XML-TIETOJEN VISUAALINEN KÄSITTELY JA ANALYYSI

Mikko Kuru

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Marraskuu 2012

TAMPEREEN YLIOPISTO, Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
KURU, MIKKO: XML-tietojen visuaalinen käsittely ja analyysi
Pro gradu -tutkielma, 75 s.
Marraskuu 2012

Tutkielmassa on käsitelty XML-tietojen visuaalista käsittelyä ja analysointia. XML on suosittu tiedostoformaatti tiedon esittämiseen ja jakamiseen. Tutkielmaa varten kehitetyssä ja toteutetussa visuaalisessa käyttöliittymässä hyödynnetään XML-relaatiota, joka mahdollistaa XML-tietojen käytön relaatiotietokantajärjestelmässä kadottamatta XML-tietolähteistä mitään informaatiota. XML on merkintäkieli puolirakenteisen tiedon kuvaamiseen. Puolirakenteinen tieto on rakenteellista tietoa, joka sisältää oman rakennekuvauksensa. Itsenäiseen käyttöön suunnitellut XML-tietolähteet eivät välttämättä noudata mitään ennalta määriteltyä kaaviokuvausta (esim. DTD tai XML Schema). Tästä seuraa, että eri organisaatioiden XML-tiedot on usein esitetty käyttäen erilaisia rakenteita ja niiden välillä saattaa esiintyä tietokonflikteja. Mikäli kuitenkin esiintyy tarve näiden tietojen yhteiskäytölle, täytyy tietokonflikteja aiheuttavat heterogeenisyystekijät ottaa huomioon. Tutkielmassa esitellään visuaalisia primitiivejä näiden heterogeenisyystekijöiden havaitsemiseen ja poistamiseen ts. tietolähteiden harmonisointiin. Käyttöliittymässä XML-lähde visualisoidaan rakennepuuna. Se muodostetaan käsiteltävän XML-tietolähteen kaaviotason tiedoista. Se noudattaa ns. maksimaalisuusperiaatetta, eli siinä visualisoidaan kaikki tietyllä hierarkiatasolla esiintyvät samannimiset elementtien ja attribuuttien ilmentymät yhtenä tietoalkiona. Rakennepuu toimii myös käyttöliittymänä tietolähteiden ilmentymätasolla sijaitsevien tietojen tutkimiselle. Sen tietoalkioiden kautta voidaan muodostaa tietopilviä ja erilaisia taulumuotoisia esityksiä ilmentymätason tiedoista sekä tuottaa niistä erilaisia tunnuslukuja käyttäen aggregointeja. Tutkielmassa kehitetään myös visuaalinen käyttöliittymä RXQL-kyselykielelle. Tämä käyttöliittymä nostaa RXQL:n deklaratiivisuuden astetta entisestään hyödyntämällä ihmisen visuaalista havainnointikykyä.

Avainsanat: XML, visualisointi, harmonisointi

Esipuhe

Haluan kiittää FT Timo Niemeä tämän tutkielman ohjaamisesta ja tieteellisen ilmaisun opettamisesta. Kiitos myös Katja Moilaselle, Turkka Näppilälle ja Kalervo Järvelinille tutkielman taustalla olevan teoreettisen viitekehyksen luomisesta. Lisäksi haluan kiittää vanhempiani, joita ilman tämä tutkielma ei olisi koskaan valmistunut.

Tampereella 28. Marraskuuta 2012

Mikko Kuru

Sisällysluettelo

1 JOHDANTO

Rakenteellinen tieto on tietoa, jolla on säännöllinen rakenne ja joka määritellään tietomallin avulla. Tietomalli puolestaan koostuu joukosta loogisia tietoyksiköitä ja niiden välisiä suhteita. Relaatiotietokannoissa oleva tieto on rakenteellista tietoa, joka määritellään relaatiomallin [Codd, 1969; 1970] avulla. Codd [1985a; 1985b] määritteli myöhemmin myös, mitä ominaisuuksia tietokannalla pitää olla ja mitä ominaisuuksia sillä saa olla, jotta se olisi aito relaatiotietokanta. Relaatiomallissa ovat seuraavat tietoyksiköt: relaatiotietokanta, joka koostuu joukosta nimettyjä relaatioita, jotka puolestaan koostuvat joukosta nimettyjä attribuutteja. Nimeämällä tietomallin tietoyksiköt ja niiden keskinäiset suhteet ilmaistaan kaaviotaso. Voidakseen hyödyntää relaatiomallia käyttäjän täytyy tuntea relaatioon ja sen attribuutteihin liittyvä semantiikka. Attribuuteille voidaan määritellä tietotyyppejä kuten päivämäärä, teksti, kokonaisluku ja näille tyypeille mahdollisia raja-arvoja. Relaation ns. ilmentymätaso esitetään relaation riveinä, joissa ilmaistaan attribuuttien arvot. Relaatiomallin noudattaessa vähintään ensimmäistä normaalimuotoa [Codd, 1970], rivien ja attribuuttien järjestyksellä ei ole merkitystä. SQL [ISO, 2008] (Structured Query Language) on relaatioalgebraan [Codd, 1970] perustuva kyselykieli, joka on suunniteltu relaatiotietokantojen käsittelyyn.

Rakenteettomalla tiedolla tarkoitetaan tietoa, jolla ei ole olemassa minkäänlaista ennalta määriteltyä kaaviotason kuvausta tai selkeää tietomallia. Tällainen on esimerkiksi lehtiartikkelin sisältävä tekstitiedosto. Kyseisellä lehtiartikkelilla voi olla journalistisessa tai kirjallisessa mielessä rakenne, mutta nämä rakenteet eivät ilmene riittävän säännöllisesti, jotta niitä voitaisiin hyödyntää tietojenkäsittelyssä. Toisin sanoen artikkeliin ei voi kohdistaa täsmällisiä kyselyitä kuten ”palauta kaikki väliotsikot”. Tiedon hahmottaminen ja rakenteen kartoittaminen tämankaltaisesta tiedosta edellyttää tiedonlouhinnan ja tekstianalyysin menetelmien soveltamista, yhdessä tai erikseen. Rakenteeton tieto voi esiintyä myös binäärisessä muodossa. Esimerkki tällaisesta on kuva- tai videotiedosto. Tiedostomuodon formaatti (esim. JPEG tai MPEG) noudattaa tarkasti tiettyä rakennetta, mutta sen sisältö ei tee niin. Tiedostoformaatti on siis pelkkä varasto (rakenteettomalle) tiedolle vastaavalla tavalla kuin sanomalehti on formaatti sen sisältämille artikkeleille.

Puolirakenteinen tieto on rakenteellisen tiedon muoto, jolla ei ole yksiselitteistä ulkoista kaaviota. Puolirakenteisessa tiedossa ei täten ole selkeästi toisistaan erotettavia kaavio- ja ilmentymätasoja, vaan ne esiintyvät siinä lomittain. Tästä syystä puolirakenteinen tieto sisältää oman rakennekuvauksensa, ja sitä kutsutaan itsensä kuvaavaksi tiedoksi. Puolirakenteiselle tiedolle on tyypillistä, että se on epäsäännöllistä, epätäydellistä ja sen rakenne usein vaihtelee ennustamattomalla tavalla. Sen rakenteen ei kuitenkaan tarvitse olla edellä mainitun kaltaista, vaan sen sille voidaan asettaa tiukkoja rajoituksia käyttäen ulkopuolisia kaaviokuvauksia. Tunnettuja formaatteja puolirakenteisen tiedon esittämiseen ovat OEM [Papakonstantinou *et al.*, 1995] ja XML [W3C, 2008].

Internetin ja WWW:n kehityksen myötä tiedon jakaminen on helpottunut ja tietoa löytyy monesta erilaisesta lähteestä. Yleensä tietolähteet on suunniteltu itsenäiseen käyttöön, eikä ulkopuolisten tietotarpeita välttämättä ole otettu huomioon. Tämä synnyttää heterogeenisiä tekijöitä tietolähteiden välillä. Niitä syntyy, kun eri tahot, jotka kontrolloivat tietolähteitä, nimeävät tiedot ja organisoivat tietolähteet autonomisesti. Kyky välittää tietoa fyysisellä, bittien ja tavujen tasolla, ei välttämättä johda kykyyn välittää niihin liittyvää tulkintainformaatiota [Goh, 1997]. Tietolähteet voivat olla keskenään heterogeenisiä usealla eri tavalla: esitysformaatin, semantiikan, syntaksin tai rakenteensa perusteella. Tässä tutkielmassa keskitytään XML-muotoisiin tietolähteisiin, joten voidaan todeta, että esitysformaattiin liittyvää heterogeenisyyttä ei esiinny niiden välillä.

Heterogeenisyys aiheuttaa ongelmia tiedon käsittelyssä, koska tietolähteet eivät ole keskenään yhteensopivia. Niemi *et al.* [2009] määrittelevät semanttisen heterogeenisyyden tilanteena, jossa kahdessa tai useammassa tietolähteessä tarkoitukseltaan samaa tietoa on esitettynä eri tavoin tai tilannetta jossa tarkoitukseltaan erilaista tietoa on esitettynä samalla tavalla. Syntaktisella heterogeenisyydellä tarkoitetaan synonymiaa, jossa sama asia on ilmaistu eri tavoilla tai homonymiaa, jossa eri asiat on ilmaistu samalla tavalla. Rakenteellista heterogeenisyyttä esiintyy, kun samantyyppinen tieto esitetään erilaisten rakenteiden avulla.

Heterogeeniset tietolähteet sisältävät tietojen välisiä konflikteja. Kim ja Seo [1991] määrittelevät konfliktit monitietokantajärjestelmien yhteydessä joko kaavio- tai ilmentymätason konflikteiksi. Heidän mukaansa kaaviotason konfliktit johtuvat joko erilaisista rakenteista samalle tiedolle (relaatiolla ja attribuutilla on sama nimi) tai erilaisista rakenteen määrittelyistä (eri nimiä/tyyppejä/raja-arvoja käytetään semanttisesti samaa

tarkoittaville relaatioille/attribuuteille). Ilmentymätason konfliktit johtuvat joko ristiriitaisesta tiedosta tai erilaisista esitysmuodoista. Epäjohdonmukaista tietoa esiintyy tilanteissa, joissa tietokannan eheyttä ei ole ylläpidetty riittävän huolellisesti. Tällöin useammassa kuin yhdessä tietokannan relaatiossa esiintyvää tietoa ei päivitetä kaikkiin niihin tietokannan relaatioihin, joissa se esiintyy. Esitysmuodosta johtuvassa konfliktissa on kyse synonymiasta tai homonymiasta, yksiköistä tai tarkkuudesta. Niemi *et al.* [2009] laajentavat konfliktien käsittelyn puolirakenteiseen, erityisesti XML-muotoiseen tietoon. He määrittelevät viisi erilaista konfliktityyppiä:

1) Arvojen välinen konflikti
2) Attribuutin ja elementin välinen konflikti, jossa sama tieto esitetään attribuutteina yhdessä ja elementteinä toisessa tietolähteessä
3) Dokumenttien välinen konflikti, jossa sama informaatiosisältö esitetään erilaisten rakenteiden avulla
4) Arvojen ja tietoalkioiden nimien välinen konflikti, jossa sama tieto esitetään jossakin tietolähteessä attribuuttien/elementtien sisältönä ja jossakin toisessa tietolähteessä attribuuttien/elementtien nimenä.
5) Attribuuttien/elementtien keskinäinen konflikti, jossa samaa tarkoittavaa tietoa nimetään eri tavoilla attribuuteissa/elementeissä tai eri asiaa tarkoittava tieto nimetään samoilla attribuuttien/elementtien nimillä.

