelimen ohjelmiston komponenttien ja erilaisten tietovarastojen sijoittelu ja niiden keskinäinen toimintalogiikka.

Database hoitaa rajapinnan ja liikennöinnin tietokantaan.

loadArticle huolehtii halutun artikkelin lataamisesta tietokannasta. Tämän servletin kautta hoidetaan artikkelien siirtäminen asiakassovellukselle palvelimelta. Jos tietokantaoperaatio epäonnistuu palautetaan virheilmoitus

loadDTD huolehtii halutun DTD:n lataamisesta tietokannasta. Tämän servletin kautta hoidetaan DTD:iden siirtäminen asiakassovellukselle palvelimelta. Jos tietokantaoperaatio epäonnistuu palautetaan virheilmoitus

saveArticle huolehtii asiakassovellukselta tulleen artikkelin tallennuksesta tietokantaan. Servletti pitää myös yllä muita artikkeliin liittyviä kenttiä tietokannassa.



Kuva 4: Palvelimen tietovuokaavio - 3. taso

Jos tietokantaoperaatio epäonnistuu palautetaan virheilmoitus

listArticles listaa tietokannassa olevista artikkeleista otsikkotietoja, jotka se lähettää asiakassovellukselle. Jos tietokantaoperaatio epäonnistuu palautetaan virheilmoitus

listDTD listaa tietokannassa olevista DTD:istä otsikkotietoja, jotka se lähettää asiakassovellukselle. Jos tietokantaoperaatio epäonnistuu palautetaan virheilmoitus

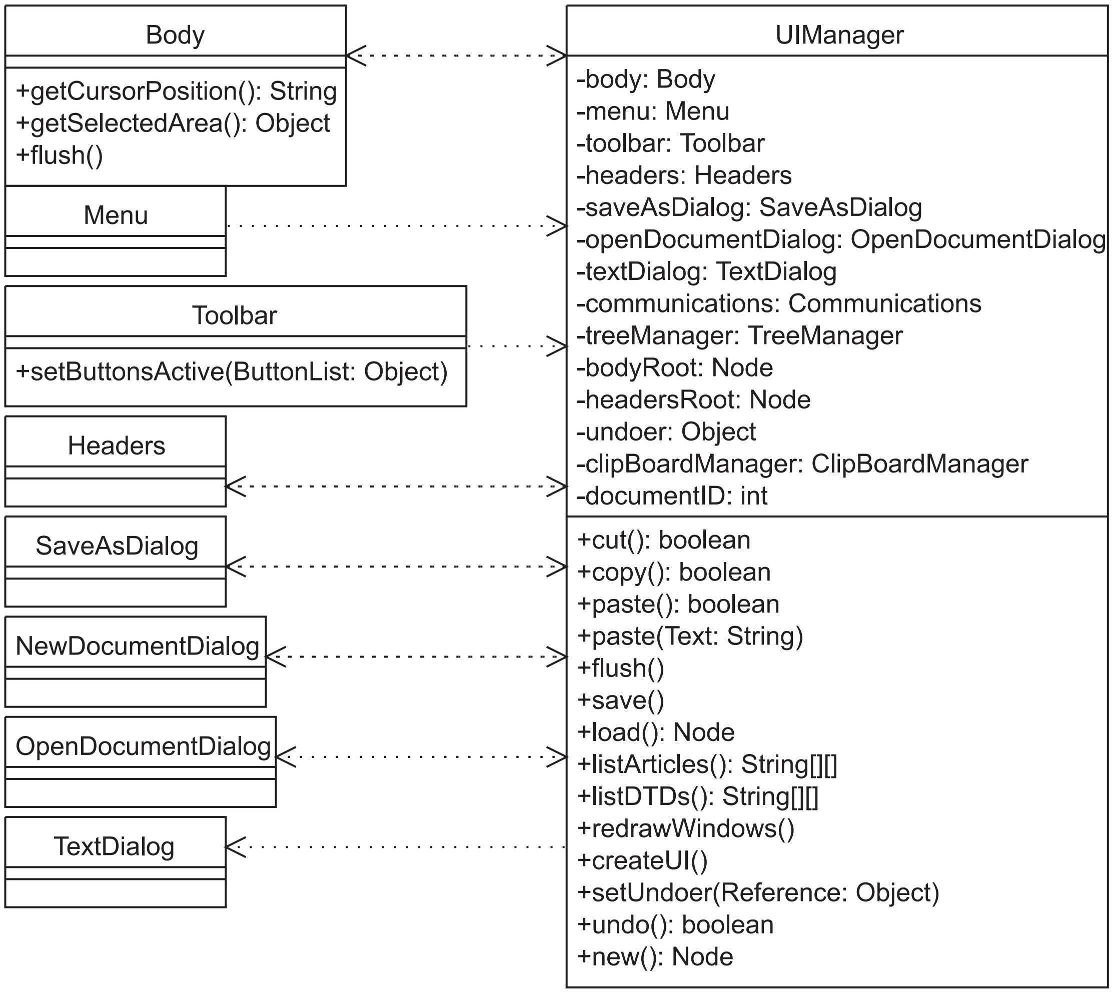
3.4 Luokkien väliset suhteet

Tässä kappaleessa kuvataan eri luokkien ja osajärjestelmien väliset suhteet ja riippuvaisuudet.

Kuvassa 28 (Liite A, sivulla 63) olevat moduulit voidaan jakaa toimintojensa puolesta kolmeen selkeään osaan. Nämä osakokonaisuudet ovat sisällön käsittelyyn keskittyneet moduulit, tiedon tallennukseen ja lataukseen keskittyneet moduulit sekä käyttäjän kanssa kommunikointiin keskittyneet moduulit.

Kuvasta 5 nähdään käyttäjän kanssa kommunikointiin keskittyneet luokat. Näistä rungon muodostaa UIManager-luokka, joka sisältää viitteet kaikkiin käyttöliittymän komponentteihin. UIManager toimii käyttöliittymän runkona. Se toteuttaa kaikki käyttöliittymältä saatavat komennot ja hoitaa myöskin käyttöliittymän piirtämisen.

Kuvasta 6 nähdään tietosisällön muokkaamiseen tarvittavia luokkia. Kuvassa näkyvä luokka TreeManager toimii ohjelman tietosisällön runkona ja samalla myös muodostettavan puun vastaavana elementtinä. TreeManagerista voidaan suorit-

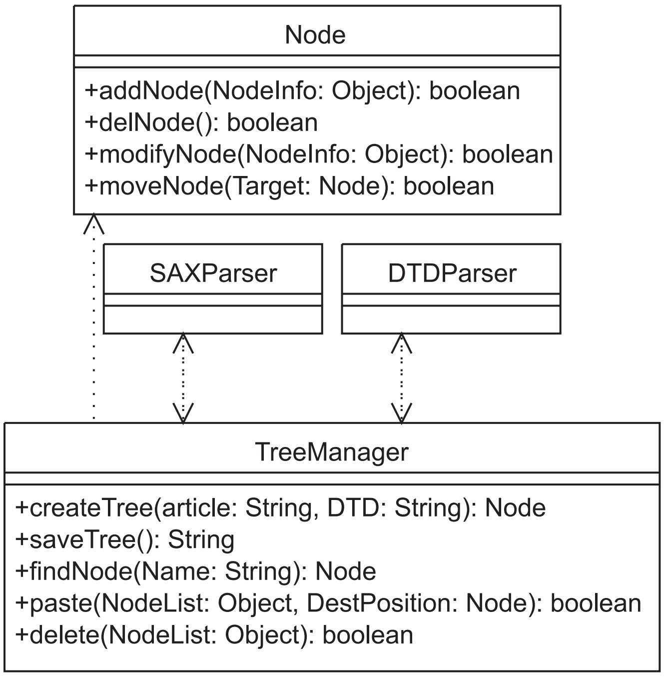


Kuva 5: Käyttöliittymän rakenne

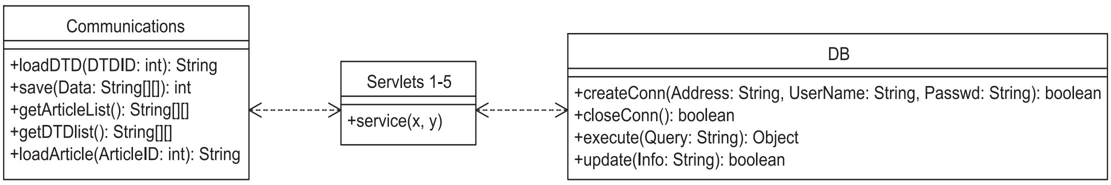
taa koko puuta koskevat yleiset operaatiot. Puu rakennetaan Node-elementeistä, jotka sisältävät yksilökohtaisia operaatioita, jotka kohdistuvat juuri tiettyyn kohtaan puuta. TreeManager on myös yhteydessä XMLParseriin sekä DTDWutka-parseriin, joita käytetään uutta puuta luotaessa.

Kolmannen osakokonaisuuden muodostaa kuvassa 7 olevat Communications-luokka sekä servletit ja Database-luokka. Nämä elementit vastaavat ohjelmiston tiedon fyysisestä tallennuksesta ja latauksesta.

Kuvassa 8 nähdään miten eri osakokonaisuudet liittyvät toisiinsa. UIManager toimii TreeManagerin toimintojen kutsujana ja tiedon välittäjänä. UIManager toimii myös kutsujana ja tiedonvälittäjänä tallennus- ja lataustapauksissa.



Kuva 6: Tietosisällön rakenne



Kuva 7: IO rakenne

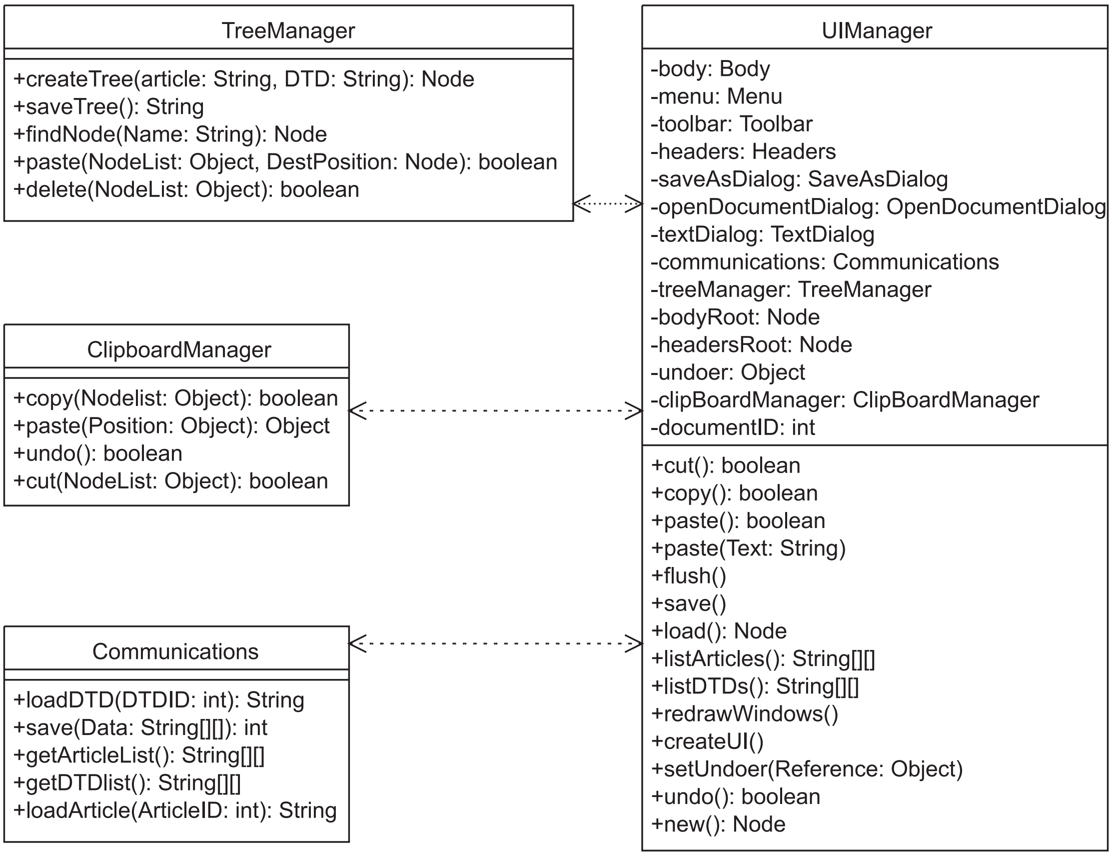
3.5 Tietokannan kuvaus (JS)

Järjestelmään kuuluva tietokanta rakentuu neljästä taulusta (Kuva 9): *alias, journalist, document ja dtd.* Jokaisessa taulussa on sarake *lid*, joka yksilöi rivin. Tämä sarake on myös jokaisen taulun pääavain. Tietokantaan kirjoittamisen ja sieltä lukemisen hoitaa yksinomaan palvelinohjelmisto.

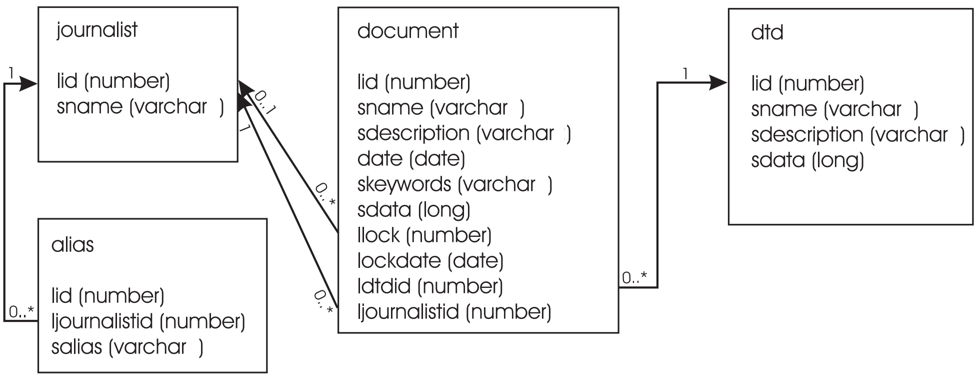
3.5.1 Taulu alias

Alias-tauluun tallennetaan toimittajien mahdolliset nimimerkit eli aliakset. Jokaisella toimittajalla voi olla rajattomasti aliaksia. Ohessa jokainen taulun sarake kuvauksineen.

Sarake:	**Tehtävä:**
lid	Pääavain (*number*), joka yksilöi aliaksen.



Kuva 8: Yleisrakenne



Kuva 9: Taulujen kuvaus

ljournalistid	Vierasavain (*number*), joka viittaa journalist-taulun pääavaimeen (*lid*). Sarakkeen avulla selviää kenen toimittajan alias on kyseessä.
salias	Alias tekstinä (*varchar*).

3.5.2 Taulu document

Tauluun *document* tallennetaan editorilla tehdyt dokumentit. Pelkän asiakirjan lisäksi tauluun tallennetaan monia tietoja muun muassa hakujen helpottamiseksi. Ohessa jokainen taulun sarake kuvauksineen.