Kyseiset konfliktityypit esittävät syntaktisia (1), rakenteellisia (2–3) ja semanttisia (4–5) heterogeenisyyden muotoja XML-tietolähteissä.

Mikäli tietolähteiden välillä esiintyy konflikteja, ne eivät ole keskenään yhteensopivia. Tällaisessa tapauksessa tietolähteiden käsittely edellyttää yhteensopimattomuus- eli heterogeenisuustekijöiden huomioonottamista. Niemi ja Järvelin [2006] mainitsevat esimerkkinä tämänkaltaisesta tilanteesta yritysfuusion, jossa osapuolten taloudelliset luvut on esitetty käyttäen erilaisia rakenteita. Kuitenkin esiintyy tarve tarkastella näitä lukuja yhdessä.

Ehdotettuja menettelytapoja heterogeenisuustekijöiden huomioimiseen ovat *tiedon integrointi* (engl. data integration) [Cruz and Xiao, 2001], *tiedon kääntäminen* (engl. data translation) [Abiteboul *et al.*, 1999] ja *tiedon harmonisointi* (engl. data harmonization) [Niemi *et al.*, 2009]. Tiedon integroinnissa luodaan tietolähteiden välinen kaaviokuvaus, joka ilmaisee

miten tietolähteissä olevat tiedot liittyvät toisiinsa. Kaaviokuvauksen muodostaminen saattaa kuitenkin olla työlästä, ja tietotarve voi olla ainoastaan satunnainen. Tietolähteiden rakenne saattaa myös muuttua ajan kuluessa, eikä kaavio siten kuvaa enää alkuperäisiä kohteita tarkasti. Tällöin kaaviokuvausta on päivitettävä. Tiedon kääntämisellä pyritään tietolähteissä olevien tietojen uudelleennimeämisen ja uudelleenorganisoinnin kautta saavuttamaan tietolähteissä olevien tietojen keskinäinen yhteensopivuus. Ongelmana tässä tavassa on, että käännösprosessi perustuu kaaviotason kuvaukseen, joka saa aikaan vastaavat muutokset ilmentymätasolla. Se ei siis sisällä mekanismeja, joilla tietoa voidaan siirtää kaaviotasolta ilmentymätasolle tai päinvastoin. Täten tiedon kääntämisellä ei kyetä käsittelemään kohdassa 4 esiteltyjä konfliktitilanteita. Tiedon harmonisointi on prosessi, jossa heterogeenisisten tietolähteiden väliset *tietokonfliktit* (engl. data conflict) on otettu huomioon. Harmonisoinnin toteuttamiseen voidaan käyttää polkuorientoituneita XML-kieliä kuten XQuery [W3C, 2010a] tai varsinaista kyselykieltä RXQL:ää [Näppilä *et al.*, 2011], joka on toistaiseksi ainoa erityisesti XML-tietojen harmonisoinnin huomioon ottava kieli.

Käsiteltäessä käyttäjälle ennalta tuntemattomia tietolähteitä on välttämätöntä selvittää niiden rakenne ja sisältö, jotta niitä kyetään hyödyntämään. Tämän tutkielman tavoitteena on kehittää graafinen käyttöliittymä visualisointien luomiseen XML-pohjaisista tietolähteistä. Näiden visualisointien avulla käyttäjä kykenee kartoittamaan tietolähteitä ja saa lisää tietämystä sekä ymmärrystä niistä ts. tuntemattomien tekijöiden määrä tietolähteissä asteittain vähenee. Howe *et al.* [2008] kutsuvat tätä *dataspacen* profiloinniksi. Dataspace [Franklin *et al.*, 2005] on tiedonhallintajärjestelmä, jonka tulee sisältää tietyn organisaation kaikki relevantit tiedot niiden muodosta ja paikasta riippumatta. Lisäksi sen pitää luoda että ylläpitää eri tietolähteiden keskinäisiä suhteita. Näppilä ja Niemi [2012] määrittelevät *XML-dataspacen* kokoelmana XML-dokumentteja, jotka on kerätty itsenäisistä ja heterogeenisistä tietolähteistä tiettyä tehtävää tai tarkoitusta varten. Heidän mukaansa XML-dataspacen profiloinnilla on kaksi päätarkoitusta:

1) Käytettävissä olevien XML-tietolähteiden hyödyllisyyden arviointi käyttäjän tietotarpeiden kannalta, pohjautuen sisällön ja rakenteen analysointiin, ja
2) Relevanttien XML-tietolähteiden yhdenmukaisuuden toteaminen avustamalla mahdollisten tietokonfliktien havaitsemista.

Tutkielman eräs osatavoite on kehittää RXQL-kyselykielelle graafinen käyttöliittymä, joka helpottaa harmonisointioperaatioita, selkeyttää tulosdokumenttien tulkintaa ja mahdollistaa RXQL:n muiden ominaisuuksien hyödyntämisen.

2 XML

XML (Extensible Markup Language) [W3C, 2008] on The World Wide Web Consortiumin (W3C) ylläpitämä puolirakenteisen tiedon merkintäkieli, joka on suunniteltu tiedon siirtämiseen ja varastointiin. XML mallinnetaan yleensä nimettynä järjestettynä puuna[1]. Se on kenties maailman yleisin puolirakenteisen tiedon esittämiseen ja siirtämiseen käytetty tietomuoto. XML-dokumentti on hyvin muodostettu ja siten aito XML-dokumentti, mikäli sen sisältö on esitetty laillisilla XML:n syntaksiin perustuvilla ilmauksilla. Jos XML-dokumentti on hyvin muodostettu ja sen rakenne noudattaa annettua kaaviokuvausta, on kyseinen XML-dokumentti oikeellinen kyseisen kaaviokuvauksen suhteen. XML on SGML:lla [ISO, 1986] määritelty kieli.

2.1 XML kielenä

XML on tiedon esittämiseen kehitetty kieli. Itse asiassa XML on kieli, jolla voidaan kuvata toisia kieliä. Toisin sanoen se on metakieli. XML:llä määriteltyä kuvausta voidaan kutsua XML-sovellukseksi. XML toimii metakielenä usealle tunnetulle kielelle. Esimerkiksi SVG (Scalable Vector Graphics) [W3C, 2011a] on kaksiulotteisen vektorigrafiikan kuvauskieli ja XHTML (EXtensible HyperText Markup Language) [W3C, 2002] on tiedon visuaaliseen ilmaisemiseen suunniteltu merkintäkieli. Nämä molemmat ovat siis XML-sovelluksia. XML-dokumentti alkaa aina dokumentin esittelyllä, jossa määritellään tiedon esittämisessä käytettävä versio ja mahdollisesti siinä käytettävä merkistökoodaus. Esimerkiksi ilmaisu

<?xml version="1.0" encoding="ISO-8859-1"?>

määrittelee, että käytössä on XML:n versio 1.0 ja käytettävä merkistökoodaus on ISO-8859-1.

XML:ssä tiedon rakenne kuvataan elementeillä. Nämä elementit voivat sisältää rakenteetonta tietoa, toisia elementtejä tai attribuutteja. Rakenteeton tieto on joko jäsenneltyä (PCDATA) tai jäsentämätöntä (CDATA). Elementtien ja attribuuttien nimet voivat sisältää kirjaimia,

[1] XML-tietolähteeseen liitetty DTD (Document Type Definition) voi aiheuttaa syklisyyttä REFID(S)-attribuuttien myötä, jolloin puun rakenteen ehdot eivät enää täyty.

numeroita ja muita merkkejä. Niiden nimet eivät kuitenkaan voi alkaa numerolla tai välimerkillä, eivätkä ne voi sisältää välilyöntiä. Kirjainten koosta riippumatta kirjainyhdistelmä ’XML’ ei saa esiintyä elementin tai attribuutin nimen alussa. Useat XML-jäsentimet eivät valvo tätä sääntöä, koska kyseiselle rajoitukselle ei ole vielä löydetty käyttöä. W3C on halunnut kuitenkin varata ko. kirjainyhdistelmän tulevaisuudessa tapahtuvia nimiavaruuspäivityksiä varten.

Dokumentin esittelyn jälkeen ilmaistaan juurielementti. XML-dokumentilla on aina yksi juurielementti, jonka nimistä elementin ilmentymää ei saa esiintyä muualla dokumentissa. Juurielementti on siten itse ainoa ilmentymänsä. Jos dokumenttiin ei ole liitetty minkäänlaista kaaviokuvausta, elementtien ja attribuuttien nimeäminen, niiden sisällön määrittely ja niiden esiintymismääreet ovat dokumentin laatijan vapaasti päätettävissä.

Elementit määritellään tunnisteilla (engl. tags), joita on kolmea eri tyyppiä: aloitustunniste, lopetustunniste sekä tyhjä tunniste. Elementin nimeltä *Kirja* alkutunniste on <*Kirja*>, lopputunniste <*/Kirja*> ja tyhjä tunniste <*Kirja /*>. Jokaista alkutunnistetta kohden tulee esiintyä myös lopputunniste. Tyhjä tunniste ei ole välttämättä sisällöltään tyhjä, sillä se voi sisältää attribuutteja. Tyhjien tunnisteiden tehtävänä on toimia paikkana attribuuteille, joita voidaan liittää elementteihin kuvaamaan niiden ominaisuuksia. Tyhjiä tunnisteita voidaan käyttää myös tulostuksen ohjaukseen dokumenttiorientoituneissa XML-sovelluksissa. Tällainen on XHTML-sovelluksissa käytettävä tyhjä elementti <*br /*>, jolla ei ole rivinvaihdon lisäksi muuta tarkoitusta.

Attribuuttien tehtävänä on säilyttää elementteihin liittyvää metatietoa. Metatiedolla tarkoitetaan tässä yhteydessä elementtiä kuvailevaa tietoa, joka ei suoranaisesti liity elementin rakenteeseen vaan kuvaa elementin ominaisuuksia. Attribuutit sijoitetaan elementteihin määrittelemällä niille nimi ja arvo. Attribuutti nimeltä *kustantaja* arvolla ’Otava’ tyhjässä elementissä *Kirja* ilmaistaan

<*Kirja kustantaja=”Otava” /*>.

XML:ssä elementtien ja attribuuttien nimet ovat merkkikoosta riippuvaisia, joten esimerkiksi *kirja* ja *Kirja* ovat eri elementteihin liittyviä nimiä. Attribuuttien keskinäisellä järjestyksellä ei ole merkitystä. Mikäli elementti sisältää useaita attribuutteja, niiden erottimena on välilyönti. Elementti *Kirja*, jolla on attribuutit *kustantaja* ja *kirjoittaja* ilmaistaan

<Kirja kustantaja=”Otava” kirjoittaja=”Hemingway”>.

Elementeillä ja attribuuteilla voi olla XML-dokumentin sisällä useita ilmentymiä. Yksi elementin ilmentymä voi sisältää useita samannimisiä elementtejä välittöminä jälkeläisinään, mutta ainoastaan yhden tietyn nimisen attribuutin. Ilmaisu

<Kirja luku=”1” luku=”2” />

ei ole siten XML:ssä laillinen. Elementtien esiintymät voivat olla rekursiivisia siten, että samanniminen elementin esiintymä voi sisältää toisen samannimisen elementin esiintymän esimerkiksi

<Juuri> <Kirja> <Kirja /> </Kirja> </Juuri>.

Juurielementillä ei voi olla kuitenkaan kuin yksi esiintymä, joten se ei voi olla koskaan rekursiivinen. Lehtisolmut (l. attribuutit ja alirakenteettomat elementit) ovat keskenään samankaltaisia. W3C ei esitä suosituksia siitä, milloin tulisi käyttää attribuutteja ja milloin alirakenteettomia elementtejä.

XML-dokumentti on aina hierarkkinen. Hierarkkisuudella tarkoitetaan sitä, että dokumentin sisällä on määritelty keskinäinen järjestys elementtien ja attribuuttien (mutta ei attribuuttien keskinäisten) esiintymien suhteen. Järjestys määritellään sen tason mukaan, jossa elementti tai attribuutti XML-puussa sijaitsee. Elementin sijaitessa välittömästi toisen elementin alapuolella, on sen yläpuolella oleva elementti alkuperäisen elementin vanhempi. Mikäli kahdella tai useammalla elementillä on yhteinen välitön vanhempi, sijaitsevat ne samalla hierarkiatasolla ja ovat täten keskenään sisarelementtejä. Välillisiä jälkeläisiä ovat kaikki elementin sellaiset jälkeläiset, jotka eivät ole ko. elementin välittömiä jälkeläisiä. Attribuutit eivät voi olla elementtien tai toisten attribuuttien vanhempia. Ne liittyvät aina johonkin elementtiin ja ovat täten ko. elementin välittömiä jälkeläisiä.

2.2 Orientaatiot

XML on monikäyttöinen merkintäkieli. Sitä voidaan käyttää niin tekstidokumenttien kuvauskielenä kuin tietokantatyyppisissä ratkaisuissa tiedon säilyttämiseen, noutamiseen ja vaihtamiseen. XML-tieto voidaan näitä käyttötarpeita varten organisoida *dokumentti-* tai *tieto-*

orientoituneesti. Arvola [2011] huomauttaa, että erottelu näiden kahden orientaation välillä on usein sopimuksenvaraista, sillä XML-tietolähde voi sisältää ominaisuuksia molemmista orientaatioista. Tästä syystä tulisi puhua mielummin dokumentti- tai tieto-orientoituneisuuden asteesta kuin suoraan luokitella XML-tietolähde toisen orientaatiotyypin alle.

Dokumenttiorientoituneet XML-sovellukset on tarkoitettu ensisijaisesti tietojen julkaisemista varten, ja niiden tulkitsijana toimii usein ihminen. Niillä ei välttämättä ole selkeää (tieto)rakennetta. Dokumenttiorientoituneisiin XML-sovelluksiin liittyvät käsittelytarpeet ovat pääosin luonteeltaan tiedonhakuun liittyviä. Niihin kohdistetaan siis suoria avainsanahakuja, joilla pyritään selvittämään miten relevantti kyseinen dokumentti on annettujen avainsanojen suhteen. Koska XML-tietolähde sisältää oman rakennekuvauksensa tunnisteisiin perustuen, voidaan avainsanakyselyt kohdistaa sekä kaavio- että ilmentymätasolle. Dokumenttiorientoituneissa sovelluksissa elementeillä voi olla ulkoasullisia sekä rakenteellisia merkityksiä, ja niiden järjestys on tarkkaan määritelty.

Tieto-orientoituneet XML-sovellukset on tarkoitettu esittämään ja säilyttämään tietoja, joiden tyyppi ja muoto on tarkkaan määritelty. Osaa sen elementeistä käytetään organisoimaan tieto hierarkkisesti, kun taas XML-puun lehdissä esitetään rakenteeton tieto. Tieto on siis systemaattisemmin organisoidumpaa kuin dokumenttiorientoituneissa sovelluksissa. Tieto-orientoituneisiin sovelluksiin liittyvät usein tietokantatyyppiset käsittelytarpeet kuten tietojen ryhmittely, järjestely ja aggregointi. Nämä käsittelytarpeet edellyttävät, että käyttäjä tuntee dokumentin rakenteen ja sisällön. Sisarelementtien, eli tietyn elementin lapsisolmujen keskinäisellä järjestyksellä ei tieto-orientoituneessa XML:ssä ole merkitystä.

2.3 XML:n kaaviokuvaus

XML-dokumenttien sisällön ja rakenteen määrittelemiseen on mahdollista käyttää erilaisia kaaviokuvauksia. Näillä kaaviokuvauksilla luodaan XML-dokumentille erillinen sisältömäärittely, joka ilmaisee minkälaista tietoa dokumentti voi sisältää ja minkälaista tietoa sen pitää sisältää. Tässä luvussa esitellään kaksi tunnettua XML:n kaaviokuvaustapaa: Document Type Definition (DTD) [W3C, 2008] ja XML Schema [W3C, 2004a].

2.3.1 Document Type Definition

DTD määrittelee, miten XML-dokumentin elementit ja attribuutit ja siten myös koko dokumentti tulee muodostaa. DTD määritellään XML-standardin [W3C, 2008] yhteydessä, joten sitä voidaan pitää luonnollisena kaaviokuvaustapana XML:lle. DTD:llä on oma XML:stä poikkeava syntaksinsa. Tämä saattaa aiheuttaa väärinkäsityksiä, koska DTD:n syntaksi muistuttaa osin XML:n vastaavaa. DTD on XML-dokumentin kielioppi ja määrittelee, mitkä ovat kuvauksen mukaisia XML-dokumentteja. DTD määritellään koskemaan joko tiettyä XML-dokumenttia tai niiden kokoelmaa. Huomattavaa on, että DTD:n määrittelystä riippuen sillä on mahdollista määritellä joko vain tiettyä rakennetta noudattavia tai vastaavasti useita eri rakenteita mahdollistavia XML-dokumentteja. DTD voidaan määritellä samassa tiedostossa kuin itse XML-dokumentti, jolloin puhutaan sisäisestä DTD:stä. Jos se määritellään erillisessä tiedostossa, kyseessä on ulkoinen DTD. Myös näiden kahden DTD-tyypin yhdistelmä on mahdollinen.

DTD alkaa nimeämällä dokumenttityyppi (kuva 1; rivi 1) ja sen juurielementti. Dokumenttityyppi voi olla julkinen (PUBLIC) tai yksityinen (SYSTEM). Kuvan 1 esimerkissä kyse on yksityisestä dokumenttityypistä, joka sijaitsee tiedostossa nimeltä Joukkue.dtd. Siinä määritellään jalkapallojoukkueen kokoonpano sarjapeliä varten. Koska DTD määritellään toisessa tiedostossa, se on ulkoinen DTD. Elementit määritellään listauksena, jossa kerrotaan jokaisen elementin nimi ja sisältö (kuva 1; 2–6). Sisältö tulee määritellä yksiselitteisesti käyttäen siihen tarkoitettuja säännöllisiä lauseita, jotka ilmaisevat miten mahdolliset sisällöt esiintyvät elementeissä. Elementin sisällön järjestyksellä on merkitystä, joten sen tulee ilmetä XML-dokumentissa aina vastaavalla tavalla kuin se on DTD:ssä määritelty. Mikäli halutaan määritellä DTD, jossa kuvatun elementin sisällön järjestyksellä ei ole merkitystä, pitää kyseisen elementin mahdolliseksi sisällöksi listata kaikki mahdolliset permutaatiot siten, että ne erotetaan toisistaan DTD:n tai-operaattorilla. Esimerkiksi ilmaisu

<!ELEMENT A ((B, C)|(C, B))>

määrittelee, että elementillä A on sisältönään aina elementit B ja C, eikä niiden keskinäisellä järjestyksellä ole merkitystä. Attribuutit määritellään omana listauksenaan (kuva 1; rivit 7–10), jossa kerrotaan minkä elementin sisälle ne kuuluvat. Tämän lisäksi ilmaistaan attribuuttien nimet, tyypit ja oletusarvot, sekä tieto siitä, onko attribuutin esiintyminen

elementissä pakollinen (#REQUIRED), vapaaehtoinen (#IMPLIED) vai vakio (#FIXED). Attribuutin tyyppiä tai esiintymismäärettä ei voida säädellä XML-dokumentissa, johon ei ole liitettynä DTD:tä tai muuta säätelyn mahdollistavaa kaaviokuvausta. Parametriviittauksella (kuva 1; rivi 11) voidaan määritellä lyhenteitä tekstuaalisille arvoille.