Sarake:	**Tehtävä:**
lid	Pääavain (*number*), joka yksilöi aliaksen.
sname	Dokumentin nimi muodossa *varchar*.
sdescription	Pidempi selitys dokumentin sisällöstä (*varchar*).
date	Dokumentin viimeisin tallennusaika (muodossa *date*).
skeywords	Kuvaavia hakusanoja, jotka helpottavat hakua (*varchar*).
sdata	Sarake, johon itse dokumentti on tallennettu (*long*).
llock	Kirjoituslukko, joka toimii samalla vierasavaimena journalist -tauluun. Jos lukko on määritelty, näkee samalla kuka sen on varannut.
lockdate	Lukon asetusaika (muodossa *date*). Tätä tietoa käytetään varmistamaan, että lukko ei ole jäänyt päälle esimerkiksi tietoliikenneyhteyden katkeamisen vuoksi.
ldtdid	Vierasavain, joka osoittaa dtd-taulun pääavaimeen. Sarakkeesta käy ilmi minkä DTD:n mukaan asiakirja on luotu.
ljournalistid	Vierasavain (*number*), joka viittaa journalist-taulun pääavaimeen (*lid*). Tämän sarakkeen avulla selviää dokumentin edellinen tallentaja.
salias	Alias tekstinä (*varchar*).

3.5.3 Taulu dtd

Tauluun *dtd* tallennetaan dokumenttirakenteen määräävät DTD-tiedostot. Pääavaimen *lid* lisäksi tauluun tallennetaan DTD:n nimi (*sname*), kuvaus (*description*) ja itse DTD, joka tulee sarakkeeseen (*sdata*). Ohessa jokainen taulun sarake kuvauksineen.

Sarake:	**Tehtävä:**
lid	Pääavain (*number*), joka yksilöi DTD:n.
sname	DTD:n nimi *(varchar)*.
sdescription	Pidempi kuvaus DTD:stä *(varchar)*.
sdata	Sarake, johon itse DTD on tallennettu (*long*).

3.5.4 Taulu journalist

Tauluun *journalist* tallennetaan toimittajien tiedot. Taulussa on pääavaimen *lid* lisäksi *sname*, johon tallennetaan toimittajan nimi. Ohessa jokainen taulun sarake kuvauksineen.

Sarake:	**Tehtävä:**
lid	Pääavain (*number*), joka yksilöi DTD:n.
sname	Toimittajan nimi *(varchar)*.

3.6 Leikepöytätoiminnot

Tavallisesti appletit eivät pääse käsiksi käyttöjärjestelmän leikepöytään, ainakaan selaimissa. Microsoft jakelee kuitenkin pakettia, jolla oikeudet voidaan antaa Explorer-selaimessa toimivalle sovelmalle. JouTon editorin leikepöytätoiminnot rakennetaan varautuen mahdollisuuteen käyttää järjestelmän leikepöytää. Käyttöjärjestelmän leikepöydän avulla ainakin tekstimuotoista tietoa voitaisiin siirtää eri sovellusten välillä.

Leikepöytä toteutetaan Javassa *Clipboard*-oliona, jolle voidaan antaa sisällöksi *Transferable*-rajapintaa soveltava olio. JouTossa leikepöytätoimintoja hallinnoi erityinen *ClipboardManager*-luokka, joka sisältää Clipboard-olion. Tämän se luo käyttöjärjestelmän leikepöydäksi, mikäli mahdollista. ClipboardManager sisältää myös edellisen leikepöydän sisällön peruutustoimintoa varten. Olio kuuntelee leikepöytätoimintoja käskyttäviä *copy-*, *cut-* ja *paste*-painikkeita sekä *Edit*-valikon vastaavia valintarivejä.

ClipboardManager ottaa vastaan joko merkkijonomuotoista syötettä mistä tahansa editorin tekstikentästä tai puurakenteista syötettä dokumentin rungon muokkauskomponentin (*Body*-olion) käskyttämänä. Syöte haetaan viimeksi fokusoidusta komponentista. Tieto viimeksi aktivoituneesta komponentista on UIManagerilla, jolta ClipboardManager saa selville leikepöytäsyötteen lähteen. Jokaisen tekstikentän aktivoitumista täytyy kuunnella.

3.7 Peruutustoiminto

JouTo-editorissa on yhden askelen peruutustoiminto. Kaikki TreeManager-luokan kautta elementtipuuhun kohdistuvat muutokset ovat peruutettavissa. Myös *cut-* ja *paste*-leikepöytätoiminnot ovat peruutettavia. *Copy*-käskyä ei peruuteta, koska se muuttaa vain leikepöydän sisältöä. Leikepöytätoimintojen peruuttamisesta huolehtii ClipboardManager ja puun muutoksien peruuttamisesta TreeManager. Peruutuspainiketta ja Edit-valikon *undo*-valintaa kuuntelee kuitenkin UIManager, joka tietää, oliko edellinen muokkaustoiminto leikepöytätoiminto vai muu suoraan puun rakenteeseen kohdistettu muokkauskäsky.

Jokaisen cut- ja paste-toimenpiteen päätteeksi ClipboardManager suorittaa UI-Managerin *setUndoer*-metodin ja antaa viitteen itseensä. TreeManager toimii vastaavasti. Kun käyttäjä antaa peruutuskäskyn, ClipboardManager kutsuu *undoer*-olion *undo*-metodia.

Jokaisen peruutettavan muokkaustoiminnon yhteydessä ClipboardManager ja Tree-Manager päivittävät oman undo-muuttujansa, joka osoittaa sen metodin, joka suoritetaan, jos UIManager kutsuu undo-metodia.

Edellisen muutoksen peruuttamiseksi TreeManagerillä on muistissa kokonaisina viimeksi muutetut lapsielementtivektori ja attribuuttitaulukko. Pelkkiä olioviitteitä sisältäviä vektoreita ja taulukoita on nopea kopioida.

Kun leikepöydälle tuodaan uusi sisältö, se pitää kopioida. Vastaavasti paste-käskyn yhteydessä sijoitetaan kohteeseen leikepöydän sisällön kopio. Huomattavan suuria puurakenteita kopioitaessa saattaa esiintyä viivettä. *TreeNode*-olioiden *cloneTree*-metodi palauttaa solmusta alkavan alipuun kopion.

Yhden askelen peruutustoiminto tarkoittaa, että peruutustoiminnon peruuttaminen johtaa tilaan ennen edellistä peruutuskäskyä (*redo*). ClipboardManagerissä tämä tarkoittaa leikepöydän korvaamista viimeksi korvatulla leikepöydän sisällöllä. TreeManagerissä vastaavasti vaihdetaan viimeksi luotu lapsielementtivektori sen korvaamaan vektoriin tai viimeksi luotu attribuuttitaulukko sen korvaamaan taulukkoon.

Kun esimerkiksi poistetaan elementti, muutos kohdistuu sen isäelementin lapsivektoriin. Asetetaan *oldChildren* viittaamaan tähän lapsivektoriin, luodaan vektorin pohjalta kokonaan uusi lapsivektori jättämällä kohde-elementti välistä pois ja linkitetään isäelementti tähän uuteen lapsivektoriin. "Poistettua" elementtiä ei siis tuhottu, vaan isäelementin näkymää solmuihin rajoitettiin. Peruutustoiminto on nopea — tarvitsee vain vaihtaa keskenään *oldChildren*-viite ja isäelementin *children*-viite.

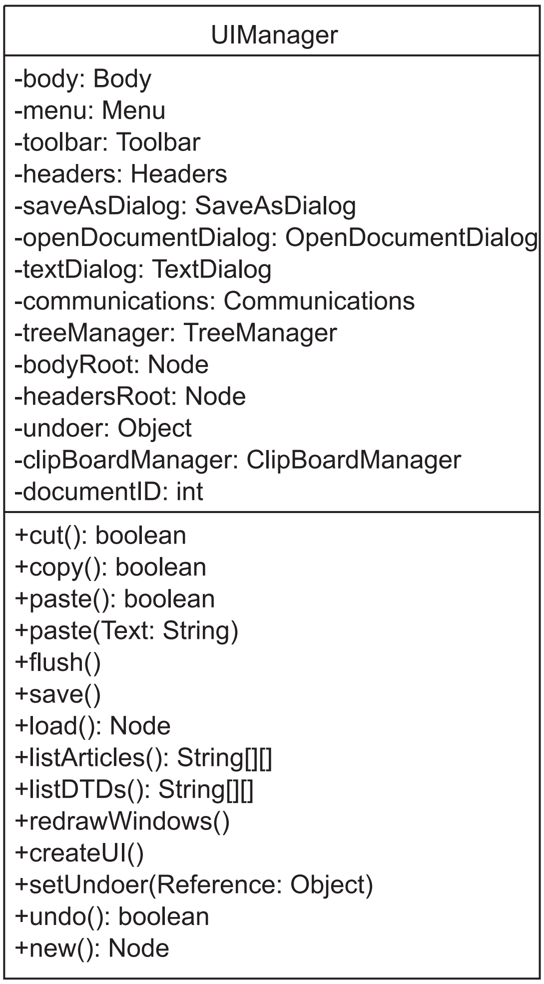
ClipboardManagerin cut- ja paste-toiminnot koostuvat myös puun muokkaustoiminnoista, joten peruuttavat metodit *uncut* ja *unpaste* kutsuvat TreeManagerin undo-metodia. cut- ja paste-metodeissa on pidettävä huolta siitä, että UIManagerille ilmoitetaan *undoer* puun muokkausmetodien kutsumisen jälkeen, koska niissä TreeManager ilmoittaa "peruuttajaksi" itsensä.

4 Luokkakuvaukset

Tässä luvussa esiteltään ohjelman eri luokat. Luokat kuvataan yksitellen. Jokaisesta luokasta kerrotaan sanallisesti luokan toimintaa, sekä luetellaan luokan attribuutit ja metodit lyhyine kuvauksineen.

4.1 UIManager (JS)

UIManager-olio käynnistää sovelluksen. Se toimii hallinnoivana luokkana, ja luo kahvat muihin luokkiin constructorissaan. Sen tehtävä on huolehtia käyttöliittymän toiminnasta ja dokumentin editoinnista. Käyttöliittymäoliot kutsuvat UIManageria ja käynnistävät toimintoja. UIManager jakautuu metodeihin, jotka käyvät ilmi kuvasta 10.



Kuva 10: UIManager-luokan metodit

4.1.1 Rajapinnat (JS)

Koska UIManager on muita luokkia hallinnoiva luokka, sillä on paljon rajapintoja. UIManagerista on rajapinta jokaiseen graafiseen komponenttiin, jota editorissa esiintyy. Näitä komponentteja ovat: Body, Menu, Toolbar, Headers, SaveAsDialog, NewDocumentDialog, OpenDocumentDialog ja TextDialog. Graafisten komponenttien lisäksi on rajapinnat Communications ja TreeManager -luokkiin. Koko järjestelmän rajapinnat käyvät ilmi kuvasta 28 sivulta 63.

4.1.2 Attribuutit

private Body body
Kahva Body-elementtiin.

private Menu menu
Kahva Menu-elementtiin.

private Toolbar toolbar
Kahva Toolbar-elementtiin.

private Headers headers
Kahva Headers-elementtiin.

private SaveAsDialog saveAsDialog
Kahva SaveAsDialog-elementtiin.

private OpenDocumentDialog openDocumentDialog
Kahva OpenDocumentDialog-elementtiin.

private TextDialog textDialog
Kahva TextDialog-elementtiin.

private Communications communications
Kahva Communications-elementtiin.

private TreeManager treeManager
Kahva TreeManager-elementtiin.

private ClipBoardManager clipBoardManager
Kahva ClipBoardManager-elementtiin.

private Node bodyRoot
Kahva ensimmäiseen Body-solmuun (juureen).

private Node headersRoot
Kahva ensimmäiseen Headers-solmuun (juureen).

private Object undoer
Tieto edellisen muutoksen tekijästä.

private int documentID
Tieto käsiteltävän dokumentin ID:stä.

4.1.3 Metodit (JS)

Luokan metodit kuvataan tässä osassa.

boolean cut()

parametrit: -

palauttaa: Boolean arvon, josta käy ilmi poiston onnistuminen

toiminta: Metodi kutsuu Body-metodin getSelectedArea-metodia, joka palauttaa valitun alueen. Se kutsuu ClipBoardManagerin cut-metodia, joka laittaa valitun rakenteen leikepöydölle. Valittu alue poistetaan myös käyttöliittymästä.

Hahmotelma metodin toiminnasta:

```
if (clipBoardManager.cut(body.getSelectedArea())){
   body.DeleteArea();
   return true;
}
return false;
```

boolean copy()

parametrit: -

palauttaa: Boolean arvon, josta käy ilmi kopioinnin onnistuminen

toiminta: Metodi kutsuu Body-metodin getSelectedArea-metodia, joka palauttaa valitun alueen, ja ClipBoardManagerin copy-metodia, joka kopioi tiedot leikepöydälle.

Hahmotelma metodin toiminnasta:

```
return clipBoardManager(body.getSelectedArea());
```

boolean paste()

parametrit: -

palauttaa: Boolean arvon, josta käy ilmi liitämisen onnistuminen

toiminta: Metodi selvittää valitun kohteen Bodyn getCursorPosition metodilla. Se kutsuu ClipBoardManagerin paste-metodia, joka yrittää leikepöydän sisällön valittuun kohtaan. Jos liittäminen onnistuu, metodi kirjoittaa tiedot myös käyttöliittymään.

Hahmotelma metodin toiminnasta:

```
if (clipBoardManager(body.getCursorPosition()){
   paintToBody(clipBoardData);
   return true;
}
return false;
```

void flush()

parametrit: -

palauttaa: -

toiminta: Metodi kutsuu kaikkia käyttöliittymäkomponenttien flush-metodeja. Metodit kirjoittavat puskureissa olevan datan tietorakenteeseen.

Hahmotelma metodin toiminnasta:

```
body.flush();
```

void save()

parametrit: -

palauttaa: -

toiminta: Metodi kutsuu aluksi flush-metodia ja sitten TreeManagerin toString-metodia, joka palauttaa tietorakenteen sisällön XML-muotoon muunnettuna. Se yhdistää nämä tiedot muihin käyttöliittymästä saataviin tietoihin, ja kutsuu Communications-luokan save-metodia, joka tallentaa tiedot. Käyttäjää informoidaan onnistumisesta.

Hahmotelma metodin toiminnasta:

```
flush();

String saveData[4]=new String(documentID,saveAsDialog.DTDID,
   saveAsDialog.description,treeManager.toString());

if (communications.save(saveData))
   statusBar.write("Document saved.");

textDialog.write("Save failed.");
```

Node load()

parametrit: -

palauttaa: Viitteen Bodyn ensimmäiseen solmuun.

toiminta: Metodi kutsuu aluksi Communications-luokan loadDTD ja loadArticle-metodeja. Nämä palauttavat DTD:n ja artikkelin String-muodossa. Tämän jälkeen metodi kutsuu TreeManagerin createTree-metodia, joka luo puun näistä tiedoista. CreateTree palauttaa viitteen ensimmäiseen solmuun. Lopuksi load päivittää käyttöliittymän.