Rivillä 3 määritellään millaisia *Joukkue*-elementin ilmentymät voivat olla. Ne sisältävät yhden tai useamman (+) *Pelaaja*-elementin ilmentymän, aina yhden *Valmentaja*-elementin ilmentymän, mahdollisesti (?) yhden *Divisioona*-elementin ilmentymän ja nolla tai useamman (*) *Huoltaja*-elementin ilmentymän. Muut elementit sisältävät ainoastaan rakenteetonta jäsenneltyä tietoa (PCDATA). Rivillä 7 määritelty *pelinumero*-attribuutin tyyppi on ID, jota voidaan käyttää eri *Pelaaja*-elementtien ilmentymien yksilöintiin. Sen esiintymismäärä on REQUIRED, josta seuraa, että sen pitää löytyä kaikista *Pelaaja*-elementin ilmentymistä. ID-tyyppiset attribuutit vastaavat täten relaatiotietokannoissa käytettäviä pääavaimia, vaikka niiden arvotyyppiä ei voida määritellä vastaavalla tavalla. Rivillä 8 määritellyn *kapteeni*-attribuutin tyyppi on IDREF, ja sen esiintymismäärä on REQUIRED. IDREF-tyyppiset attribuutit ovat luonteeltaan vierasavaimia. Niiden arvojen täytyy viitata aina johonkin ID-tyyppisen attribuutin arvoon. Tätä ID-tyyppistä attribuuttia ei kuitenkaan ole määritelty mitenkään, joten riittää että arvo löytyy minkä tahansa elementin ID-tyyppisestä attribuutista. Rivillä 9 määriteltävä *pelipaikka*-attribuutti on vapaaehtoinen (IMPLIED). Rivillä 10 määritelty *nimi*-attribuutti liittyy *Joukkue*-elementtiin ja sillä on kiinteä (FIXED) arvo ”Toijalan Pallo ’49”. Rivillä 11 määritellään parametriviittaus *osoite* ja sillä on arvona ”ToPa PL 37801 Toijala”.

```
1.  <!DOCTYPE SYSTEM Joukkue  [
2.          <!ELEMENT Joukkue (Pelaaja+, Valmentaja, Divisioona?, Huoltaja*)>
3.          <!ELEMENT Pelaaja (#PCDATA)>
4.          <!ELEMENT Valmentaja (#PCDATA)>
5.          <!ELEMENT Divisioona (#PCDATA)>
6.          <!ELEMENT Huoltaja (#PCDATA)>
7.          <!ATTLIST Pelaaja pelinumero ID #REQUIRED>
8.          <!ATTLIST Joukkue kapteeni IDREF #REQUIRED>
9.          <!ATTLIST Pelaaja pelipaikka CDATA #IMPLIED>
10.         <!ATTLIST Joukkue nimi CDATA #FIXED "Toijalan Pallo ’49">
11.         <!ENTITY osoite "ToPa PL 37801 Toijala">
12. ]>
```

Kuva 1. Joukkue.dtd

2.3.2 XML Schema

XML Schema, joka tunnetaan myös nimellä XML Schema Definition (XSD), on ensimmäinen ulkopuolinen kaaviokuvaustapa XML:lle, joka on saanut W3C:ltä suositus-statuksen. Toisin sanoen se on päässyt osaksi W3C:n suosittelemaa standardia WWW-tekniikkojen hyödyntämiseksi ja siten osaksi W3C:n XML-strategiaa. XML Scheman ja DTD:n suurin ero on XML Scheman kyky määritellä XML:n nimiavaruuksia [W3C, 2009] ja erilaisia tietotyyppejä. Tietotyyppi määritellään joukkona arvoja. Näitä arvoja voidaan yleensä käsitellä tietyillä aritmeeettisilla operaattoreilla. Date [2004] huomauttaa, että XML Scheman tietotyypit eivät kuitenkaan tue muita operaattoreita kuin suuruuden vertailussa käytettäviä pienempi kuin (<), suurempi kuin (>) ja yhtä suuri kuin (=). Tästä syystä ne ovat muistuttavat COBOL-ohjelmointikielessä käytettävien PICTURE-ehtojen mukaisesti määriteltyjä rajoitettuja arvokenttiä, sillä ne määrittelevät ainoastaan tietyt merkkijonoesitykset jokaiselle mahdolliselle tietotyypille. Koska XML-kuvausten tarkoituksena on nimenomaan esittää tietoa, eikä operoida sillä, on kyseinen XML Scheman tietotyyppien määrittely täysin riittävä. Kyky ilmaista tietotyyppejä tekee XML Schemasta luontaisen määrittelyvaihtoehdon etenkin tieto-orientoituneille XML-sovelluksille. Lisäksi XML Schema esitetään XML:llä, eli se on itsessään myös XML-dokumentti. Tämä mahdollistaa XSD-tiedostojen laajentamisen ja uudelleenkäytön XML-tietojen käsittelyyn soveltuvilla kielillä.

Käytettävän XML-version määrittelyn jälkeen pitää kaavion laatijan ilmaista kaaviokuvaukseen käytettävä nimiavaruus tai -avaruudet. Nimiavaruuksien tehtävänä on ryhmitellä toisiinsa liittyvät elementit ja attribuutit yhteen. Tällä voidaan estää johdannossa esitelty attribuuttien/elementtien välinen tyypin 5 tietokonflikti. Nimiavaruuksilla tulee olla yksilöivät tunnisteet.

XML Schema tarjoaa elementeille ja attribuuteille 19 erilaista, suoraan saatavilla olevaa tietotyyppiä, kuten päivämäärä, merkkijono ja desimaaliluku. Sillä on myös mahdollista luoda täysin uusia tietotyyppejä käyttäen hyväksi olemassa olevia tietotyyppejä. XML Scheman valmiissa tietotyypeissä piilee kuitenkin vaara. Esimerkiksi tietotyyppi date (päivämäärä) on muotoa vvvv–kk–pp, joten joulukuun viimeinen päivä vuonna 2012 esitetään sillä muodossa 2012–12–31. Suomalaisen käytännön mukaisesti vastaava päivämäärä esitetään 31–12–2012, kun taas Yhdysvalloissa esitysmuoto on 12–31–2012. Tämä XML Scheman tietotyyppi ei siten vastaa kumpaakaan esitystapaa. Kyseistä tyyppiä käytettäessä tulee siis tiedostaa tämä

poikkeavuus. Uudet tietotyypit voidaan määritellä kahteen eri luokkaan kuuluviksi: yksinkertaisiin (simpleType) tietotyyppeihin ja kompleksihin eli rakenteellisiin tietotyyppeihin (complexType). Kompleksia tietotyyppiä edustavilla elementeillä voi olla oma alirakenteensa, eli niillä voi olla toisia elementtejä ja attribuutteja välittöminä ja välillisinä jälkeläisinään. Sillä voidaan siis esittää useamman tietotyypin hierarkkinen kokoelma. Yksinkertainen tietotyyppi on rakenteeton, toisin sanoen sillä ei ole hierarkkisesti organisoitua alirakennetta. Attribuuteilla ei voi olla jälkeläisiä, joten ne ovat aina yksinkertaisia tietotyyppejä. Käyttäjä voi asettaa molemmille tietotyypeille erilaisia rajoituksia kuten minimi- ja maksimiarvot, pituudet sekä asettaa niille kantatietotyyppejä. Kantatietotyypillä tarkoitetaan tietotyyppiä, kuten esimerkiksi kokonaisluku tai merkkijono, jolle uusi tietotyyppi määrittelee lisää ominaisuuksia rajoitusten avulla. Uusi tietotyyppi voisi siten olla kokonaisluku väliltä 1–99. Tämän lisäksi on mahdollista määritellä säännöllisten lausekkeiden avulla, että tietotyypin arvon tulee esiintyä aina tietyssä muodossa. Tällaisia XML:n elementtien rajoituskokonaisuuksia kutsutaan *faseteiksi*.

Kuvassa 2 annetaan esimerkkinä XML Schemalla tuotettu kaaviokuvaus jalkapallojoukkueelle. Kyseinen kuvaus pyrkii noudattamaan kuvassa 1 esiteltyä DTD:llä ilmaistua kuvausta ja laajentamaan sitä käyttämällä XML Scheman mahdollistamia uusia ominaisuuksia. Rivillä 4 ilmaistaan tietotyypiltään kompleksi *Joukkue*-elementti ja rivillä 6 ilmaistaan, että tämän elementin sisällön järjestyksellä on sen esiintymissä merkitystä, eli sen sisältö esitetään listan muodossa. Mikäli halutaan ilmaista, että elementin sisällön järjestyksellä ei ole merkitystä, tämä voidaan ilmaista sijoittamalla sisältö *xs:sequence*-elementin sijasta *xs:All*-elementin sisään. Sisältönä *Joukkue*-elementillä on neljä alielementtiä: *Pelaaja*, *Valmentaja*, *Divisioona* ja *Huoltaja*. Tämän lisäksi *Joukkue*-elementillä on attribuutti *nimi* vakioarvolla ”Toijalan Pallo ’49”. *Pelaaja*-elementti on tyypiltään PType, joka on määritelty riveillä 15–19 ja se on myös kompleksi tietotyyppi. PTypen pohjatietotyyppi on string (merkkijono), ja se sisältää attribuutit *pelinumero* (rivi 17) ja *pelipaikka* (rivi 18). Näistä attribuuteista *pelinumero* on pakollinen (*use=”required”*) ja *pelipaikka* valinnanvarainen (ei esiintymismäärettä). *Pelaaja*-elementti voi esiintyä 1–14 kertaa, eli niin monta kertaa kuin joukkueen kokoonpanossa voi yhdessä ottelussa olla pelaajia. *Valmentaja*-elementtiin liitettävä tyyppi VType on yksinkertainen merkkijonopohjainen tietotyyppi ja sen ilmentymä esiintyy aina yhtä *Joukkue*-elementin ilmentymää kohdin. *Divisioona*-elementin tyyppi on DType, joka on kantatietotyypiltään

kokonaisluku (Integer). Sen minimiarvo on 1 ja maksimiarvo on 6. *Divisioona*-elementti voi esiintyä joko kerran tai ei ollenkaan *Joukkue*-elementin ilmentymässä. *Huoltaja*-elementtiin liittyvä tyyppi HType on yksinkertainen merkkijonopohjainen tietotyyppi. Se voi esiintyä rajattoman monta kertaa tai olla esiintymättä lainkaan *Joukkue*-elementin ilmentymässä. XML Schemassa ei voi määritellä parametriviittauksia samalla tavalla kuin DTD:ssä. XML Schemalla tehdyt kaaviokuvaukset ovat kuitenkin itsessään XML-dokumentteja, joten niihin voidaan liittää DTD ja tätä kautta mahdollistaa parametriviittauksien käyttäminen.

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Joukkue">
<xs:complexType>
<xs:sequence>
<xs:element name="Pelaaja" type="xs:PType" minOccurs="1" maxOccurs="14" />
<xs:element name="Valmentaja" type="xs:VType" />
<xs:element name="Divisioona" type="xs:DType" minOccurs="0" maxOccurs="1" />
<xs:element name="Huoltaja" type="xs:HType" minOccurs="0" maxOccurs="unbounded" />
</xs:sequence>
<xs:attribute name="nimi" type="xs:string" fixed="Toijalan Pallo '49" />
</xs:complexType>
</xs:element>
<xs:complexType name="PType">
<xs:restriction base="xs:string" />
<xs:attribute name="pelinumero" type="xs:integer" use="required" />
<xs:attribute name="pelipaikka" type="xs:string" />
</xs:complexType>
<xs:simpleType name="VType">
<xs:restriction base="xs:string" />
</xs:simpleType>
<xs:simpleType name="DType">
<xs:restriction base="xs:integer">
<xs:minInclusive value="1" />
<xs:maxInclusive value="6" />
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="HType">
<xs:restriction base="xs:string" />
</xs:simpleType>
</xs:schema>
```

Kuva 2. Joukkue.xsd

3 XML-RELAATIO LÄHTÖKOHTANA VISUALISOINNILLE

XML-relaatio on Niemen ja Järvelinin [2006] kehittämä relationaalinen esitystapa XML-dokumenttien tallentamiseen ja käsittelemiseen relaatiotietokannoissa. XML-relaatioesitystapa poikkeaa relaatiotietokannoissa käytetystä relaatiokäsitteestä monilta osin. Se on kehitetty esittämään puolirakenteista tietoa – ei rakenteellista tietoa kuten perinteisissä relaatiotietokantajärjestelmissä. Relaatiotietokannan rivi voi sisältää ainoastaan ilmentymätason tietoa, kun taas XML-relaation rivi voi sisältää myös kaaviotason tietoa. XML-relaatiolla on aina tasan kolme attribuuttia, kun taas perinteisen mallin mukaisessa relaatiossa asteluku on vapaa. XML-relaation attribuutit on myös määritelty tarkasti, eikä käyttäjä kykene muuttamaan niitä. Tässä luvussa esitetään perusteluita XML-relaation käyttämiselle tutkielmaa varten luodun visuaalisen käyttöliittymän toteutuksessa.

3.1 XML-relaation määritelmä

XML-relaatio on kolmipaikkainen relaatio D(C, T, I), jossa D on XML-dokumentin nimi, C dokumentissa D esiintyvä komponentti, T komponentin C tyyppi (elementti, attribuutti tai arvo) ja I on komponenttiin C liittyvä indeksi. C, T ja I ovat siis XML-relaation D attribuutteja. Nämä attribuutit ovat relaatiotietokannan attribuutteja, eikä niitä pidä sekoittaa XML:n attribuutteihin.

XML-relaatioesityksessä kaavio- ja ilmentymätasot erotetaan toisistaan tyypin perusteella. Tyypin ollessa ’*e*’ komponentti C ilmaisee elementin nimen. Mikäli tyyppi on ’*a*’, ilmaisee C attribuutin nimen ja jos tyyppi on ’*v*’, niin C ilmaisee arvon tai sen osan. Elementtien ja attribuuttien nimet kuuluvat kaaviotasolle, kun taas niiden sisältämät tekstuaaliset arvot kuuluvat ilmentymätasolle. Indeksi I ei yksikäsitteisesti liity kumpaankaan tasoon, vaan toimii tunnisteena komponenteille. I myös erottaa toisistaan samannimisten attribuuttien ja elementtien ilmentymät ja ilmaisee komponenttien väliset hierarkkiset rakenteet [Niemi *et al.*, 2009].

XML-dokumentissa elementin arvo voi olla monitulkintainen. Elementin sisältäessä sekä rakenteetonta tekstimuotoista tietoa että toisia elementtejä nousee ongelmaksi elementin arvon

määritteleminen. Esimerkkitapauksessamme (kuva 3) *Kirjoittajat*-elementillä on tekstisisältönä ”John Smith” sekä yksi *Pääkirjoittaja*-elementti. Vaikka XML-dokumentti sisältää oman rakennemäärittelynsä, ei se silti sisällä omaa tulkintaansa. Näin ollen oikean semantiikan määrittely XML-dokumentissa esiintyville elementeille on vaikeaa. XML-relaatioesitystavassa tekstisisältö evaluoituu elementin arvoksi, vaikka dokumentin perusteella voisi päätellä alielementin arvon (”David Jones”) olleen myös yksi kirjoittajista. XML-relaatio ei siis pyri tulkitsemaan dokumentin semantiikkaa millään tavalla. Huomattavaa on, että esimerkkimme edustaa selkeästi dokumenttiorientoitunutta XML-rakennetta, eli sen käsittelyyn soveltuvat paremmin tiedonhaulliset menetelmät kuin suoraan sen rakenteeseen kohdistetut kyselyt. Dokumentti voidaan siis tunnistaa helposti potentiaalisesti relevantiksi, mutta tietokantatyyppistä hakua, jolla pyritään palauttamaan kaikki kirjoittajat, ei ole mahdollista toteuttaa yksinkertaisesti.

```
1. <Kirja nimi="Book">
2. <Kirjoittajat>
3. John Smith
4. <Pääkirjoittaja>
5. David Jones
6. </Pääkirjoittaja>
7. </Kirjoittajat>
8. </Kirja>
```

Kuva 3. Kirja.xml

Yhteen relaatiotietokannan relaatioon tallennetaan kerrallaan aina yksi XML-dokumentti, joka esitetään XML-relaation avulla. Indeksointipiirteinä siinä ovat elementtien ja attribuuttien nimien esiintymät sekä niiden arvot. Mikäli arvo koostuu merkkijonosta, jokainen siinä oleva yksittäinen *sana* indeksoidaan erikseen. Sanalla tarkoitetaan tässä välilyönnillä erotettua merkkiyhdistelmää, jossa merkkien tyyppiä ei ole rajoitettu mitenkään. Se sisältää siis luonnollisen kielen sanat, aakkosnumeeriset arvot, symbolit sekä kaikki näiden mahdolliset yhdistelmät.

Indeksointi säilyttää XML-dokumentin hierarkkisen rakenteen. Taulussa 1 on esitelty kuvan 4 XML-dokumentti XML-relaatioesityksenä. Juurielementtinä on *Kirjasto* ja siihen liittyy indeksi (I-attribuutti XML-relaatiossa) arvolla <1>. Kirjasto-elementin välittömiä jälkeläisiä ovat kaksi *Kirja*-elementin esiintymää, joihin liittyvät indeksiarvot <1,1> ja <1,2>. XML-

relaatiolla on XML-dokumentin tapaan aina juuri. XML-relaatioesitystavassa se on komponentti, joka on tyyppiä *'e'* eli elementti ja sen indeksiarvo on aina <1>.

XML-relaatioesitystavassa välittömät jälkeläiset tunnistaa helposti indeksiarvosta. Komponentin C välittömiä jälkeläisiä ovat kaikki ne komponentit, joihin liitetty indeksiarvo sisältää ensin C:hen liitetyn indeksiarvon, jota seuraa yksi numero indekseissä. Välillisiä jälkeläisiä ovat komponentit, joihin liittyvät indeksiarvot sisältävät ensin C:hen liitetyn indeksiarvon ja tämän jälkeen kaksi tai useampaa numeroa. Näin ollen komponentin 'Kirja', johon liittyy indeksiarvo <1,1> välittömiä jälkeläisiä ovat komponentit 'kirjoittaja' indeksiarvolla <1,1,1>, 'Sieppari' indeksiarvolla <1,1,2> sekä 'ruispellossa' indeksiarvolla <1,1,3>. Välillisiä jälkeläisiä ovat komponentit 'J.D.' indeksiarvolla <1,1,1,1> sekä 'Salinger' indeksiarvolla <1,1,1,2>. Sisarkomponentit ovat komponentteja, joilla on sama välitön vanhempi, eli niihin liitetty indeksiarvo on viimeistä numeroa lukuun ottamatta sama.

C	T	I
Kirjasto	e	<1>
Kirja	e	<1,1>
kirjoittaja	a	<1,1,1>
J.D.	v	<1,1,1,1>
Salinger	v	<1,1,1,2>
Sieppari	v	<1,1,2>
ruispellossa	v	<1,1,3>
Kirja	e	<1,2>
kirjoittaja	a	<1,2,1>
Ernest	v	<1,2,1,1>
Hemingway	v	<1,2,1,2>
Kenelle	v	<1,2,2>
kellot	v	<1,2,3>
soivat	v	<1,2,4>

Taulu 1. Kirjasto.xml - XML-relaatioesityksenä