Hahmotelma metodin toiminnasta:

```
documentID=loadDialog.documentID;
String DTD=Communications.loadDTD(loadDialog.DTDID);
String article=Communications.loadArticle(documentID);
bodyRoot=treeManager.createTree(article,DTD);
return bodyRoot;
```

String[][] listArticles()

parametrit: -

palauttaa: Artikkelilistan muodossa (ID,DTDID,Nimi,Selitys).

toiminta: Metodi listArticles hakee tietokannasta listan tallennetuista artikkeleista Communications luokan getArticleList-metodilla, ja palauttaa sen kutsujalle.

Hahmotelma metodin toiminnasta:

```
return communications.getArticleList();
```

String[][] listDTDs()

parametrit: -

palauttaa: DTD-listan muodossa (ID,Nimi).

toiminta: Metodi listArticles hakee tietokannasta listan tallennetuista artikkeleista Communications-luokan getDTDList-metodilla, ja palauttaa sen kutsujalle.

Hahmotelma metodin toiminnasta:

```
return communications.getDTDList();
```

void redrawWindows()

parametrit: -

palauttaa: -

toiminta: Piirtää kaikki näytöllä olevat ikkunat uudestaan tietorakenteen mukaisella datalla.

Hahmotelma metodin toiminnasta:

```
for each open dialog{
   killdialog;
   createdialog;
}
```

boolean undo()

parametrit: -

palauttaa: Boolean arvon, joka kertoo toiminnon onnistumisesta.

toiminta: Kutsuu ClipboardManagerin undo-metodia, joka palauttaa ennen edellistä muutosta olleen tilanteen.

Hahmotelma metodin toiminnasta:

```
if (clipBoardManager.undo())
   return reDrawWindows();
return false;
```

boolean setUndoer(Reference: Object)

parametrit: Viite kutsujaan.

palauttaa: -

toiminta: Asettaa viimeisen muutoksen tekijäksi parametrinaan saamansa olion attribuuttiin undoer.

Hahmotelma metodin toiminnasta:

```
undoer=Reference;
```

New load()

parametrit: -

palauttaa: Viitteen Bodyn ensimmäiseen solmuun.

toiminta: Metodi kutsuu aluksi Communications-luokan createDTD. Metodi palauttaa DTD:n String-muodossa. Tämän jälkeen metodi kutsuu TreeManagerin createTree-metodia, joka luo puun juuren näistä tiedoista. CreateTree palauttaa viitteen ensimmäiseen solmuun. Lopuksi load päivittää käyttöliittymän.

Hahmotelma metodin toiminnasta:

```
documentID=loadDialog.documentID;
String DTD=Communications.loadDTD(loadDialog.DTDID);
bodyRoot=treeManager.createTree();
return bodyRoot;
```

4.2 TreeManager (PH)

TreeManager-olio huolehtii yhden NITF-dokumentin hallinnasta kokonaisuutena. Se vastaanottaa dokumentin merkkijonona UIManagerin välityksellä Communications-oliolta ja rakentaa sen pohjalta elementtipuun. Vaihtoehtoisesti se voi rakentaa uuden artikkelin pohjan. Kun puu on luotu, dokumentin muokkaaminen tapahtuu TreeManagerin muokkausmetodien kautta. Metodit pitävät huolen siitä, että jokaisen muutoksen jälkeen dokumentti noudattaa rakennemäärittelyä. Tarvittaessa TreeManager palauttaa puun sisällön XML-muodossa.

TreeManager perii luokalta *org.xml.sax.helpers.DefaultHandler* rajapinnan SAX-jäsentimeen tapahtumankäsittelymetodeineen. Puun elementit ovat Element-olioita ja elementtien attribuutit Attribute-olioita.

Kun sovelma vastaanottaa DTD:n, pitää luoda uusi TreeManager-olio. Kutsutaan createDTD-metodia, jolle annetaan rakennemäärittely parametrina merkkijonomuodossa. Merkkijono käsitellään *Mark Wutkan* DTD-jäsentimellä[1], joka luo *DTD*-olion. Tämä puolestaan sisältää rakennemäärittelyssä esitellyt elementit hajautustaulussa. Elementtien ja attribuuttien määrittelyt jäsentyvät eri luokkien olioiksi. Jäsennin ja sen luoma tietorakenne on kuvattu osoitteessa
`http://www.wutka.com/dtdparserapi/`

Kun DTD on jäsennetty, TreeManager käy läpi elementtien lapsirajoitteet ja tekee säännöllisistä lausekkeista epädeterministisiä äärellisiä automaatteja muistuttavia sääntöverkkoja. Sääntöverkot rakentuvat ANode-olioista. Sääntöverkot sijoitetaan hajautustauluun *childConstraints* elementtien nimillä, joilla validointiin käytettävät solmujen metodit pääsevät niihin käsiksi.

Sääntöverkkojen avulla tarkistetaan, johtaako elementin lapsisarjaan kohdistuva muokkaustoiminto[2] DTD:n sallimaan rakenteeseen. Elementin lapsisarjaan kohdistuva muutos sallitaan vain, jos sen säännöllinen lauseke tai siitä luotu sääntöverkko tuottaa muutoksen tuloksena syntyvän lapsisarjan.

Attribuutteihin liittyvät rajoitteet selviävät DTD-jäsentimen elementtikohtaisesti luomista DTDAttribute-olioista.

Kun tietokannasta ladataan artikkeli, käytetään elementtipuun rakentamiseen *Apache Xerces Java* -kirjaston[3] SAX-jäsennintä. XML-kirjaston dokumentaatio on osoitteessa
`http://xml.apache.org/xerces-j/api.html`

Koska DTD:t ovat tietokannassa eikä niillä ole URL-osoitetta, tieto DTD:stä ei kulje itse XML-osuuden mukana. SAX-jäsennin tarkistaa vain vastaanotettavan dokumentin hyvämuotoisuuden. Jäsennintä käytetään vain rakennettaessa ladatusta dokumentista elementtipuuta.

[1] http://www.wutka.com/dtdparser.html

[2] Lapsisarjaan kohdistuvia muutoksia ovat lasten lisäys, poisto, järjestyksen muuttaminen ja korvaaminen.

[3] http://xml.apache.org/xerces-j/

Dokumentin jäsennys SAX-rajapinnalla on tapahtumaohjattua. Jäsennin lukee dokumenttia suoraviivaisesti alusta loppua kohti ja jokaisen elementin alkaessa ja loppuessa lähettää SAX-tapahtuman. XML-elementti validoidaan vasta jäsentimen kohdatessa elementin lopun, jolloin vastaava SAX-tapahtuma siirtää suorituksen *endElement*-metodiin. Vasta tällöin kaikki lapsielementit ovat tiedossa. Lapsisarja validoidaan sääntöverkoilla ja attribuutit DTD-jäsentimen luomien DTDAttribute-olioiden tiedoilla.

TreeManagerin *toString*-metodi palauttaa artikkelin XML-muodossa.

Kun luodaan uusi artikkeli, TreeManager rakentaa hyväksyttävän artikkelipohjan läpikäymällä sääntöverkkojen solmuja. Elementille lisätään aina verkon ensimmäisen polun määrittelemä lapsisarja. Verkot rakennetaan niin, että ensimmäinen polku on pääsääntöisesti lyhin mahdollinen. Jos elementti ei vaadi lapsielementtejä, niitä ei luoda.

4.2.1 Attribuutit

private DTD dtd
DTD-jäsentimen palauttama DTD-olio.

private UIManager UIManager
Viite UIManageriin.

private Element root
Viite elementtipuun juurisolmuun, joka on NITF-elementti.

private Element head
Viite head-elementtiin.

private Element body
Viite body-elementtiin.

private Attribute[] childConstraints
Hajautustaulu, joka sisältää jokaista DTD:ssä määriteltyä elementtiä kohti sen lapsirajoitteet esittävän säännöllisen lausekkeen pohjalta rakennetun sääntöverkon. Hajautustaulun avaimina käytetään elementtien nimiä.

public final Hashtable BLOCKS
Hajautustaulu, jossa luetellaan kaikkien lohkotyyppisten elementtien tunnisteni-met. Lohkoista tehdään Block-olioita.

private Element curParent
Elementti, jonka lapsia tai attribuutteja viimeksi muutettiin.

private Vector oldChildren
curParentin edellinen lapsisarja.

private Attribute[] oldAttributes
*curParent*in edellinen attribuuttisarja.

private Stack stack
Pino, jota käytetään rakennettaessa puuta XML-merkkijonosta.

private int undo
Ilmaisee nykyisen peruutustoiminnon ja saa seuraavia arvoja:
public static final int NONE = 0
public static final int RESTORE_CHILDREN = 1
public static final int RESTORE_ATTRIBUTES = 2
Kun kutsutaan *undo*-metodia, se kutsuu *undo*-muuttujan arvon osoittamaa metodia.

4.2.2 Metodit

public TreeManager(UIManager uim)
Luo tyhjän TreeManager-olion. Parametri *uim* on viite UIManageriin. DTD ja dokumentti rakennetaan UIManagerin pyynnöstä erillisillä metodeilla, jotka voivat lähettää poikkeuksen.

public void createDTD(String dtd) throws IOException
Jäsentää rakennemäärittelyn ja palauttaa DTD-olion. *dtd* on rakennemäärittely merkkijonomuodossa. Jos jäsennys epäonnistuu, lähetetään poikkeus. Jäsennys tehdään luomalla Mark Wutkan DTDParser-pakkauksen *DTDParser*-luokan olio, välittämällä sille DTD merkkijonomuodossa ja kutsumalla olion *parse*-metodia. Tämän jälkeen kutsutaan vielä metodia *makeChildConstraintGrammars*, joka luo elementtien lapsirajoitteista kielioppisääntöverkot.

public void createTree(String doc) throws Exception
Luo elementtipuun merkkijonosta *doc*. Jos puun rakentaminen epäonnistuu, lähetetään poikkeus.

public void createTree()
Luo sisällöttömän, hyväksyttävän puun DTD-olion pohjalta. Luo nitf-juurielementin ja soveltaa siihen *buildValidSubtrees*-metodia.

public void createTree(String dtd, String doc) throws Exception
Yhdistää rakennemäärittelyn ja dokumentin lataamisen. Lähettää poikkeuksen, jos jotain menee pieleen. Kutsuu *createDTD*-metodia. Jos *doc* on null, kutsuu parametritonta *createTree*-metodia, muutoin createTree-metodia parametrilla *doc*.

public String toString()
Palauttaa puun XML-merkkijonomuodossa. Jatkaa vakiorivejä