```
1. <Kirjasto>
2. <Kirja kirjoittaja="J.D. Salinger">
3. Sieppari ruispellossa
4. </Kirja>
5. <Kirja kirjoittaja="Ernest Hemingway">
6. Kenelle kellot soivat
7. </Kirja>
8. </Kirjasto>
```

Kuva 4. Kirjasto.xml

3.2 Ero muihin XML-dokumenttien esittämis- ja tallentamismenetelmiin

Puolirakenteisen tiedon ja XML-dokumenttien tallentamiseen ja esittämiseen relaatiotietokannoissa on kehitetty monia erilaisia menetelmiä. Perinteisillä relaatiotietokantajärjestelmien relaatiokäsitteillä on käytännössä lähes mahdotonta esittää puolirakenteista tietoa menettämättä jotakin olennaista informaatiota (esim. järjestysinformaatiota) [Pal *et al.*, 2004]. XML-relaatioesitystapa kykenee säilyttämään XML-dokumentin sisäisen järjestyksen ja rakenteen, joten se voidaan aina palauttaa alkuperäiseen tekstuaaliseen muotoonsa.

XML-dokumenttien pilkkominen tietokannan relaatioihin on usein käytetty lähtökohta XML-dokumenttien käsittelemiseen relaatiotietokannoissa [Halverson *et al.*, 2004; Shanmugasundaram *et al.*, 1999]. Tällainen lähestymistapa vaatii kuitenkin ulkopuolisen kaavion, joka määrittelee miten taulut muodostetaan. Tämä on toimiva ratkaisu, mikäli dokumenttien rakenne on säännöllinen eikä niissä ilmene monitulkintaisuutta. XML-dokumenteille on kuitenkin ominaista niiden ilmaisun monimuotoisuus, eikä pilkkominen relaatioihin ole siten kaikkiin tilanteisiin sopiva menettelytapa.

Toinen ehdotettu lähestymistapa on tallentaa XML-dokumentti erillisenä objektina johonkin relaatiotietokannan relaation attribuuttiin, jonka jälkeen kyseistä attribuuttia voidaan käsitellä XQuery-kielellä [W3C, 2010a] (Microsoft SQL Server) tai SQL:n laajennuksella SQL/XML-kiellä [ISO, 2003] (Oracle Database). Tämä lähestymistapa edellyttää kuitenkin uusien proseduraalisten kyselykielien opettelemista. Se vaatii lisäksi kielten ilmausten synkronointia, sillä relaatiomalli perustuu riviorientoituneisuuteen, kun taas XML-dokumentteja käsitellään puumallin pohjalta. Se ei myöskään ole yhteensopiva kaikkien relaatiotietokantojen tai niiden ohjelmointirajapintojen kanssa.

Florescu ja Kossman [1999] ehdottavat menetelmää, jossa XML-dokumentti esitetään graafin muodossa. Elementin ja sen alielementtien väliset suhteet kuvataan numeroiduilla kaarilla, jotka on nimetty alielementtien mukaan. Jokaisella alirakenteen sisältävällä elementillä, eli elementillä jolla on jälkeläisenään attribuutteja tai elementtejä, tulee olla yksikäsitteinen tunniste. Mikäli tällaista tunnistetta ei ole, se lisätään automaattisesti. Tässä menetelmässä alirakenteettomien elementtien ja attribuuttien välillä ei tehdä eroa, joten dokumentin alkuperäinen rakenne rikkoontuu. Kehittäjät kuitenkin huomauttavat, että elementtien ja attribuuttien erottamisen mahdollistava laajennus olisi toteutukseltaan suhteellisen yksinkertainen.

3.3 XML-relaation hyödyt visualisoinnin toteuttamisessa

Relaatiotietokannan käyttämiselle visualisoinnin toteuttamisessa on olemassa useita puoltavia tekijöitä. Tekstuaalisessa muodossa oleva XML-dokumentti saattaa olla rakenteeltaan voimakkaasti vaihteleva, ja sen hyödyntäminen vaatii erillisen jäsentimen käyttämistä. Relaatiotietokannat on suunniteltu suurten tietomassojen käsittelyyn, ja niistä on helppo saada selkeästi tulkittavaa, rivimuotoista, informaatiota käyttäjälle. Tältä osin relaatiomalliin pohjautuvan järjestelmän käyttö on perusteltua.

Relaatiotietokanta mahdollistaa erilaisten aggregointi- ja ryhmittelyfunktioiden käyttämisen SQL-kyselyissä. Ohjelman käyttämä kanta voidaan sijoittaa erilliselle palvelimelle, jolla on laskennalliset valmiudet suoriutua raskaistakin tehtävistä. Tämä mahdollistaa visualisointiohjelman käyttämisen suorituskyvyltään heikompien laitteiden kautta. Relaatiokannoissa on mahdollista määritellä dokumenttikokoelmia ja ylläpitää näitä kokoelmia kuvaavia tauluja (viitteitä yksittäisiin dokumentteihin). Turvallisuustekijät myös puoltavat relaatiotietokannan käyttämistä, koska relaatiotietokantajärjestelmissä on yleensä toteutettuna hyvin korkeatasoinen tietojen suojaus ja varmuuskopiointi. Relaatiotietokannassa on myös mahdollista määritellä, että kokoelmaa kuvaavaan tauluun ei voida lisätä viittauksia sellaisiin dokumentteihin, joita ei löydy kannasta. Lisäksi voidaan määritellä, että jos dokumentti poistetaan kannasta, viittaukset siihen poistetaan automaattisesti myös kokoelmia kuvaavista tauluista. Näin säilytetään tietokannan *viite-eheys* (engl. referential integrity). Relaatiotietokanta mahdollistaa *näkymien* (engl. view) käyttämisen. Näkymä on

relaatiotietokannasta konstruoitavissa oleva virtuaalinen taulu. Se muodostetaan[2] käyttäen SQL:n kyselyoperaatioita. Hyödyllisiä näkymiä voivat olla esimerkiksi taulu, joka sisältää kaikkien XML-relaatiossa esiintyvien elementtien tai attribuuttien nimet tai taulu, joka sisältää kaikki kyseisessä XML-relaatiossa esiintyvät numeromuotoiset arvot. Yleisesti ottaen relaatiotietokantajärjestelmät ovat ohjelmistokehityksellisessä mielessä kypsä tekniikka [Humphrey, 1987].

Perinteisiä relaatiotietokantoja voidaan käyttää tallentamaan XML-relaatioita, mutta vain osin käsittelemään niitä. Relaatiotietokantamalliin on tämän vuoksi toteutettu laajennus, joka mahdollistaa XML-relaation käsittelyn relaatiotietokannoissa. Niemi ja Järvelin [2006] esittelevät tämän laajennuksen formaalissa muodossa. Laajennuksen avulla relaatiotietokannasta on mahdollista hakea mm. elementtien välillisiä ja välittömiä jälkeläisiä ja vanhempia, palauttaa XML-dokumentista sen osia, sekä poistaa että lisätä siihen rakenteita.

XML-dokumenttien rakenne visualisoidaan puuna, jolla on juuri. XML-rakennepuussa on sekä lehti- että sisäsolmuja. Lehtisolmut ovat solmuja, joilla ei ole jälkeläisiä ja joihin puussa esitetty hierarkia päättyy, kun taas sisäsolmujen osalta hierarkia jatkuu. Visualisoinnin kannalta on tärkeää tunnistaa näiden solmujen erot. XML-relaatio mahdollistaa:

1) puussa olevien erilaisten solmujen löytämisen ja
2) niiden välillä vallitsevan hierarkian muodostamisen.

Luvussa 4 esiteltävä XML-dokumentin rakennepuu on tietoalkioiden (elementtien tai attribuuttien) nimistä koostuva puu. Jokaiseen rakennepuun solmuun liittyy siten yksi tai useampi elementin tai attribuutin ilmentymä. Yhteen tietoalkioon ei kuitenkaan voi liittyä sekä elementin että attribuutin ilmentymiä ts. samalla hierarkiatasolla esiintyvät samannimiset elementtien ja attribuuttien ilmentymät erotetaan rakennepuussa toisistaan. XML-relaatiossa jokaisella elementin ja attribuutin ilmentymällä on oma indeksinsä, joka on helppo liittää tunnisteeksi rakennepuun solmuihin (tietoalkioihin). Näiden indeksien kautta siten tiedetään, mistä elementtien ja attribuuttien ilmentymistä rakennepuun tietoalkiot on muodostettu.

---