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nitf SYSTEM "nitf.dtd">
```

juurielementin *root toString*-arvolla.

Viitteet alipuihin niitä käsitteleville olioille:

public Element getRoot()
Palauttaa viitteen puun juurielementtiin.

public Element getHead()
Palauttaa viitteen puun head-elementtiin. Viite välitetään artikkelin metatietojen muokkauskomponentille.

public Element getBody()
Palauttaa viitteen puun body-elementtiin. Viite välitetään artikkelin rungon muokkauskomponentille.

Hallinnolliset tiedot:

public String getDocID()
Palauttaa dokumentin ID:n.

public void setDocID(String id)
Asettaa dokumentin ID-arvoksi merkkijonon *id*.

public String getDocName()
Palauttaa dokumentin nimen.

public String getDocDate()
Palauttaa dokumentin muutospäivämäärän.

public String[] getDocKeywords()
Palauttaa avainsanat.

Seuraavilla SAX-tapahtumankäsittelymetodeilla rakennetaan puuta ladattua dokumenttia luettaessa:

public void startElement(String uri, String name, String qName, Attributes atts)
Kohdattiin elementin alku. *name* on elementin nimi ja *atts* sisältää kaikki elementin attribuutit. Muita parametreja ei käsitellä. Luodaan uusi *name*-niminen elementti, lisätään sille attribuutit *atts* ja kaikki mahdolliset muut attribuutit tyhjinä merkkijonoina. Jos pinossa on päällimmäisenä PCData-olio, poistetaan se pinosta (vertaa *characters*-metodiin). Lisätään elementti pinoon. Jäsennys keskeytetään, jos tuli vieraan niminen elementti tai attribuutti tai attribuutti väärällä arvolla.

public void endElement(String uri, String name, String qName)
Kohdattiin elementin loppu. Parametreja ei käsitellä. Tässä vaiheessa kaikki lapsielementit ovat tiedossa, joten tarkistetaan, onko lapsisarja hyväksyttävä metodilla *hasValidChildren*. Mikäli ei, jäsennys keskeytetään. Ennen jäsennyksen jatkamista elementti poistetaan pinosta.

public void characters(char[] ch, int start, int length)
Luettiin *length* merkkiä PCDATAa taulukoksi *ch*. Luettu osuus alkoi sarakkeesta *start*. *ch* ei välttämättä kata kokonaista PCDATA-osaa, joten tarkistetaan, onko

pinon päällimmäisenä ennestään PCData-olio. Jos on, liitetään *ch* sen perään. Muutoin luodaan uusi PCData-olio ja laitetaan se pinoon.

Sääntöverkkojen rakentaminen:

private void makeChildConstraintGrammars()
Yksityinen metodi, jolla luodaan sääntöverkot sisältävä hajautustaulu *childConstraints*. Läpikäydään kaikki rakannemäärittelyssä esitellyt elementit, jotka ovat DTDElement-olioina vektorissa *dtd.elements*. Kutsutaan *buildGrammar(item, null)*, missä *item* on DTDElementin *content*-kenttä, joka sisältää DTDItem-olion. Katso DTD-jäsentimen dokumentaatiota.

private ANode buildGrammar(DTDItem item, ANode suc)
Rekursiivinen metodi. Luo säännöllisen lausekkeen osasta *item* sääntöverkon, joka päättyy viitteeseen *suc*. Ylimmällä tasolla *suc* on null. Verkko rakennetaan kuvan 13 (ks. luokka ANode) esittämin periaattein.

Peruutustoiminto:

public void undo()
Peruuttaa edellisen muokkaustoimenpiteen kutsumalla *undo*-muuttujaa vastaavaa peruutusmetodia.

private void restoreChildren()
Peruuttaa elementin *curParent* lapsisarjaan kohdistuneen muutoksen. *curParent*in lapsiksi asetetaan taas *oldChildren*. *oldChildren* puolestaan saa *curParent*in nykyiset lapset.

private void restoreAttributes()
Peruuttaa elementin *curParent* attribuuttisarjaan kohdistuneen muutoksen. *curParent*in attribuuteiksi asetetaan taas *oldAttributes*. *oldAttributes* puolestaan saa *curParent*in nykyiset attribuutit.

Puun muokkaaminen ja muokkaustoiminnon hyväksyttävyyden selvittäminen:

public boolean canReplace(Element parent, int startChild, int startChar, int endChild, int endChar, Vector source)
Voiko *replace*-metodin muokkaustoimenpiteen suorittaa. Rakennetaan elementtien nimiä vastaavista merkkijonoista korvaamisen tuloksena syntyvän lapsisarjan prototyyppi ja kysytään, tuottaako elementtiä *parent* vastaava sääntöverkko sellaisen lapsisarjan (sääntöverkon aloittavan ANoden metodilla *acceptPath*).

public boolean replace(Element parent, int startChild, int startChar, int endChild, int endChar, Vector source)
Jos rakennemäärittely sallii muokkaustoimenpiteen, asetetaan elementille *parent* uusi lapsivektori, joka on kuin *parent.children*, jossa väli (*startChild*, *startChar*) ... (*endChild*, *endChar*) on korvattu vektorin *source* noodeilla. *startChild* on

ensimmäisen korvattavan lapsen indeksi. *startChar* (≥ 0) ilmaisee ensimmäisen korvattavan merkin indeksin *startChild*-lapsessa. *endChild* on ensimmäisen ei-kokonaan korvattavan lapsen indeksi. *endChar* (≥ 0) ilmaisee ensimmäisen ei-korvattavan merkin indeksin *endChild*-lapsessa. Mikäli *startChild*-lapsi ei ole PC-Data, *startChar*-arvon tulee olla 0. Mikäli *endChild*-lapsi ei ole PCData, *endChar*-arvon tulee olla 0. Reunoille mahdollisesti syntyvät peräkkäiset PCDatat yhdistetään.

public boolean canDelTree(Element parent, int startChar, int endChild, int endChar)
Voiko *delTree*-metodin muokkaustoimenpiteen suorittaa.

public boolean delTree(Element parent, int startChar, int endChild, int endChar)
Sama kuin *replace(parent, startChild, startChar, endChild, endChar, new Vector(0))*. Poistaa alueen (*startChild*, *startChar*) ... (*endChild*, *endChar*).

public boolean canAdd(Element parent, int child, int char, Vector source)
Voiko *add*-metodin muokkaustoimenpiteen suorittaa.

public boolean add(Element parent, int child, int char, Vector source)
Sama kuin *replace(parent, child, char, child, char, source)*. Lisää kohtaan (*child*, *char*) vektorin *source* elementit.

public boolean canEncapsulate(Element parent, int startChild, int startChar, int endChild, int endChar, String encapsulator)
Voiko *encapsulate*-metodin muokkaustoimenpiteen suorittaa. Toimintaperiaate kuten *canReplace*-metodilla: rakennetaan lapsisarjan prototyyppi ja katsotaan, tuottaako elementin *parent* säännöllinen lauseke sen.

public boolean encapsulate(Element parent, int startChild, int startChar, int endChild, int endChar, String encapsulator)
Lisää uuden *encapsulator*-nimisen elementin alueen (*startChild*, *startChar*) ... (*endChild*, *endChar*) isäelementiksi ja samalla elementin *parent* lapseksi. Luodaan uusi Element ja annetaan sille lapsiksi annetun alueen elementit. Siten kirjoitetaan elementin *parent* lapsivektori uudelleen niin, että annettu alue on korvattu juuri luodulla elementillä.

public boolean canDecapsulate(Element parent, int index)
Voiko *decapsulate*-metodin muokkaustoimenpiteen suorittaa. Toimintaperiaate kuten *canReplace*-metodilla: rakennetaan lapsisarjan prototyyppi ja katsotaan, tuottaako elementin *parent* säännöllinen lauseke sen.

public boolean decapsulate(Element parent, int index)
Poistaa elementin *parent* lapsen *index* muttei sen alipuita, niin että tuhotun elementin paikalle ilmestyvät sen mahdolliset lapset. Kirjoitetaan elementin *parent* lapsivektori uudelleen niin, että indeksissä *index* olleen viitteen paikalle laitetaan sitä vastaavan elementin lapset.

public void setAttributes(Element parent, Attribute[] source)
Korvaa elementin *parent* attribuuttitaulun attribuuttitaululla *source*.

public String[] getAcceptableReplacement(Element parent, int startChild, int startChar, int endChild, int endChar)
Palauttaa annetun alueen korvaajiksi käyvien yksittäisten elementtien nimet merkkijonotaulukkona. Otetaan huomioon koko lapsisarja. Tällä metodilla saadaan selville, mitkä työkalupalkin painikkeet voidaan aktivoida. Rakennetaan lapsisarjasta prototyyppi (merkkijonotaulukko) ja katsotaan, generoiko elementtiä *parent* vastaava sääntöverkko tämän merkkijonotaulukon.

public void buildValidSubtrees(Element parent)
Rakentaa elementille *parent* hyväksyttävät alipuut, jos rakennemäärittely vaatii elementille lapsia. *childConstraints*-hajautustaulusta löytyy sääntöverkko, jonka aloitussolmun (ANode) *getFirstPath*-metodi palauttaa hyväksyttävän lapsisarjan mallin merkkijonotaulukkona. Jokaisesta merkkijonosta syntyy vastaavanniminen elementti tai PCData-olio. Kaikkiin uusiin elementteihin sovelletaan rekursiivisesti buildValidSubtrees-metodia. Usein elementti ei vaadi yhtään lapsielementtiä.

4.3 TreeNode (PH)

Elementtipuu rakennetaan abstraktin **TreeNode**-luokan aliluokkien olioista. Luokalta peritään nimikenttä *name* ja metodi *toString*, joka tulostaa solmun sisällön. Kaikki TreeNodet ovat alipuineen myös kloonattavia metodilla *cloneTree*.

PCData-olio vastaa XML-dokumentin PCDATA-osuutta eli tekstiä. Tekstisolmuilla ei ole lapsisolmuja.

Element-luokan olio vastaa XML-elementtiä. Element-olioilla on vektori lapsielementeille ja hajautustaulu attribuuteille.

Lohkoelementit ovat *Block*-olioita. Block on kuin Element, mutta se muistaa korkeutensa riveinä ja tuntee isäelementtinsä. Dokumentin selaaminen riveittäin perustuu lohkojen korkeustietoihin. Kun luodaan uusi tyhjä lohko, sen korkeudeksi asetetaan 1. Samalla lisätään isäelementin korkeutta yhdellä rivillä. Aina kun korkeutta muutetaan, muutetaan elementin isän korkeutta yhtä paljon. Kun lohko tuhotaan, vähennetään isälohkon korkeutta tuhotun lohkon korkeuden verran. Lohkoja käytetään vain dokumentin body-elementin alipuussa. bodyn isäelementiksi asetetaan null-osoite. Body on linkitetty body-elementtiin.

DTD:ssä ei rivi- ja lohkoelementtejä ole eritelty, vaan TreeManager-luokan hajautustaulussa *BLOCKS* on kaikkien lohkotyyppisten elementtien tunnistenimet. Jos body-alipuun elementti mainitaan taulussa, siitä tehdään Block, muutoin Element. body-alipuun ulkopuolella kaikki elementit ovat tavallisia Element-olioita.

Attribute on elementtien attribuuttien luokka. Attribuutilla on nimi ja arvo.

Kuva 11 esittää puun solmujen luokkahierarkian. Esimerkki puun rakenteesta on kuvassa 12.

4.3.1 Attribuutit

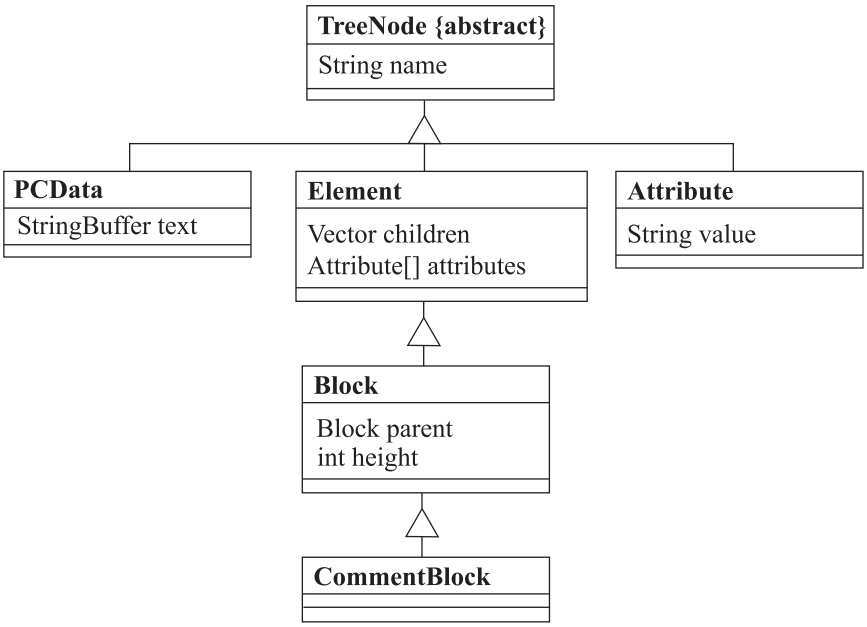
protected String name
Elementin tunnistenimi tai attribuutin nimi. PCData-olion nimi on aina "#PC-DATA". CommentBlock-olion nimi on aina "#COMMENT".

4.3.2 Metodit

public String getName()
Palauttaa solmun nimen.

public String toString()
Palauttaa solmun XML-esityksen.

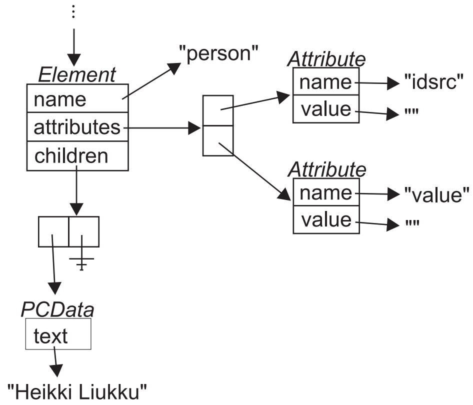
public TreeNode cloneTree()
Palauttaa solmun aloittavan alipuun kopion. Metodia käytetään kopioitaessa alueita leikepöydälle.



Kuva 11: Elementtipuun solmujen luokat.

4.4 PCData (PH)

PCData-olio vastaa PCDATAa XML-dokumentissa.



Kuva 12: Puun rakennetta.

4.4.1 Attribuutit

private StringBuffer text
Teksti.

4.4.2 Metodit

public PCData()