[2] Yksinkertainen näkymä nimeltä *Elements*, joka sisältää kaikki Tietokanta-nimisen XML-relaation elementtien ilmentymät, luodaan XML-relaatiota hyödyntävässä relaatiotietokannassa ilmaisulla *create view Elements as select * from Tietokanta where type='e';*

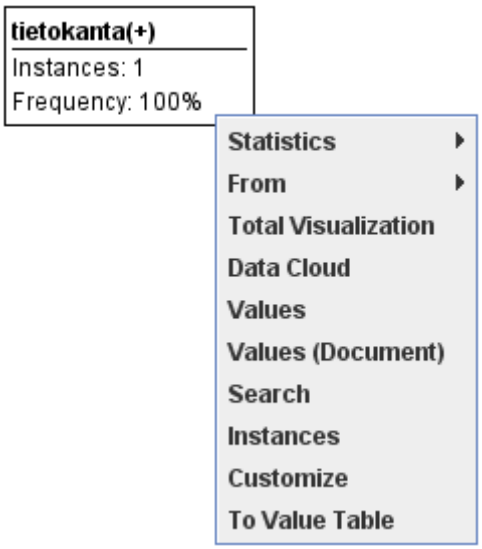
Tutkielmassa esiteltävään visualisointiin on sisällytetty myös tiedonhaullisia menetelmiä. Näiden menetelmien tarkoituksena on lisätä käyttäjän tietämystä dokumentista. Tietopilven muodostaminen on yksi näistä menetelmistä. Tietopilvi (alakohta 4.1.2) on visuaalinen esitys XML-dokumentissa tai sen alipuissa esiintyvistä sanoista. Sanan fonttikoko ilmaisee sen esiintymisen suhteellisen osuuden käsiteltävässä dokumentissa tai sen osassa. XML-relaatio tukee esiintymistiheyden laskemista, koska jokaisella dokumentissa esiintyvän sanan esiintymiskerralla on myös rivi XML-relaatioesityksessä. Indeksiarvoilla ilmaistaan, mistä kyseisen sanan esiintymät löytyvät. Tätä ominaisuutta voidaan hyödyntää yksittäisten sanojen etsinnässä. XML-relaatiossa kaavio- ja ilmentymätasoihin liittyvät tiedot erotetaan toisistaan T-attribuuttia (tyyppi) käyttäen. Tämä informaatio voidaan ilmaista tietopilvessä käyttämällä eri värejä.

4 XML-RELAATION VISUALISOINTI

Näppilä ja Niemi [2012] määrittelevät XML-dataspacen yhteydessä yhdeksi tärkeimmäksi tavoitteeksi kehittää mekanismeja, joilla kyetään tutkimaan tuntemattomien, heterogeenisten ja autonomisten tietolähteiden sisältöä, rakennetta ja semantiikkaa. Tässä luvussa esitellään näiden tavoitteiden saavuttamiseksi kehitettyjä visuaalisia työkaluja.

4.1 Visualisoinnin kohteet

XML-relaation visualisoinnissa termillä *tietoalkio* (engl. data item) tarkoitetaan käsittelyn kohteena olevan XML-dokumentin tietyllä tasolla olevien elementtien ja attribuuttien nimiä. Kuvassa 5 on esitettynä visualisointiohjelman käyttöliittymä. Ponnahdusvalikko saadaan esiin painamalla visualisoidussa rakennepuussa hiiren toissijaista painiketta halutun tietoalkion kohdalla.



Kuva 5. Visualisointiohjelman käyttöliittymä

4.1.1 Dokumentin rakennekuvaus

Visualisoinnin pääasiallisena kohteena on XML-dokumentin rakenne, joka koostuu dokumentin elementtien ja attribuuttien nimistä ja niiden keskinäisistä suhteista. Tietoalkiot esittävät tässä visuaalisessa kuvauksessa kaikkia kyseisellä tasolla esiintyviä tietyn elementin tai attribuutin ilmentymiä. Tietoalkiot visualisoidaan nimettyinä suorakulmioina, joiden väliset viivat ilmaisevat niiden keskinäiset suhteet. Ilmentymätason kuvausta, eli elementtien ja attribuuttien ilmentymissä esiintyviä arvoja, ei siis visualisoida lainkaan rakennekuvauksen

yhteydessä. XML-dokumentti visualisoidaan juurellisena puuna. Jokainen esiintymä ei kuitenkaan noudata visualisointia, koska puolirakenteisuuden luonteesta johtuen jotkut tiedot voivat puuttua joistakin tietoalkioiden esiintymistä. Toisin sanoen jokaisella hierarkiatasolla visualisoidaan maksimaalinen tietosisältö, joka dokumentin eri tietoalkioiden esiintymissä voi esiintyä. Periaatteessa on siis mahdollista, että tietyllä hierarkiatasolla esiintyvä maksimaalinen tietosisältö ei esiinny sellaisenaan missään ilmentymässä.

Visualisoinnissa käyttäjä voi joko laajentaa rakennepuuta asteittain, kartoittaen siten tuntemattomia tekijöitä vähitellen, tai konstruoida se kokonaisuudessaan. Myös välimuodot ovat mahdollisia. Hiiren osoittimen ollessa tietoalkion päällä vaihtuu kyseisen tietoalkion väri siniseksi, joka ilmaisee sen olevan aktiivisen mielenkiinnon kohteena. Käyttäjä voi tämän jälkeen laajentaa rakennepuuta klikkaamalla tietoalkiota hiiren ensisijaisella painikkeella. Mikäli käyttäjä haluaa visualisoida rakennepuun kokonaisuudessaan, tulee hänen valita ponnahdusvalikosta aktivoidun tietoalkion yhteydessä **Total Visualization** -komento.

XML-dokumentissa attribuuteilla ei voi olla jälkeläisiä, joten ne ovat visualisoidussa puussa aina lehtisolmuina. Alirakenteen sisältävät elementit (poislukien juurielementti) ovat puussa sisäsolmuja. Visualisoinnissa alirakenteettomat elementit ja attribuutit ovat samassa asemassa, ja ne erotetaan toisistaan värikoodauksella. Attribuutteja esittävien tietoalkioiden visualisoinnissa käytetään harmaata väriä, kun taas alirakenteettomat elementit visualisoidaan samalla värillä kuin muutkin elementit (valkoisella).

XML-dokumentti visualisoidaan hierarkkisena puuna, joka koostuu suorakulmioista ja niitä yhdistävistä viivoista. Suorakulmiolla kuvataan ja nimetään dokumentissa oleva tietoalkio. Kaksi nimettyä suorakulmiota yhdistävä viiva esittää vanhempi-lapsi-suhteen. XML-dokumentin visuaalinen esitystapa ilmaisee käyttäjälle tietoalkion nimen, tyypin (lehtisolmu, mikäli nimen perässä ei ole laajentuvuutta ilmaisevaa (+/-)-merkkiä, sisäsolmu tai juurisolmu), tietoalkioiden esiintymien lukumäärän (Instances), sekä mikä on kyseisen elementin tai attribuutin ilmentymien prosentuaalinen frekvenssi rakenteessa (Frequency). Jos tietoalkio B riippuu välittömästi tietoalkiosta A (ts. A on B:n vanhempi), niin prosentuaalinen frekvenssi ilmaisee, kuinka monella prosentilla A:n esiintymistä on välittömänä jälkeläisenään myös B:n esiintymä. Esimerkiksi kuvassa 6 esitetyssä XML-dokumentissa jokaisella *yliopisto*-elementin esiintymällä on *nimi*-attribuutti, joten niiden välinen prosentuaalinen frekvenssi on 100 %. Ainoastaan Tampereen yliopiston opiskelijoilla on

attribuutti *id*, eli neljästä *opiskelija*-elementin ilmentymästä vain kahdella on välittömänä jälkeläisenään *id*-attribuutti. Tästä seuraa, että suhteen frekvenssi on 50 %. Pelkkä ilmentymien lukumäärä ei kerro, missä suhteessa kyseinen elementti tai attribuutti esiintyy, koska yhdellä elementillä saattaa olla useampi kuin yksi samanniminen välitön jälkeläinen. Näin ollen suhteellista frekvenssiä tarvitaan heterogeenisten tekijöiden selvittämiseksi dokumentin rakenteesta. XML-dokumentilla on aina yksi ja vain yksi juuri. Kuvan 6 dokumentissa se on elementti *tietokanta*. Juurella ei ole vanhempaa, mutta koska se esiintyy aina, on sen frekvenssi dokumentin suhteen 100 %. Koska yhdessä XML-dokumentissa esiintyy aina vain yksi juurisolmu, on juurisolmujen ilmentymien lukumäärä siten myös aina yksi.

C	T	I
tietokanta	e	<1>
yliopisto	e	<1,1>
nimi	a	<1,1,1>
Tampereen	v	<1,1,1,1>
yliopisto	v	<1,1,1,2>
opiskelija	e	<1,1,2>
id	a	<1,1,2,1>
12345	v	<1,1,2,1,1>
Veikko	v	<1,1,2,2>
Virtanen	v	<1,1,2,3>
opiskelija	e	<1,1,3>
id	a	<1,1,3,1>
23456	v	<1,1,3,1,1>
Pekka	v	<1,1,3,2>
Pekkanen	v	<1,1,3,3>
yliopisto	e	<1,2>
nimi	a	<1,2,1>
Turun	v	<1,2,1,1>
yliopisto	v	<1,2,1,2>
opiskelija	e	<1,2,2>
Tapani	v	<1,2,2,1>
Timonen	v	<1,2,2,2>
opiskelija	e	<1,2,3>
Veikko	v	<1,2,3,1>
Veikkola	v	<1,2,3,2>

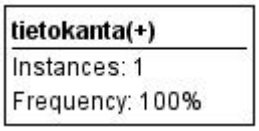
Taulu 2. XML-relaatioesitys kuvan 6 (Tietokanta.xml) XML-dokumentista