Luo PCData-olion, joka sisältää tyhjän merkkijonon.

public PCData(String s)
Luo PCData-olion, joka sisältää merkkijonon *s*.

public String getText()
Palauttaa tekstin.

public void setText(String text)
Asettaa tekstiksi merkkijonon *text*.

public String getText(int start, int end)
Palauttaa *text*-muuttujan sisällön merkkivälillä *start*...*end*-1.

public void setText(int start, int end, String s)
Korvaa *text*-muuttujan sisällön merkkivälillä *start*...*end*-1 merkkijonolla *s*.

4.5 Element (PH)

Element-luokan olio esittää yhtä XML-elementtiä.

4.5.1 Attribuutit

private Vector children
Elementin lapsielementit Element- ja PCData-olioina.

private Attribute[] attributes
Elementin attribuutit Attribute-olioina.

4.5.2 Metodit

public Element(String name)
Luo *name*-nimisen elementin.

public Vector getChildren()
Palauttaa elementin lapsielementit.

public void setChildren(Vector children)
Asettaa elementin lapsivektoriksi vektorin *children*.

public Attribute[] getAttributes()
Palauttaa elementin attribuutit taulukkona.

public void setAttributes(Attribute[] atts)
Asettaa elementin attribuuteiksi taulukon *atts* attribuutit.

4.6 Block (PH)

Block on lohkotyyppinen XML-elementti ja *Element* on sen yliluokka. Dokumentin runkoa piirrettäessä jokainen Block alkaa uudelta riviltä. Rungon selaaminen perustuu siihen, että jokainen lohko tuntee korkeutensa riveinä. Lohkoja ovat muun muassa kappaleet, listat ja otsikot. Kun uusi lohko luodaan, se saa korkeuden 1. Aina, kun korkeutta muutetaan, lisätään myös lohkon isäelementin korkeutta muutoksen verran.

4.6.1 Attribuutit

private int height
Lohkon korkeus riveinä.

private Object parent
Isäelementti tai vastaava.

4.6.2 Metodit

public Block(String name)
Luo *name*-nimisen Blockin.

public Object getParent()
Palauttaa isäelementin tai vastaavan.

public void setParent(Object parent)
Asettaa isäelementiksi tai vastaavaksi olion *parent*.

public int getHeight()
Palauttaa lohkon korkeuden riveinä.

public void setHeight(int height)
Asettaa lohkon korkeudeksi *height* riviä. Jos *parent* ei ole null, vaan Block- tai Body-olio, lisää samalla sen korkeutta muutoksen verran kutsumalla sen metodia *changeHeight*.

public void changeHeight(int incr)
Lisää korkeutta *incr* riviä.

4.7 CommentBlock (PH)

CommentBlock on XML-kommenttilohko. Käyttäjän kannalta se on lohko muistiinpanoja varten. CommentBlock kuormittaa *Block*-luokan metodeja.

4.8 Attribute (PH)

Tämän luokan olio esittää XML-elementin yhtä attribuuttia.

4.8.1 Attribuutit

private String name
Attribuutin nimi.

private String value
Attribuutin arvo.

4.8.2 Metodit

public Attribute(String name, String value)
Luo *name*-attribuutin arvolla *value*.

public String getValue()
Palauttaa attribuutin arvon.

public void setValue(String value)
Asettaa attribuutin arvoksi *value*.

4.9 XMLAssistant (PH)

Kirjastoluokka, joka sisältää muutaman yleiskäyttöisen apumetodin XML-dokumenttien käsittelyyn.

4.9.1 Metodit

public static String toStringWithEntities(String s)
Rakentaa merkkijonosta *s* merkkijonon, jossa XML:n erotinmerkit on muutettu XML-entiteeteiksi.

public static String toStringWithoutEntities(String s)
Rakentaa merkkijonosta *s* merkkijonon, jossa XML-entiteetit on muutettu erotinmerkeiksi.

4.10 ANode (PH)

ANode on sääntöverkon solmu. Solmulla on arvo ja taulukko seuraajasolmuja. Seuraajasolmujen arvot kertovat siirtymät. Verkon polut päättyvät null-viitteeseen.

Kunkin solmun seuraajataulukkoon asetetaan seuraajat seuraavaan järjestykseen: ensiksi mahdollinen ohitussiirtymä, seuraavaksi kaikki ei-tyhjät siirtymät ja viimeiseksi mahdollinen silmukkasiirtymä. Kuvassa 13 on esitetty verkon rakennussäännöt.

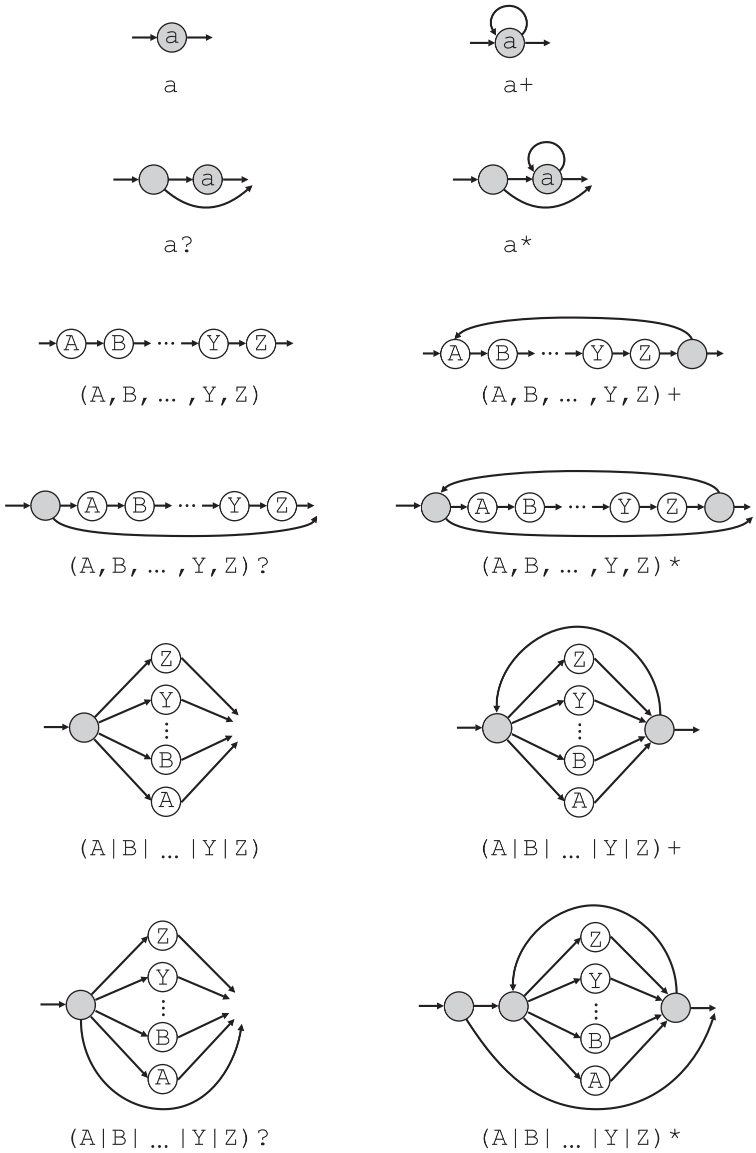
Käytännössä solmut sisältävät elementtien nimiä ja verkkoja käytetään elementin lapsisarjan hyväksyttävyyden tarkistamiseen. Verkon aloitussolmun *acceptPath*-metodia kutsutaan parametreilla *path* ja *index*. *path*in pitää sisältää lapsisarjan elementtien nimet String-taulukossa ja *index*-muuttujan arvon pitää olla 0. *acceptPath* on rekursiivinen metodi ja *index* osoittaa kohtaa, johon asti polku on hyväksytty. Silmukoiden välttämiseksi matkan varrella merkitään tyhjän merkkijonon solmut (joiden arvo on *LAMBDA*) kulloisellakin indeksillä. Jos saavutaan lambdasolmuun samalla indeksillä kuin millä se on merkattu, tiedetään, että jatkamalla jouduttaisiin silmukkaan. Näissä tilanteissa kurkistus hylätään.

Kun käyttäjä yrittää lisätä uuden lapsielementtejä vaativan elementin, joudutaan luomaan validi alipuu soveltamalla aina solmun ensimmäisen seuraajan sääntöä. Esimerkiksi lausekkeella (a | b) luodaan lapsi a, lausekkeella (a | b)* ei luoda yhtään lasta, ja lausekkeella (((a | b), c) | d)+ luodaan lapset a ja c. Myöhemmin käyttäjä voi tietysti korvata elementtejä joillakin toisilla. Ensimmäisen säännön periaate tulee ottaa huomioon rakennemäärittelyä kirjoitettaessa.

ANode-luokan *getFirstPath*-metodi palauttaa verkkoa vastaavalle elementille hyväksyttävän lapsisarjan mallin String-taulukkona. Itse elementtien luonti tehdään *TreeManager*-luokan *buildValidSubtrees*-metodilla.

4.10.1 Attribuutit

private String value
Solmun tuottama merkkijono.



Kuva 13: Sääntöverkon rakentaminen säännöllisestä lausekkeesta.

private int index
Apukenttä, jolla voidaan merkitä kuljettu polku silmukoiden välttämiseksi.

private ANode[] successors
Solmun seuraajat.

public static final String LAMBDA
Tyhjä merkkijono "".

4.10.2 Metodit

public ANode(String value, ANode[] successors)

Luodaan uusi solmu, jolle annetaan arvoksi *value* ja seuraajiksi *successors*.

public boolean acceptPath(String[] path, int index)
Kertoo, tuottaako jokin solmusta alkava polku merkkijonot *path[index]...path[N]*, missä N on taulukon suurin indeksi.

public String[] getFirstPath()
Palauttaa mallin verkkoa vastaavan elementin hyväksymästä lapsisarjasta String-taulukkona. Metodia käyttää *TreeManager*-luokan *buildValidSubtrees*-metodi.

4.11 BodyPanel (PH)

Tämän Panelin aliluokan olion avulla selataan dokumentin runkoa. Se sisältää vierityspalkin, kaksi painiketta ja varsinaisen *Body*-muokkauskomponentin, joka kuuntelee painikkeita ja vierityspalkin säätötapahtumia. Vierityspalkilla dokumenttia selataan riveittäin ja nuolen kuvin varustetuilla painikkeilla sivuttaissuunnassa puolen sivun verran kerrallaan. Sivuttaissuuntaista selausta tarvitaan pitkien esimuotoiltujen tekstirivien katseluun.

4.11.1 Attribuutit

private Scrollbar lineBar
Vierityspalkki, jolla dokumenttia selataan riveittäin.

private Button left
Painike vasemmalle selaamiseen.

private Button right
Painike oikealle selaamiseen.

private Body body
Varsinainen artikkelin rungon muokkauskomponentti.

4.11.2 Metodit

public BodyPanel()
Luo BodyPanel-olion.

4.12 Body (PH)

Body-komponentin kautta käyttäjä muokkaa artikkelin runkoa.

Body pitää muistissa kohdistimen kulloistakin paikkaa. Sillä on muistissa paitsi rivi ja sarake myös isäelementti, lapsen indeksi ja positio lapsen sisällä. Kohdistimen siirto uuteen paikkaan `shift` pohjassa tarkoittaa, että yritetään valita vanhan ja uuden paikan välinen alue. Valinta tehdään, mikäli vanhan ja uuden paikan isäelementit ovat samat.

Valittuun alueeseen kohdistetaan tavallisesti muokkaustoimenpiteitä. Paitsi että alueen sisältöä voidaan muuttaa, sisältö voidaan kopioida leikepöydälle copy- tai cut-toiminnolla. Tieto alueesta palautetaan *AreaSelection*-oliona. Itse alueen sisältö kopioituu vasta, kun suoritetaan muokkaustoiminto. Leikepöytätoiminnoista huolehtii *ClipboardManager*-olio.

4.12.1 Attribuutit

public static final Font TEXT_FONT
Käytetty kirjasintyyppi.

public static final int CHAR_WIDTH
Merkin leveys pikseleinä.

public static final Color BG_COLOR
Taustaväri.

public static final Color TEXT_COLOR
Tekstin väri.

public static final int LINE_HEIGHT
Rivikorkeus.

public static final int TOP_MARGIN
Ylämarginaali.

public static final int BOTTOM_MARGIN
Alamarginaali.

public static final int LEFT_MARGIN
Vasen marginaali.

public static final int RIGHT_MARGIN
Oikea marginaali.

public static final int MAX_COLS
Kuinka monta saraketta voidaan näyttää kerralla.

public static final int MIN_COLS
Kuinka monta saraketta rivin tulisi olla vähintään pitkä, ennen kuin se voidaan katkaista.

public static final int MAX_LINES
Kuinka monta riviä voidaan näyttää kerralla.

private int curLine, curCol
Kohdistimen rivi ja sarake.

private int topLine
Yläreunassa näkyvä rivi.

private int leftCol
Vasemmassa reunassa näkyvä sarake.

private Element curParent
Elementti, jonka sisällä valittu alue sijaitsee.

private int curStartChild, curStartChar
Kohdistimen nykyinen paikka elementin *curParent* sisällä.

private AreaSelection selectedArea
Valittu alue. Jos ei ole valittu aluetta, muuttuja on null.

private BodyPanel bodyPanel
Viite BodyPaneliin, joka sisältää tämän komponentin.

private Scrollbar lineBar
Viite BodyPanelin vierityspalkkiin.

private Button left, right
Viitteet BodyPanelin selauspainikkeisiin.

private Block body
Viite elementtipuun body-elementtiin.

private Image hiddenImage
Kuva piirtopuskurista.

private Graphics hiddenGraphics
Piirtopuskuri, johon teksti piirretään ennen näyttämistä.

private UIManager uim
Viite UIManager-olioon.

private TreeManager tree
Viite TreeManager-olioon.

private ElementPopup elementPopup
Elementtikohtainen valikkoikkuna.

private AttributesDialog attributesDialog
Dialogi, jossa yhden elementin attribuutteja muokataan.

private boolean isShiftDown
Onko `shift`-näppäin pohjassa.

4.12.2 Metodit

public Body(Scrollbar lineBar, Button left, Button right, UIManager uim)
Luo Body-olion. Parametreina välitetään viitteet BodyPanelin Scrollbariin ja selauspainikkeisiin, joiden säätötapahtumia kuunnellaan ja UIManageriin. UIManagerilta saadaan vielä viite TreeManageriin.

public void paint(Graphics vg)
Piirtää ja näyttää näkymän artikkelin runkoon. *vg* on viite näkyvään grafiikkaan. Artikkelin sisältö ladotaan erityisellä rivitysalgoritmilla. Jokainen Block alkaa ja loppuu omalla rivillään, muut osat pyritään sijoittamaan niin, että rivit täyttyvät. Piirrettäessä käytetään tasalevyistä fonttia ja tasakorkuista rivitystä. Kun käyttäjä painaa hiiren painiketta pisteessä (x, y), voidaan laskea, mihin riviin ja sarakkeeseen kohdistin pyritään viemään. Elementtien tunnisteet piirretään samoista kuvista kuin Toolbarin painikkeisto, tunnisteet kehystetään viivoin.

Rivitysalgoritmi pääpiirteissään:

```
jos seuraavaksi tulostettava kokonaisuus mahtuu nykyiselle riville
  tulosta kokonaisuus ja siirry tulosteen pituuden verran oikealle
muutoin
  jos kyseessä on tunniste (jota ei voida katkaista)
    siirry seuraavalle riville ja aloita alusta
  muutoin (kyseessä on PCDATA)
    etsi katkeamiskohdasta alkua kohti välilyöntiä tai väliviivaa
    jos tultiin alkuun saakka
      jos kohta > MIN_COLS
        siirry seuraavalle riville ja aloita alusta
      muutoin
        katkaise merkkijono sivun oikeassa laidassa,
        tulosta, siirry seuraavalle riville ja aloita alusta
    muutoin
      katkaise merkkijono välilyönnin tai väliviivan jälkeen,
      tulosta, siirry seuraavalle riville ja aloita alusta
```

Jos *shiftDown* on true, pyritään valitsemaan alue (*oldLine*, *oldCol*) ... (osoitettu kohta). Alue voidaan valita, jos sen aloittavalla ja päättävällä kohdalla on sama isäelementti. PCDatan sisällä tämä tulkitaan PCDatan isäelementiksi. Jos *curLine* ja *curCol* osoittavat elementin tunnisteen päälle, avataan *elementPopup* (ks. seuraava luku).

public void drawTag(Graphics g, Element e, boolean isStart)
Piirtää elementin tunnisteen grafiikkaan *g*. Piirretään elementin *e* nimeä *e.getName* vastaava Image (saadaan UIManagerilta) ja ympäröidään se viivalla. Jos elementin jokin muokattavissa oleva attribuutti sisältää epätyhjän merkkijonon ja kyseessä on aloittava tunniste, piirretään tunnisteen oikeaan reunaan tähti.

public void keyPressed(KeyEvent e)
Suoritetaan, kun jokin erikoisnäppäin painetaan alas näppäintapahtumana *e*. Jos näppäin oli `shift`, asetetaan muuttujan *shiftDown* arvoksi true.

public void keyReleased(KeyEvent e)
Suoritetaan, kun jokin erikoisnäppäin vapautetaan näppäintapahtumana *e*. Jos näppäin oli `shift`, asetetaan muuttujan *shiftDown* arvoksi false.

public void keyTyped(KeyEvent e)
Tapahtumankäsittelymetodi, joka suoritetaan, kun käyttäjä painaa tavallista kirjoitusnäppäintä. *e* on vastaava näppäintapahtuma. Jos kirjoitettiin merkki, lisätään se PCData-olioon, siirretään kohdistinta yksi sarake oikealle ja tehdään *repaint()*.

public void mouseReleased(MouseEvent e)
Pyritään siirtämään kohdistin hiiren koordinaatteja vastaavalle riville ja sarakkeelle. Alustavasti asetetaan *curCol = (x - LEFT_MARGIN) / CHAR_ WIDTH* ja *curLine = (y - TOP_MARGIN) / LINE_HEIGHT*. Kutsutaan *repaint()*, joka etsii osoitetun kohdan. Jos `shift` ei ole alhaalla (*shiftDown* == false), asetetaan *oldLine = curLine* ja *oldCol = curCol*.

public void actionPerformed(ActionEvent e)
ElementPopupin ja AttributesDialogin kuuntelu.

public AreaSelection getSelectedArea()
Palauttaa valitun alueen. Jos aluetta ei ole valittu, palauttaa null.

4.13 ElementPopup (PH)

JPopupMenun aliluokka. *ElementPopup*-olio näyttää yhden elementin muokkaustoiminnot (kuva 14). Body kuuntelee ElementPopupin tapahtumia. ElementPopup sisältää seuraavat kentät:

Attributes... Body avaa *AttributesDialog*-ikkunan elementin attribuuttien muokkausta varten.

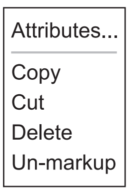
Copy Body kopioi elementin sisältöineen leikepöydälle (*UIManager.copy()*).

Cut Body kopioi elementin sisältöineen leikepöydälle ja samalla poistaa sen artikkelista (*UIManager.cut()*).

Delete Body poistaa elementin sisältöineen (*TreeManager.delTree()*).

Un-markup Body poistaa elementin mutta säilyttää sisällön (*TreeManager.decapsulate()*).

Body sulkee ElementPopupin, kun valitaan jokin muu kohde kuin kyseinen elementti.



Kuva 14: ElementPopup.

4.13.1 Metodit

public ElementPopup(Element parent, int child, int x, int y
Luo elementin *parent* lapsesta *child* JPopupMenun pisteeseen (x, y).

4.14 AttributesDialog (PH)

*Dialog*in aliluokka, jonka ilmentymä sisältää tekstikentät annetun elementin muokkausta varten. Tekstikenttään voi kirjoittaa vain, jos attribuutin arvo ei ole kiinnitetty. Muutokset hyväksytään *Ok*-painikkeella tai hylätään *Cancel*-painikkeella:

Ok Body päivittää elementin attribuutit (*Element.setAttributes()*) ja sulkee ikkunan.

Cancel Body sulkee ikkunan päivittämättä attribuutteja puuhun.

Monivalinta-arvoisia attribuutteja ei ole.

4.14.1 Attribuutit

private Element element
Elementti, jonka attribuutteja muokataan.

private Label elementLabel
Otsikko, jossa muokattavan elementin nimen.

private Label[] attributeLabels
Attribuuttien nimet tekstikenttäkohtaisesti.

private JTextField[] attributeTexts
Tekstikentät, joihin kirjoitetaan attribuuttikohtaisia arvoja.

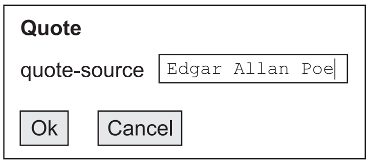
private Button ok, cancel
Painikkeet, joilla muutokset joko hyväksytään tai hylätään.

4.14.2 Metodit

public AttributesDialog(Element e)
Luo JFrame-olion, joka sisältää tekstikentät elementin e attribuuttien muokkausta varten (kuva 15). Muutokset hyväksytään *Ok*-painikkeella tai hylätään *Cancel*-painikkeella. Tekstikentät luodaan elementin e attribuuttitaulusta taulukoksi *J-TextField*-olioita.

4.15 AreaSelection (PH)

Body palauttaa tiedon valitusta alueesta *AreaSelection*-oliona, joka toimii vain tietueena.



Kuva 15: AttributesDialog.

4.15.1 Attribuutit

public Element parent
Elementti, jonka sisällä valittu alue sijaitsee.

public int startChild
Indeksi, joka osoittaa elementin *parent* ensimmäisen ainakin osittain valitun lapsielementin.

public int startChar
Indeksi, joka osoittaa ensimmäisen valitun lapsielementin sisällä ensimmäisen valitun merkin. *startChar* voi olla suurempi kuin 0 vain jos alue alkaa PCData-olion sisällä.

public int endChild
Indeksi, joka osoittaa elementin *parent* sisällä sen lapsielementin, joka ei enää ainakaan kokonaan kuulu alueeseen.

public int endChar
Indeksi, joka osoittaa alueen lopettavan lapsielementin *endChild* sisällä ensimmäisen ei-valitun merkin. *endChar* voi olla suurempi kuin 0 vain jos alue loppuu PCData-olion sisällä.

4.15.2 Metodit

public AreaSelection(Element parent, int startChild, int startChar, int endChild, int endChar)
Konstruktori. Parametrit liitetään samannimisiin kenttiin.

4.16 TreeSelection (PH)

TreeSelection-olio sisältää valitun alueen kopion. Se soveltaa *Transferable*-rajapintaa ja sopii leikepöydän sisällöksi.

4.16.1 Attribuutit

public Vector contents
Kopio valitun alueen sisällöstä *TreeNode*-olioina.

public static final DataFlavor[] SUPPORTED_FLAVORS
Sisältää yhden alkion, *DataFlavor.StringFlavor*, joka on ainoa esitysmuoto, jossa sisältö voidaan palauttaa.

4.16.2 Metodit

public TreeSelection(AreaSelection s)
Konstruktori. Alueen *s* sisältö (alipuut) kloonataan *contents*-muuttujaan. Tässä käytetään hyväksi *TreeNode*-olioiden *cloneTree*-metodia.

public Vector cloneContents()
Palauttaa kopion sisällöstä.

public Object getTransferData(DataFlavor flavor) throws Unsupported-FlavorException
Leikepöydän sisällöltä vaadittu metodi. Palauttaa sisällön esitysmuodossa *flavor*. Jos *flavor* ei ole merkkijonomuoto *DataFlavor.StringFlavor*, annetaan poikkeus. Muutoin palautetaan sisältö merkkijonona, joka saadaan katenoimalla *contents*-vektorin sisältämien *TreeNode*-olioiden *toString*-merkkijonot.

public boolean isDataFlavorSupported(DataFlavor flavor)
Palauttaa true, jos *flavor* == *DataFlavor.StringFlavor*, muutoin false.

public DataFlavor[] getTransferDataFlavors()
Palauttaa *SUPPORTED_FLAVORS*-taulukon.

4.17 ClipboardManager (PH)

Leikepöytätoimintojen hallintaan erikoistunut luokka, joka sisältää leikepöydän ja kaksi *TreeSelection*-olioita: *src* ja *bak*. *TreeSelection* sisältää jonkin elementtipuusta valitun alueen kopion. *src* sisältää copy- tai cut-komennon tuloksena kopioidun syötteen. Leikepöytä on luokan *java.awt.datatransfer.Clipboard* olio, joka sisältää *src*:n. *bak* puolestaan sisältää peruutustoimintoa varten *src*:n vanhan sisällön.

4.17.1 Attribuutit

private Clipboard clipboard
Editorin käyttämä leikepöytä. Käyttöjärjestelmän yleinen leikepöytä, mikäli mahdollista.

private TreeSelection src
Viimeksi leikepöydälle annettu sisältö.

private TreeSelection bak
Edellinen *src* eli edellinen appletin asettama leikepöydän sisältö.

private int undo
Ilmaisee nykyisen peruutustoiminnon ja saa seuraavia arvoja:
public static final int NONE = 0
public static final int UNCUT = 1
public static final int UNPASTE = 2
Kun kutsutaan *undo*-metodia, se kutsuu *undo*-muuttujan arvon osoittamaa metodia.

4.17.2 Metodit

public ClipboardManager()
Konstruktori.

public void undo()
Peruuttaa edellisen muokkaustoimenpiteen kutsumalla *undo*-muuttujan osoittamaa peruutusmetodia.

public void copy(AreaSelection s)
Korvaa leikepöydän sisällön alueen *s* osoittamalla puurakenteella. Tätä ennen ottaa korvattavan sisällön muistiin *bak*-muuttujaan.

public void cut(AreaSelection s)
Korvaa leikepöydän sisällön alueen *s* osoittaman puurakenteen kopiolla. Tätä ennen ottaa korvattavan sisällön muistiin *bak*-muuttujaan. Lopuksi poistaa kopioidun puurakenteen TreeManagerin *delTree*-metodilla.

public void uncut(AreaSelection s)
Peruuttaa cut- tai uncut-toiminnon. Suoritetaan TreeManagerin undo ja vaihdetaan leikepöydän sisältö ja *bak* keskenään.

public void paste(AreaSelection s)
Korvaa alueen *s* osoittaman puun osan leikepöydän sisällön kopiolla. Korvaaminen toteutetaan TreeManagerin *replace*-metodilla.

public void unpaste()
Peruuttaa paste-toiminnon. Suoritetaan TreeManagerin undo.

4.18 Communications

Tämä olio tarjoaa rajapinnan palvelinpuolelle. Communications-olioon on toteutettu saman nimiset functiot kuin palvelimella olevat servletit. Kaikki servleteille lähetettävä sisältö on ensin muutettava url-koodattuun muotoon. Virheen sattuessa Communications-luokan functiot aiheuttavat CommunicationsException-poikkeuksen. Communications-luokka ei varaudu tiedonsiirtovirheisiin. Tiedonsiirrossa turvaudutaan Javan tarjoamiin tiedonsiirtoluokkiin ja niiden mahdolliseen virhetilanteiden torjuntaan.

String loadArticle(integer articleID)

parametrit: Artikkelin id-numero

palauttaa: Tietokannasta ladatun artikkelin yhtenä string oliona tai servletin antaman virheilmoituksen poikkeuksessa.

toiminta: Kutsuu loadArticle nimistä servlettiä, jolle lähettää parametrinä arikkelin id-numeron. Lukee servletin palauttaman artikkelin.

String loadDTD(integer DTDID)

parametrit: DTD:n id-numero

palauttaa: Tietokannasta ladatun DTD:n yhtenä string-oliona tai servletin antaman virheilmoituksen poikkeuksessa.

toiminta: Kutsuu loadDTD nimistä servlettiä, jolle lähettää parametrina DTD:n id-numeron. Lukee servletin palauttaman DTD:n.

String saveArticle(String[] artikkeli)

parametrit: Saa parametrina String-taulukon, joka sisältää 3.5.2 luvussa kuvatun taulun kenttiä vastaavat tiedot. Taulussa kentät ovat seuraavassa järjestyksessä: lid / sname / sdescription / skeywords / sdata / ldtdid / ljournalistid / salias.

palauttaa: Palautetaan ok tai servletin antaman virheilmoituksen poikkeuksessa.

toiminta: Kutsutaan saveArticle-servlettiä taulukossa saaduilla parametreillä. Luetaan servletin palauttama merkkijono ja palautetaan se.

String[][] listArticles()

parametrit: –

palauttaa: String-taulukon joka sisältää kannassa olevien artikkeleiden listauksen tai servletin antaman virheilmoituksen poikkeuksessa.

toiminta: Kutsutaan listArticles-servlettiä ilman parametrejä. Se hakee kannasta artikkelien otsikot ja palauttaa ne yhtenä merkkijonona tälle funktiolle. Merkkijonossa on eri kentät erotettu erotinmerkillä. Merkkijonon sisältö puretaan String-taulukkoon.

String[][] listDTD()

parametrit: —

palauttaa: String-taulukon joka sisältää kannassa olevien DTD:iden otsikot tai servletin antaman virheilmoituksen poikkeuksessa.

toiminta: Kutsutaan listDTD-servlettiä ilman parametreja. Se hakee kannasta DTD:iden otsikot ja palauttaa ne yhtenä merkkijonona tälle funktiolle. Merkkijonossa on eri kentät erotettu erotinmerkillä. Merkkijonon sisältö puretaan String-taulukkoon.

4.19 CommunicationsException

CommunicationsException on Javan Exception-luokan laajennus, jota käytetään Communications-luokan virheiden välittämiseen.

Servletit

Servletit on toteutettu kirjoittamalla jokainen osakokonaisuus omaksi servletiksi. Tällä saavutetaan pienempiä ja paremmin hallittavia moduleita. Koska servletit ottavat vastaan tietoa HTTP-POST -tekniikan kautta, on jokaisen servletin kuvauksessa kerrottu saatavat parametrit.

Kaikki servletit rakentuvat yhdestä *service*-funktiosta. Tämän lisäksi ne kutsuvat Database-olion funktioita tietokannan käyttämiseksi.

4.20 loadArticle

parametrit: Parametrinä saadaan articleid jonka arvona on artikkelin id-numero.

palauttaa: Halutun artikkelin tai tietokannan virheilmoituksen

toiminta: Hakee tietokannasta artikkelin id-numeron perusteella oikean artikkelin ja kirjoittaa sen pyynnön tekijälle.

4.21 loadDTD

parametrit: Parametrina saadaan dtdid jonka arvona on DTD:n id-numero.

palauttaa: Halutun DTD:n tai tietokannan virheilmoituksen

toiminta: Hakee tietokannasta DTD:n id-numeron perusteella oikean DTD:n ja kirjoittaa sen pyynnön tekijälle.

4.22 saveArticle

parametrit: Parametreina saveArticle saa 3.5.2 kappaleessa kuvatun taulun kaikki alkiot samannimisinä kuin ne ovat taulun kuvauksessa. Tähän sisältyy myös artikkeli kokonaisena.

palauttaa: ok tai mahdollisen tietokantavirheen tuloste

toiminta: Tallentaa parametreina saadut tiedot ja artikkeli suoraan tietokantaan. Jos artikkelin id-numeroa ei saada parametrina, generoidaan artikkelille uusi id-numero, joka liitetään artikkeliin. Tällöin artikkelia ei tallenneta vanhan päälle vaan luodaan kantaan uusi rivi artikkelille.

4.23 listArticles

parametrit: —

palauttaa: Servletti palauttaa artikkelin otsikko / id / description / DTD-id kentät. Koska servletti voi palauttaa vain yhden merkkijonon täytyy kentät erottaa toisiltaan erotinmerkillä, jolloin vastaanottaja voi purkaa merkkijonon takaisin kentiksi. Virheen sattuessa servletti palauttaa tietokannan virheilmoituksen.

toiminta: Hakee tiedon artikkelit taulun sisällöstä ja kirjoittaa sen pyynnön tekijälle.

4.24 listDTD

parametrit: —

palauttaa: Servletti palauttaa DTD:n otsikko / id / kuvaus kentät. Koska servletti voi palauttaa vain yhden merkkijonon täytyy kentät erottaa toisillaan erotinmerkillä, jolloin vastaanottaja voi purkaa merkkijonon takaisin kentiksi. Virheen sattuessa servletti palauttaa tietokannan virheilmoituksen.

toiminta: Hakee tiedon DTD taulun sisällöstä ja kirjoittaa sen pyynnön tekijälle.

4.25 Database

Luokka tarjoaa tietokantayhteyden käyttöön rajapinnan.

String openCon(String address, String user, String pass)

parametrit: Saa parametreina tietokannan osoitteen, käyttäjätunnuksen ja salasanan. Tietokannan osoite annetaan muodossa: rajapinnnan nimi:ajurin nimi:koneen nimi:koneen portti:käytetyn tietokannan nimi. Esimerkiksi Oracle tapauksessa: jdbc:oracle:thin:@kontti.helsinki.fi:1522:tktb

palauttaa: Palauttaa joko tyhjän merkkijonon tai tietokannan virheilmoituksen.

toiminta: Alustaa ja avaa parametrien mukaisesti Database-luokalle yhteyden tietokantaan.

String closeCon()

parametrit: –

palauttaa: Palauttaa tyhjän tai virheen sattuessa tietokannan virheilmoituksen.

toiminta: Suljetaan avoinna oleva tietokantayhteys.

ResultSet doQuery(String query)

parametrit: Parametrina saadaan SQL-kysely.

palauttaa: Palautetaan javan ResultSet-olio, joka sisältää tehdyn SQL-kyselyn tuloksen.

toiminta: Suorittaa parametrinä saadun kyselyn avoimena olevan tietokantayhteyden yli tietokantaan ja palauttaa tuloksen. Tietokantavirheen sattuessa päästää siitä syntyvän poikkeuksen läpi kutsujalle.

String doUpdate(String query)

parametrit: Parametrinä saadaan suoritettava SQL-lause.

palauttaa: Palauttaa tyhjän merkkijonon jos operaatio onnistui tai tietokannan virheilmoituksen

toiminta: Suorittaa parametrina saadun SQL-lauseen tietokantaan.

5 Käyttöliittymä(HT ja ET)

Tässä kappaleessa kuvataan JouTo-ohjelmiston käyttöliittymä rakenne ja ulkoasu sekä tekninen toteutus. Jokainen käyttöliittymän osa on kuvattu omassa kappaleessan.

5.1 Käytettävät tekniikat(HT)

Ohjelmiston kehityksessä käytetään Bordland JBuilder 4 -kehitystyökalua ja sen tarjoamia ominaisuuksia. JBuilder 4 käyttää standardeja Java-kirjastoja sekä AWT että Swing-komponentteja joita käytetään käyttöliittymän kehittämisessä. Ohjelmistossa käytetään Javan versiota 1.2.2 ja vastaavia Swing-komponentteja.

5.2 Käyttöliittymän elementit

Tässä luvussa kerrotaan JouTo-ohjelmiston asiakkaan puolen käyttöliittymän eri elementeistä ja niiden teknisestä toteutuksesta.

5.2.1 Valikot

Tässä kappaleessa kuvataan ohjelmistosta löytyvät valikot ja niiden toteutusperiaatteet.

Valikkojen toiminta ja sisältö

Ohjelmiston valikoista käyttäjä pääsee ohjaaman monia ohjelman toimintoja. Ohjelmassa on neljä valikkoa: File, Edit, Options ja About.
File-valikosta löytyvät seuraavat toiminnot

- *New*-toiminnolla käyttäjä voi avata uuden artikkelin.
- *Open*-toiminnolla käyttäjä voi avata aikaisemmin tallennetun artikkelin.
- *Save*-toiminnolla käyttäjä tallettaa editorissa olevan artikkelin.
- *Save As New*-toiminnolla käyttäjä voi tallettaa artikkelin uudella nimellä tallentamatta sitä edellisen artikkelin päälle.
- *Exit*-toiminnolla käyttäjä pääsee pois ohjelmasta.

Edit-valikosta löytyvät seuraavat toiminnot

- *Undo*-toiminnolla käyttäjä voi perua edellisen tekstinmuokkaustoimintonsa.
- *Redo*-toiminnolla käyttäjä voi perua edellisen Undo-operaation.
- *Cut*-toiminnolla käyttäjä voi leikata valitun tekstialueen.
- *Copy*-toiminnolla käyttäjä voi kopioida valitun tekstialueen.
- *Paste*-toiminnolla valittu tekstialue voidaan joko siirtää tai kopioida.

Options-valikosta löytyvät seuraavat toiminnot

- *Article Options*-toiminnolla käyttäjä pääsee muokkaamaan ja tarkastelemaan artikkeliin liittyviä tietoja.

About-valikosta löytyvät seuraavat toiminnot

- *About JouTo*-toiminnolla käyttäjä pystyy tarkastelemaan ohjelman tietoja.

Kuva 16: Valikot

Kuvassa 16 nähdään ohjelman valikot Java Swing-komponenteilla toteutettuna.

Valikkojen tekninen toteutus

Menut sijaitsevat appletin ylälaidassa ja ne on tehty Java Swing-komponenteilla. Varsinaiset menujen luonnin ja niiden kiinnittämiset suoritetaan sen luokan konstruktorissa joka ne luo.Menubar pitää kiinnittää johonkin top-level containeriin, tässä tapauksessa JFrameen, jotta se näkyy myös käyttäjälle.
Esimerkki menujen luomisesta:

```
  import javax.swing.*;

... // luokkajulistukset jne.

  JMenuBar menuBar;        // JMenuBar-komponentti luo menubarin
  JMenu menu,              // JMenu-komponentti luo yksittäisen menun
  JMenuItem menuItem;      // JMenuItem-komponentti luo menun yhden valinnan

  //Luodaan varsinainen menubar
```

```
menuBar = new JMenuBar();
setJMenuBar(menuBar);

//Luodaan varsinainen menu juuri luotuun menubariin
menu = new JMenu("A Menu");
menuBar.add(menu);

//Luodaan menu-elementti juuri luotuun menuun
menuItem = new JMenuItem("A text-only menu item")
menu.add(menuItem);
```

Menuihin voidaan myös helposti liittää kuvia sekä pikanäppäimiä.

Menuihin liittyy myös menu-elementtien tapahtumakuuntelijat, jotka ilmaisevat ohjelmalle käyttäjän valinnat. Tällöin luokan pitää toteuttaa *ActionListener* -tapahtumakuuntelija sekä käsitellä valikoista tulleet tapahtumat.

```
// Luokka MenuDemo toteuttaa ActionListener-kuuntelijan, joka mahdollistaa
// valikoiden tapahtumien kuuntelemisen
public class MenuDemo ... implements ActionListener {

    ...
    // Luokan MenuDemo konstruktori lisää tapahtumakuuntelijat menuihin
    public MenuDemo() {
        // Jokaiseen menu-elementtiin tulee vastaava kiinnitys
        menuItem.addActionListener(this);
        ...
    }

// metodia actionPerfomed (ActionEvent e) kutsutaan kun jokin
// ActionEvent-luokan toteuttanut luokka generoi tapahtuman. Tässä
// metodissa pitää menujen tapahtumat käsitellä
      public void actionPerformed(ActionEvent e) {

        // Otetaan selvää mikä tapahtuma on kyseessä
        // Tehdään tarpeelliset toimenpiteet
```

5.2.2 Tool Bar (ET,HT)

Tool bar (työkalupalkki) sijaitsee appletin ylälaidassa menun alapuolella ja se on toteutettu Java Swing-komponenteilla.

Tool Barin kuvaus

Tool bar on jaettu kahteen riviin, ja siitä löytyy kolmen tyyppisiä painikkita

- Tekstin ja ohjelman toimintoja: undo, redo, cut, copy, paste ja special.
- Block-tason elementtien käsittelyyn liittyvät napit: paragraph, preformatted, bulleted list, numbered list ja blockquote.
- Inline-tason elementtien käsittelyyn liittyvät napit: person, location, organization ja quote.

Näistä painikkeista aktivoidaan aina ne kuvakkeet jotka ovat sillä hetkellä mahdollisia toimintoja ts. käyttäjällä on mahdollista tehdä kuvakkeen ilmentämä toiminto siinä tilateessa. Tool barin kuvakkeiden tilojen muutoksista vastaa UIManager-luokka.

Kuva 17: Tool bar

Kuvassa 17 nähdään kuva työkalupalkin toteutuksesta Java Swing-komponenteilla.

Tool Barin tekninen toteutus

Tool barin ja sen komponenttien luonnin ja kiinnittäminen suoritetaan sen luokan konstruktorissa joka ne luo. JouTo-projektissa tool barin luo UIManager-luokka. Tool bar kiinnitetään Frameen valikon alapuolelle.
Tool Barin luonti:

```
// Käytettävä kirjasto
import javax.swing.*;

...

// Luodaan ilmentymä JToolBar-luokasta
JToolBar toolBar = new JToolBar();
// Lisäätään napit (kts. seuraavaa ohjelmakoodia)
addButtons(toolBar);

// Lisätään luotu Tool bar JPanel-containeriin
```

```
JPanel contentPane = new JPanel();
contentPane.setLayout(new BorderLayout());
contentPane.add(toolBar, BorderLayout.NORTH);
```

Tool bariin voidaan lisätä komponetteja ja komponentteihin voidaan liittää kuvia.

```
  ...

// Tätä metodia kutsuttiin edellisessä esimerkissä lisäämään kuvakkeen
// tool bariin
protected void addButtons(JToolBar toolBar) {

   JButton button = null;

   //Esimerkissä lisätään images-hakemistossa oleva left.gif - kuva
   button = new JButton(new ImageIcon("images/left.gif"));
     ...
   toolBar.add(button);
     ...
```

Tool bariin voidaan lisätä myös eroittimia.

```
toolBar.addSeparator();
```

Tool barin tapahtumakuuntelijat luodaan samalla tavoin kuin valikoiden tapahtumakuuntelijat (katso lukua 5.2.1 saadaksesi lisätietoa). Jokaiseen tool bariin lisättävät komponentin on huolehdittava että lisää itseensä tapahtumakuuntelijan.

Osa kuvakkeista aktivoidaan ja deaktivoidaan riippuen käyttäjän valinnoista ja sen hetkisestä tilasta. Tämä tehdään kutsumalla *setButtonsActive*-metodia. Kutsumisesta ja parametrien oikeellisuudesta huolehtii UIManager-luokka.

public setButtonsActive(string[]buttonlist)

parametrit: Osoite elementtilistan alkuun.

palauttaa: Boolean-arvon onnistumisesta

toiminta: Aktivoi listassa mainitut elementtikuvakkeet. Ilmoittaa UIManagerille, että kuvakkeiden aktivoinnit on suoritettu.

5.3 Metatieto

Luokkakuvauksissa oleva headers jakautuu tässä käyttöliittymät -luvussa kahteen osaan

- metatietoon, jossa on yleistä tietoa artikkelista ja sen kirjoittajasta.
- artikkelin otsikkotietoihin, jotka ovat artikkelista riippuvia otsikkotietoja kuten esimerkiksi artikkelin pääotsikko (katso lukua 5.3.1).

Metatieto-osio sijaitsee tool barin (työkalupalkki) alla. Se on toteutettu Java Swing-komponenteilla.

Metatieto-osion kuvaus

Metatieto-osiossa on yleistä tietoa artikkelista ja artikkelin kirjoittajasta. Metatieto-osiosta löytyvät seuraavat tiedot:

- Kirjoittajan nimi
- Kirjoitusajankohta
- Uutisen tyyppi

Kuva 18: Metatieto-osio

Kuvassa 18 nähdään Java Swing-komponenteilla toteutettu metatieto-osio.

Metatieto-osion tekninen totetutus

Metadata-osio koostuu valmiista Swing-komponenteista eli JPanel-kentästä ja sen sisään laitettavista komponenteista (JTextField,JTextArea, JComboBox). Kentän arvoja muokataan ulkoa käsin suoraan UIManager-luokasta antamalla komponenttien huolehtia itsensä piirtämisestä.
Esimerkkikoodi:

```
// Käytetään Swing-kirjastoa
import javax.swing.*;
 ...

// Luodaan uusi JPanel
JPanel aPanel = new JPanel();
// Asetetaan luotuun JPaneliin haluttu layout
```

```
    aPanel.setLayout(new BorderLayout());

    ...

    // Luodaan halutut komponentit, jotka on tarkoitus laittaa JPaneliin
    JTextField Poksi = new JTextField(20);

    //Elementteihin laitetaan myös käsittelijät tutkimaan niiden muutoksia
    // kuten kohdassa \ref{Menut}
    Poksi.addActionListener(this);


    // Lisätään elementit JPaneliin käyttämällä add-metodia
    aPanel.add(Poksi);
    ...

// Kirjoitetaan elementtiin
aPanel.Poksi.setText="Tämä on tekstiä";
```

5.3.1 Artikkeli

Varsinainen artikkeli sijaitsee tässä osiossa. Artikkeli-osuus koostuu valmiista Java Swing-komponenteista ja JouTo-projektissa luodusta Body-komponentista.

Artikkeli-osion kuvaus

Artikkelista riippuvat otsikkotiedot sekä varsinainen artikkelin sisältö on jaettu erillisiin osiin. Otsikkotiedot näytetään Java Swing-komponenteissa artikkelista riippuvassa järjestyksessä, ja varsinainen artikkelin runko pidetään JouTo-projektissa luodussa Body-komponentissa.
Sekä otsikkotiedot että artikkelin runko ovat tabulaattorilistassa omissa välilehdissään.

Kuva 19: Artikkelin otsikkotiedot

Kuvassa 19 nähdään esimerkki artikkelin otsikkotiedot-osiosta toteutettuna Java Swing-komponenteilla.

Kuva 20: Artikkelin runko - Body-elementti

Kuvassa 20 nähdään varsinaisen artikkelin kirjoitukseen tarkoitetun Body-komponentin toteutus. Body-komponentti on JouTo-ohjelmistossa toteutettu komponentti. Lisätietoja Body-komponentista ja sen toteutuksesta löytyy kappaleesta 4.

Artikkeli-osion tekninen toteutus

Artikkeli-osio Java Swing-komponenteilla luotu artikkelista riippuva komponenttiryhmä, johon sisältyy vielä JouTo-projektissa luotu Body-komponentti. JTabbelPane-komponentti pitää sisällään valitusta artikkelista riippuvien tietojen elementtejä toisessa tabulaattorilehdessään ja artikkelin editointiin tarkoitettua Body-komponenttia toisessa.

JTabbedPane-komponentti voi sisältää seuraavat Swing-komponentit

- JTextField
- JTextArea
- JComboBox
- JScrollPanel

Nämä komponentit asetellaan JTabbedPane-komponenttiin artikkelista riippuvassa järjestyksessä.

JTabbedPanel-komponettiin lisätään artikkelista riippuvat kentät. (JTextField, JTextArea) Näiden sisältöä muokataan suoraan UIManager-luokasta.

```
import javax.swing.*;
...

JTabbelPane jTabbelPane = new JTabbelPane();
JTextBox jTextBox = new JTextBox("Poksi");

// Lisätään uusi tabulaattorilehti, jossa on teksti One ja
// tooltip Does Nothing

jTabbedPane.addTab("One", "Does nothing");
// Indeksi 0 tarkoittaa ensimmäistä
tabbedPane.setSelectedIndex(0);

...

// Lisätään komponentteja JTabbedPaneen
jTabbedPane.add(jTextBox);
```

...

5.3.2 Status bar

Status baria käytetään informaation välittämiseen ohjelmalta käyttäjälle.

Status barin kuvaus

Status bar on ikkunan alalaidassa oleva osa. Status baria käytetään näyttämään informaatiota ohjelmasta ja mahdollisia virheilmoituksia käyttäjälle.

Kuva 21: Status bar

Kuvassa 21 nähdään ohjelmiston Status bar ja siinä oleva esimerkkituloste.

Status barin tekninen toteutus

Status bar sisältyy siihen ikkunaan, jossa ohjelmaa ajetaan eikä sitä tarvitse erikseen luoda Javalla. Kuitenkin ikkunaa luodessa tulee Status bar luoda siihen ikkunaan jossa ohjelmaa ajetaan, jotta sitä voidaan ylipäätään käyttää.
Status baria käsitellään seuraavalla koodilla

```
showStatus("Tämä teksti tulee Status Bariin")
```

Seuraava metodi tarjotaan ulospäin

public boolean writeToStatusBar(String text)

parametrit: Status bariin tulostettava teksti

palaute: Boolean-arvo, joka ilmoittaa onnistumisen tai epäonnistumisen

toiminta: Metodi kirjoittaa Status barin arvoksi annetun tekstin. Metodi tarkistaa annetun arvon järkevyyden ja huolehtii mahdollisien poikkeiksien käsittelystä.

5.4 Käyttöliittymän dialogit (HT,ET)

Ohjelmistossa dialogeja käytetään ottamaan vastaan käyttäjän syötteitä sekä näyttämään käyttäjälle ohjelman vaihtoehtoja sekä tulosteita.

JouTo-ohjelmistossa on seuraavia dialogeja

- New

- Open
- Save
- Save As New
- Textdialog

Kuvaukset dialogeista ja niiden toiminnoista löytyvät seuraavista kappaleista.

5.4.1 Dialogien kuvaukset (ET)

New-dialogi

Kun käyttäjä valitsee menusta *New*:n dialogin metodi kutsuu UIManagerissa olevaa *ListDTDs()* metodia. Metodi palauttaa dialogille stringinä listan DTD:istä. Tässä vaiheessa piirretään dialogin kuva. Kun DTD:n valinta on suoritettu *OK*-painikkeella vahvistaen, dialogin metodi kutsuu vuorostaan UIManagerissa olevaa *new()*-metodia. Tässä vaiheessa dialogin suorittaminen päättyy. Dialogissa on *OK*-ja *Cancel*-painikkeet.

Kuva 22: New-dialogi

Kuvassa 22 nähdään Java Swing-komponenteilla toteutettu dialogi.

Open-dialogi

Kun käyttäjä valitsee menusta *Open*:n, dialogin metodi kutsuu UIManagerissa olevaa *ListArticles()*-metodia. Metodi palauttaa dialogille stringinä listan artikkeleista. Tässä vaiheessa piirretään dialogin kuva.Kun artikkelin valinta on suoritettu *OK*-painikkeella vahvistaen, dialogin metodi kutsuu vuorostaan UIManagerissa olevaa *load()*-metodia. Tässä vaiheessa dialogin suorittaminen päättyy. Dialogissa on *OK*-ja *Cancel*-painikkeet.

Listaus artikkeleista laskevassa järjestyksessä kirjoituspäivämäärän mukaan, joista viimeisin artikkeli on valittuna. Liittymässä näytetään vain kymmenen ensimmäistä artikkelia. Lisää artikkeleita saa vierittämällä palkista. Artikkelista liittymässä näytettävät tiedot ovat: pääotsikko, kuvaus, kirjoittaja ja kirjoituspäivämäärä. Pääotsikon kentän pituus on 25 merkkiä ja kuvauksen 50 merkkiä. Mikäli merkkijonot ovat pitempiä kuin mitä liittymässä on mahdollista näyttää, näytetään käyttäjälle merkkijonon jatkuminen kolmella pisteellä.

Kuva 23: Open-dialogi

Kuvassa 23 nähdään Java Swing-komponenteilla toteutettu dialogi.

Save-dialogi

Kun käyttäjä valitsee menusta *Save*:n, dialogi piirretään Java Swing-komponenteilla. Kun *OK*-painiketta painetaan dialogin metodi kutsuu UIManagerissa olevaa *save()*-metodia. Tässä vaiheessa dialogin suorittaminen päättyy. Dialogissa on *OK*-ja *Cancel*-painikkeet.

Käyttäjän kirjoittama artikkeli tallennetaan sellaisenaan vanhan artikkelin päälle. Tallennuksen eteneminen näytetään käyttöliittymässä tilarivissä (status bar), jossa lukee: tallennetaan. Tällöin käyttäja näkee, että järjestelmä suorittaa pyydettyä toimenpidettä. Tätä tilarivipalautetta näytetään myös *New-*, *Open-* ja *Save As New*-dialogien yhteydessä.

Kuva 24: Save-dialogi

Kuvassa 24 nähdään Save-dialogi Java Swing-komponenteilla toteutettuna.

Save As New-dialogi

Kun käyttäjä valitsee menusta *Save As New*:n, dialogi piirretään Java Swing-komponenteilla. Kun *OK*-painiketta painetaan dialogin metodi kutsuu UIManagerissa olevaa *save()*-metodia. Tässä vaiheessa dialogin suorittaminen päättyy. Käyttöliittymässä on *OK*-ja *Cancel* -painikkeet.

Käyttöliittymässä näytetään artikkelin nimi ja kuvaus. Pääotsikko on artikkelin nimi. Mikäli käyttäjä on kirjoittanut artikkelille kuvauksen, tuodaan se liittymään esille, käyttäjä voi tarvittaessa muuttaa kuvausta. Artikkelin kuvaus ei ole pakollinen. Käyttöliittymässä on *OK*-ja *Cancel* -painikkeet.

Kuva 25: Save As New-dialogi

Kuvassa 25 nähdään Save as New-dialogi Java Swing-komponenteilla toteutettuna.

Tekstidialogi

UIManagerilta tulleen ilmoituksen mukainen tekstidialogi piirretään käyttöliittymässä Java Swing-komponeteilla. Tekstidialogissa näytetään käyttäjälle ohjelman tulosteita ja virheilmoituksia. Dialogi sisältää tilanteesta riippuen joko tekstin ja *OK*-painikkeen, jolla käyttäjä pystyy sulkemaan dialogi-ikkunan tai tekstin ja *OK*-ja *Cancel*-painikkeet.

Kuva 26: Teksti-dialogi

Kuvassa 26 nähdään tekstidialogi esimerkkisyötteellä.

Tällainen virheilmoitus saadaan aikaan kutsumalla tekstidialogia (katso lukua 5.4.2) halutulla tekstillä.

Jos pyydetyn toimenpiteen seurauksena joko tietoja tuhoutuisi tai toimenpidettä ei voida muutoin suorittaa, pyydetään käyttäjältä vahvistus toimenpiteen suorittamatta jättämiselle. Vain vakavissa virhetilanteissa käyttäjältä vaaditaan hyväksyntä suorittamatta jätetylle toimenpiteelle. Tällainen toimenpide voisi olla artikkelin tallennuksen epäonnistuminen. Käyttöliittymässä pysytään toimenpidettä edeltävässä tilassa. Käyttöliittymässä on vain *OK*-painike.

Jos toimenpide voidaan suorittaa tuhoamatta varsinaista tekstiä, tällöin dialogissa on sekä *OK*- että *Cancel* -painike.

Kuva 27: Save-dialogi

Kuvassa 27 nähdään Java Swing-komponentilla toteutettu esimerkki.

5.4.2 Dialogien tekninen toteutus (HT)

Dialogit toteutetaan käyttämällä Java Swing-kirjaston JOptionPane-komponenttia pohjana ja toteuttamalla siihen tarvittavat lisätoiminnallisuudet dialogista riippuen.
Perustoiminnallisuus:

```
// Kirjastot mukaan
import javax.swing.*;

// Todella yksinkertainen dialogi-ikkuna
JOptionPane.showMessageDialog(frame,
                              "Eggs aren't supposed to be green.",
                              "Inane custom dialog",
                              JOptionPane.INFORMATION_MESSAGE;
```

Yo. ohjelmakoodi luo dialogin, jossa lukee *Eggs arent's supposed to be green.*, otsikossa lukee *Inane custom dialog* ja dialogi sisältää vielä informaatiota ilmoittavan huutomerkki-ikonin sinisellä taustalla.

Seuraavissa kappaleissa kuvataan eri dialogeihin liittyvät erikoisominaisuudet ja metodit.

New-dialogi(ET)

Käyttämällä suoraan JDialog:a JOptionPane:n sijasta saadaan dialogeja, jotka tapahtuvat editorin näytöllä. Mikäli dialogiin ei pysty sijoittamaan listaa DTD:stä tai artikkeleista, toteukseen on tällöin käytettävä jotakin muuta tapaa Java Swingin Dialog -komponentteja.

```
// Kirjastot mukaan
import javax.swing.*;

// Monipainikkeinen dialogi-ikkuna
JDialog.showOptionDialog(); //Option:a saadaan
monipainikkeinen dialogi.
```

JDialogi -metodeja, jotka luodaan dialogeja ovat

```
int showMessageDialog(...)  //Yksipainikkeinen dialogi, jossa
hieman informaatiota.
int showOptionDialog(...)  //Järjestelmäkohtainen dialogi
(laajenpi kuin edellinen dialogi.).
int showConfirmDialog(...)  //Kysyy käyttäjältä kysymyksen.
String showInputDialog(...) //Ottaa käyttjälta vastaan
informatioita.
int showInternalMessageDialog(...)   //Implementoi standardidialogia
framena
```

Konstuktorissa voidaan dialogeille. JOptionPane:ssa ja JDialog:ssa luonti ja lisääminen tapahtuu samalla tavoin.

```
//Luodaan uusi dialogi
final JOptionPane optionPane = new JOptionPane(

    //Diaoligin teksti
    "The only way to close this dialog is by\n"
    + "pressing one of the following buttons.\n"
    + "Do you understand?",

    //Dialogin ikoni ja painikkeet
    JOptionPane, QUESTION_MESSAGE,
    JOptionPane.YES_NO_OPTION);

    ...

//Arvon asettaminen
        int value = ((Integer)optionPane.getValue()).intValue();
        if (value == JOptionPane.YES_OPTION) {
            setLabel("Good.");
```

```
} else if (value == JOptionPane.NO_OPTION) {
    setLabel("Try using the window decorations "
            + "to close the non-auto-closing dialog. "
            + "You can't!");
}
```

Dialogin ominaisuuksia ovat

- *Component parentComponent* käytetään vain JDialogi:ssa, jossa ainakin on oltava frame. Ominaisuuden lisäämiseen käytetään metodia *JDialogsetLocationRelativeTo*.
- *Object message* lisätään `"...\n"` -merkeillä.
- *String title* lisätään dialogi-ikkunan otsikko.
- *int optionType* lisätään standardipainikkeina joko `DEFAULT_OPTION`, `YES_NO_OPTION`, `YES_NO_CANCEL_OPTION` tai `OK_CANCEL_OPTION`.
- *int messageType* OptionPaneen lisätään informaatioarvona joko `PLAIN_MESSAGE` (ikoniton), `ERROR_MESSAGE` (viiva punaisessa monikulmion pohjassa), `INFORMATION_MESSAGE` (i-kirjain sinisessä ympyrän pohjassa), `WARNING_MESSAGE` (huutomerkki keltaisessa kolmion pohjassa) tai `QUESTION_MESSAGE` (kysymysmerkki virheässä neliön pohjassa).
- *Odject[] options* lisätään standardialogeissa lisäinformaatiota.
- *Object initialValue* käytetään virheilmoituksissa.

Tekstin tuominen dialogiin

```
String s = customDialog.getValidatedText();
if (s != null) {
 //Tekstiä
 setLabel("Congratulations!  "
  + "You entered \""
       + s
              + "\".");
}
```

Latauksen ja tallennuksen yhteydessä käytettävän dialogin koodiin tulee laittaa

```
// Tiedoston lataus
public static final int LOAD
```

Konstruktorissa luodaan frame

```
// Luo tiedostodialogin frame
public FileDialog(Frame parent, String title) //parent - dialogin
omistaja, title - dialogin otsikko
```

Dialogille luodaan muutostilaa kuunteleva tapahtumakuuntelija.

```
//Dialogin tapahtumakuuntelija luodaan
optionPane.addPropertyChangeListener(
                new PropertyChangeListener() {
                    public void propertyChange(PropertyChangeEvent e) {
                        String prop = e.getPropertyName();
...
```

Open-dialogi

Open-dialogin toteutus tapahtuu samoilla keinoilla kuin New-dialogin toteutus.

Save-dialogi

Save-dialogin toteutus tapahtuu samoilla keinoin, käytetään SAVE:a LOAD:n sijasta ja konstruktorissa luodaan hieman toinenlainen frame.

```
// Tiedoston tallennus
public static final int SAVE

// Luo tiedostodialogin
public FileDialog(Frame parent, String title, int mode)
//parent - dialogin omistaja, title - dialogin otsikko ja mode - SAVE.
```

Save As New-dialogi

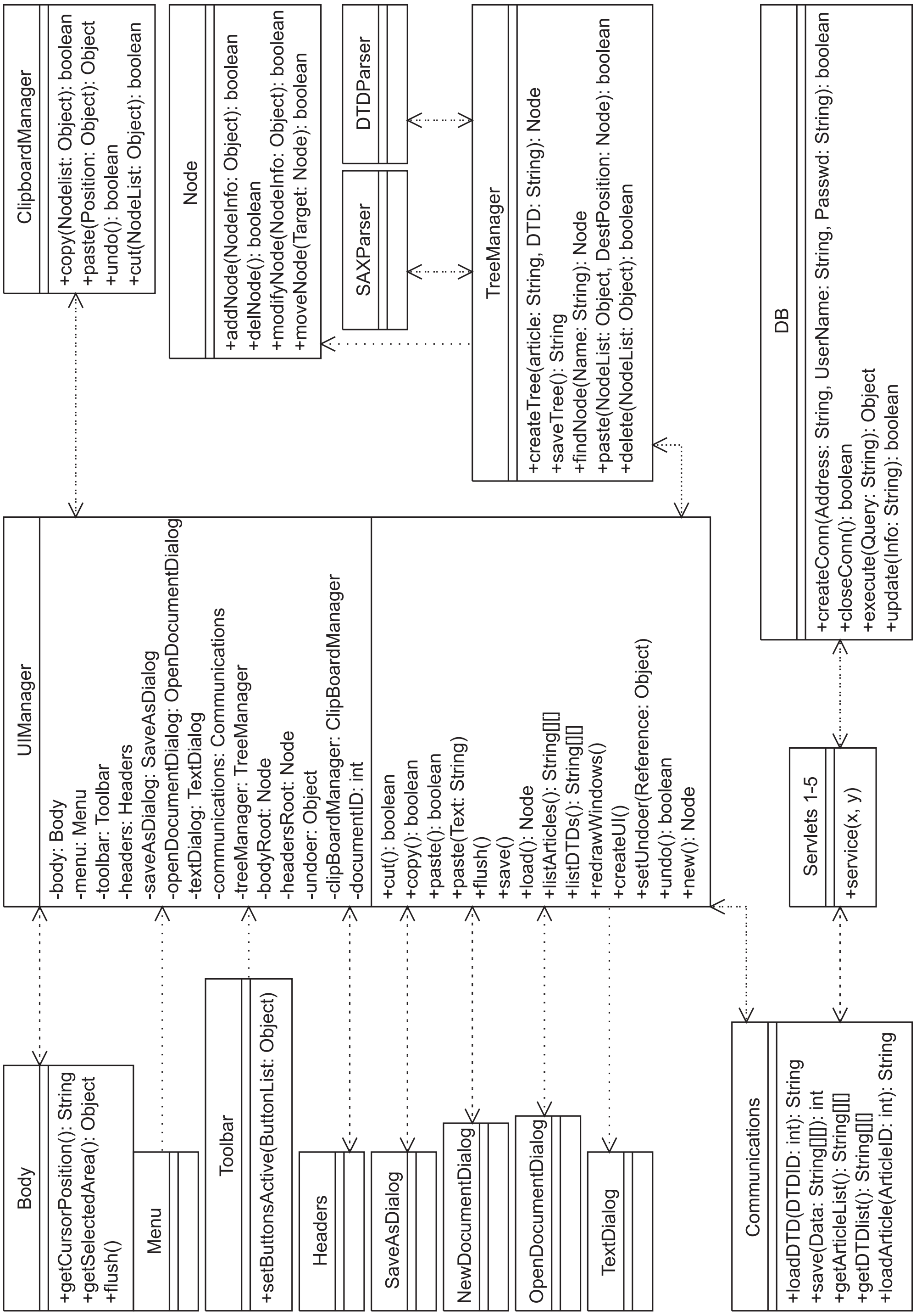
Open-dialogin toteutus tapahtuu samalla tavoin kuin New-dialogin toteutus.

Tekstidialogi

Tekstidialogit toteutetaan saman perusmallin mukaan. Jos virheilmoitus toteutetaan dialogina, virheilmoituksen koodissa annetaan sellainen ominaisuus, jolla estetään dialogin automaattinen sulkeminen. Vain käyttäjän hyväksyntä sulkee dialogin.

```
dialog.setDefaultCloseOperation(
                JDialog.DO_NOTHING_ON_CLOSE);
            dialog.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent we) {
                    setLabel("Thwarted user attempt to close window.");
                }
            });
```

A Liite: Osajärjestelmien rakennekuva



Kuva 28: Osajärjestelmien liittyminen toisiinsa