```
<tietokanta>
<yliopisto nimi="Tampereen yliopisto">
<opiskelija id="12345">
Veikko Virtanen
</opiskelija>
<opiskelija id="23456">
Pekka Pekkanen
</opiskelija>
</yliopisto>
<yliopisto nimi="Turun yliopisto">
<opiskelija>
Tapani Timonen
</opiskelija>
<opiskelija>
Veikko Veikkola
</opiskelija>
</yliopisto>
</tietokanta>
```

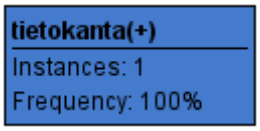
Kuva 6. Tietokanta.xml

Kuvassa 6 esitetyn XML-dokumentin asteittainen visualisointi esitetään kuvissa 7–10. Visualisointi aloitetaan avaamalla haluttu XML-dokumentti. Tämän jälkeen ohjelma visualisoi automaattisesti ko. dokumentin juurisolmun. Kuvassa 7 on esillä avatun XML-dokumentin juurisolmu *tietokanta*. Siihen voidaan liittää ainoastaan yksi XML-relaatioesityksessä oleva kaaviotason komponentti, ja kyseiseen komponenttiin liittyy aina indeksiarvo <1>. Muualta puusta ei voi löytyä samannimistä tietoalkiota (nimettyä suorakulmiota). Puun visualisointia lähdetään laajentamaan kyseisestä juurisolmusta. Kuvassa 8 juurisolmun taustaväri on muuttunut siniseksi, mikä ilmaisee kyseisen solmun olevan aktivoitu kohde. Kuvassa 9 on rakennepuuta on laajennettu yhden hierarkiatason verran klikkaamalla juurisolmua. Puun juurella on lapsenaan *yliopisto*-tietoalkio, joka pitää sisällään kaksi *yliopisto*-elementin ilmentymää (indeksit <1,1> ja <1,2>). Kuvassa 10 rakennepuuta on jälleen laajennettu yhdellä hierarkiatasolla, ja tämän seurauksena puuhun syntyy ensimmäinen haarauma. Uudet tietoalkiot ovat *nimi* ja *opiskelija*. Värityksellään *nimi*-tietoalkio on harmaa ja *opiskelija* valkoinen. Tämä ilmaisee, että *nimi* on muodostettu attribuuttien ilmentymistä ja *opiskelija* elementtien ilmentymistä. Kuvasta huomataan myös, että *nimi*-tietoalkioon liittyy kaksi esiintymää (indeksit <1,1,1> ja <1,2,1>) ja o*piskelija*-tietoalkioon neljä (indeksit <1,1,2>, <1,1,3>, <1,2,2> ja <1,2,3>). Tietoalkio *nimi* on

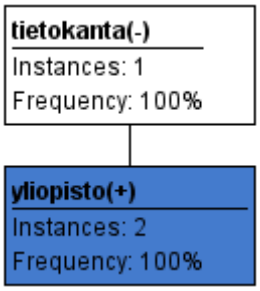
rakennepuussa lehtisolmuna, joten puun visualisointia ei voida enää laajentaa tästä solmusta. Kuvassa 11 rakennepuuta on laajennettu uudella hierarkiatasolla, joka kuvaa attribuuttina esitettävän *id*-tietoalkion. Tähän tietoalkioon liittyy kaksi esiintymää, joiden indeksit ovat <1,1,2,1> ja <1,1,3,1>. Visualisointi on päättynyt, koska ko. tietoalkion huomataan olevan lehtisolmuna. Kuvassa on siten käsittelyn kohteena olevasta XML-dokumentista muodostettu täydellinen rakennepuu.



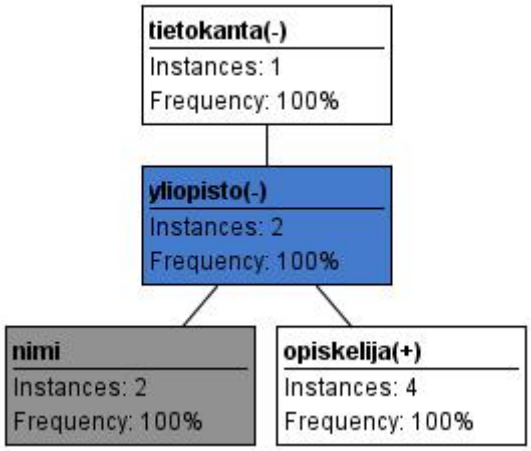
Kuva 7. Visualisoinnin alkutilanne, ensimmäinen taso



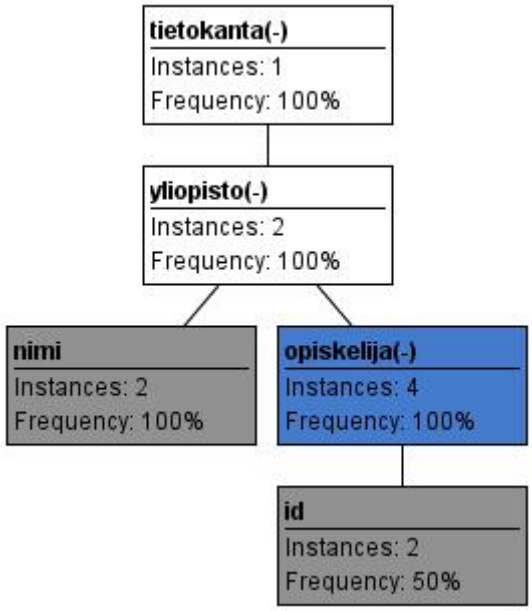
Kuva 8. Puun asteittainen laajentaminen aloitetaan juurisolmusta



Kuva 9. Toisen hierarkiatason visualisoinnin generointi



Kuva 10. Kolmannen hierarkiatason visualisoinnin generointi



Kuva 11. Kuvan 6 XML-dokumenttin liittyvä rakennepuu kokonaan visualisoituna

4.1.2 Tietopilvi

Tietopilvi (engl. data cloud) on visualisointi, joka muodostetaan XML-dokumentissa esiintyvistä sanoista. Sen tarkoituksena on kuvata tietolähteen sisältöä. Tietopilvessä visualisoidaan sanapilvestä poiketen kaikki mahdollinen tekstuaalisessa muodossa esitettävä tieto, ei pelkästään luonnollisen kielen sanoja. Dokumentissa useammin esiintyvät sanat erotetaan harvinaisemmista sanoista asettelulla tai käyttämällä erilaisia visuaalisia esitysmuotoja. Usein esiintyvä sana voidaan esimerkiksi sijoittaa keskeisemmälle paikalle kuin harvinaisempi sana, tai sen visualisointiin voidaan käyttää tiettyä väriä. Bateman *et al.* [2008] selvittivät sanapilviä koskevassa tutkimuksessaan, että fontin koko ja tummuus ovat parhaita keinoja sanojen merkittävyyden kuvaamisessa. Rivadeneira *et al.* [2007] havaitsivat sanapilvistä, että sanojen paikka pilven sisällä on merkityksellinen: käyttäjät huomasivat vasemmassa yläkulmassa olevat sanat parhaiten. Ero muihin näytön/pilven sektoreihin verrattuna ei ollut suuri, mutta se oli kuitenkin tilastollisesti merkitsevä. He epäilivät tämän syyksi länsimaista kirjoitusperinnettä, jossa teksti alkaa sivun vasemmasta yläkulmasta. Tutkimuksessa havaittiin myös, että sanapilvistä on apua käyttäjälle kohteen tietosisällön hahmottamisessa ja käsittelyssä: sanapilvi auttaa muun muassa kohteen selailussa, helpottaa yksittäisten kohteiden etsintää, edistää kokonaiskuvan muodostumista tietosisällöstä ja auttaa oikean semanttisen tulkinnan liittämistä kohteeseen.

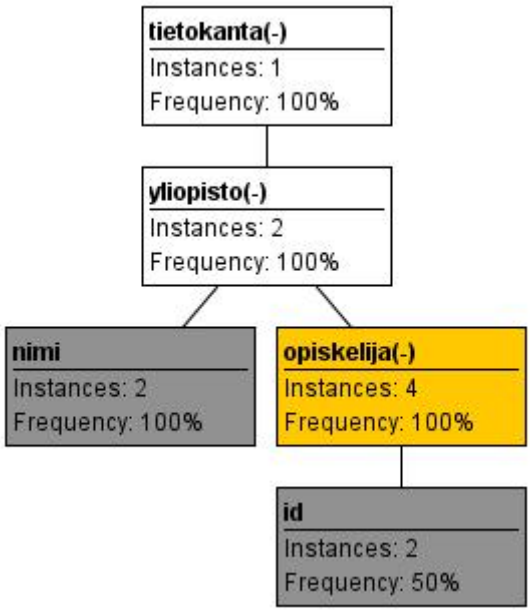
Tätä tutkielmaa varten toteutettu tietopilvi voidaan muodostaa XML-dokumenteista tai niiden osista. Tietopilvi muodostetaan aina tietystä rakennepuussa olevasta tietoalkiosta alkavaan osarakenteeseen liittyen. Käyttäjä valitsee halutun tietoalkion, johon perustuen visualisointimekanismi luo kyseisen tietoalkion esiintymissä ja niistä riippuvien tietoalkioiden esiintymissä olevista sanoista tietopilven. Mikäli tietopilvi muodostetaan visualisoidun rakennepuun juuresta, vastaa tietopilven sisältö koko dokumentin sisältöä.

XML-dokumentin pohjalta muodostettu tietopilvi pitää sisällään sekä dokumentin kaavio- että ilmentymätasoilla esiintyvät tiedot. Tietopilveen otetaan siis mukaan niin elementtien ja attribuuttien nimet kuin niiden tekstuaalisissa arvoissa esiintyvät sanat. Kaavio- ja ilmentymätasoilla esiintyvät arvot erotetaan toisistaan käyttämällä niiden visualisoimiseen eri värejä. Elementtien ja attribuuttien nimet esitetään käyttäen sinistä ja ilmentymätason sanat käyttäen punaista väriä. Kuvan 6 dokumentissa esiintyy sana ’yliopisto’ sekä elementin nimenä että tekstuaalisen arvon osana. Se siis esiintyy kaavio- ja ilmentymätasoilla. Kyseinen sana muodostaa siten ongelman värityksensä suhteen. Tässä kuvatussa tietopilvessä käytetään menetelmää, jossa sanalle lasketaan RGB-värimallin[3] mukainen arvo punaisen (255-0-0) ja sinisen (0-0-255) väliltä, käyttäen painotuksina kummankin tason ilmentymien määrää. Tästä seuraa, että sanan väritys on lähempänä sinistä, jos sen esiintymiä on enemmän kaaviotasolla ja lähempänä punaista, jos sen esiintymiä on enemmän ilmentymätasolla. Sanan ’yliopisto’ tapauksessa molemmilla tasoilla on yhtä monta (kaksi) esiintymää, joten sanan sen RGB-arvoksi (pyöristystä käyttäen) tulee 127-0-127 (*(255 * 2 + 0 * 2) / 4, (0 * 2 + 0 * 2) / 4 , (255 * 2 + 0 * 2) / 4*), joka vastaa violettia väriä. Kuvassa 12 on esimerkkidokumentista (kuva 6 / Tietokanta.xml) muodostettu tietopilvi.

Kuva 12. Esimerkkidokumentista muodostettu tietopilvi

[3] RGB-malli on punaista (R), vihreää (G) ja sinistä (B) valoa yhdistämällä luotu väriavaruus, jossa valot saavat (tässä) arvoja väliltä 0-255.

Tutkielmassa kehitetyssä tietopilvessä sanat esitetään aakkosjärjestyksessä. Numerot ja symbolit sijoitetaan tietopilven alkuun, vasemmasta ylänurkasta alkaen. Tällainen järjestys saattaa tukea arvojen välillä olevien tietokonfliktien havaitsemista, koska toisiaan aakkosnumeerisesti lähellä olevat sanat esiintyvät pilvessä lähellä myös toisiaan. Käyttäjän osoittaessa hiirellä tietopilvessä olevaa sanaa muuttuvat rakennevisualisoinnissa kyseisen sanan sisältävät tietoalkiot keltaisiksi (kuva 13).



Kuva 13. Tietopilvessä on osoitettu arvoa 'Veikko'

Mikäli taustaväriltään keltaiseksi muuttuneen tietoalkion nimi ei ole sama kuin tietopilvessä osoitettu sana, sijaitsee sana ilmentymätasolla. Toisin sanoen se esiintyy kyseisen tietoalkion jonkin esiintymän arvona tai sen osana. Kuvassa 13 on esitettynä rakenteen visualisointi tapauksessa, jossa hiirellä on osoitettu sanaa 'Veikko' kuvan 12 tietopilvessä. Tietopilvessä sijaitsevan arvon väritys (punainen) ja taustavärinsä keltaiseksi muuttaneen *opiskelija*-tietoalkion nimi ilmaisevat käyttäjälle, että kyseessä on *opiskelija*-tietoalkion esiintymän arvoon liittyvä sana. Käyttäjä kykenee päättelemään tästä, että *opiskelija*-tietoalkio sisältää opiskelijoiden etunimiä. Osoitetun sanan fontin koko on selvästi suurempi kuin pienimmillä sanoilla. Pienin mahdollinen fontti vastaa minimissään yhtä esiintymää. Tästä voidaan päätellä, että sana 'Veikko' esiintyy dokumentissa useamman kuin yhden kerran. Käyttäjä voi jatkaa tietoalkion tutkimista tarkemmin ja saada siten varmistuksen solmuun liittyvästä oikeasta semantiikasta.

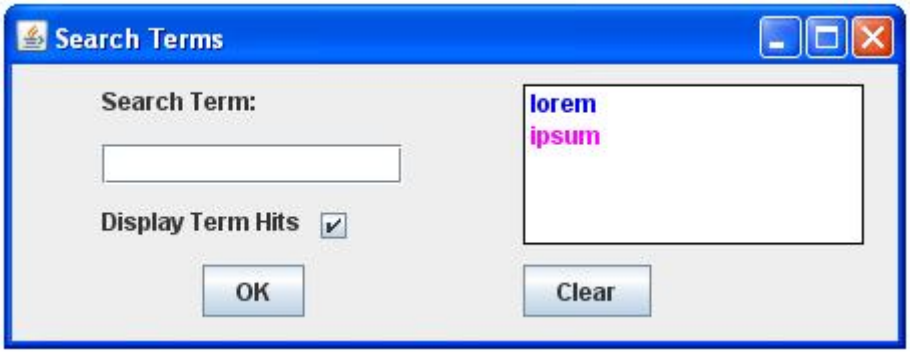
4.1.3 Ilmentymätaso

Käyttöliittymään on kehitetty visualisointimekanismeja, joilla voidaan tutkia ilmentymätason tietoja. Käyttäjällä on valittavanaan kaksi erilaista visuaalista mekanismia näiden tietojen tarkasteluun. Taulumuotoinen esitys soveltuu etenkin tieto-orientoituneiden lähteiden käsittelyyn eli lähteisiin, jotka sisältävät paljon numeerista ja/tai listamaista tietoa. Siinä visualisoidaan kaikki kyseisen tietoalkion arvot yhdellä kertaa. Tämä esitys saadaan tuotettua valitsemalla käyttöliittymästä komento **Values** aktiivisena olevan tietoalkion yhteydessä. Kuvassa 14 esitetty taulu sisältää neljän eri o*piskelija*-elementin esiintymien arvot eli sen ilmentymätason tiedot.

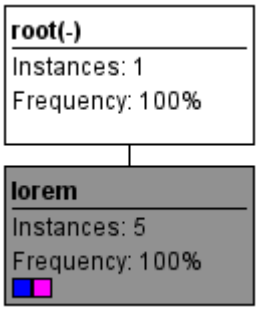
Kuva 14. *opiskelija*-tietoalkion ilmentymätason arvot visualisoituna

Dokumenttiorientoituneita XML-lähteitä varten voidaan luoda visualisointi, jossa jokaisen elementin tai attribuutin ilmentymän arvot visualisoidaan yksitellen. Visuaalinen mekanismi yksittäisten arvojen tarkasteluun saadaan esiin komennolla **Values (Document)**. Käyttäjän on mahdollista määritellä kyseiselle mekanismille yksittäisiä merkkijonoja, joita visualisoinnissa korostetaan. Korostus tapahtuu käyttämällä näiden merkkijonojen tulostamiseen muusta tekstistä poikkeavia värejä. Merkkijonot syötetään erillisen valikon kautta, joka on esitettynä kuvassa 15. Siinä korostettavat merkkijonot ovat ”lorem” sinisellä ja ”ipsum” magentanvärisenä. Hakusanojen merkkikoolla ei ole merkitystä. Toiminto on samankaltainen kuin normaalin tekstieditorin hakutoiminto, jossa haun palauttamat sanat erotetaan muusta tekstistä. Erotuksena on kuitenkin mahdollisuus määritellä etukäteen useita haettavia/korostettavia merkkijonoja. Mikäli käyttäjä on valinnut ruudun ”Display Term Hits”, piirretään rakennepuun niihin tietoalkioihin, jotka sisältävät ennalta määritellyn merkkijonon, kyseistä hakutermiä vastaavalla värillä palkkikuvio (kuva 16).

Esimerkkidokumentti koostuu juurielementistä *root* ja viidestä Lorem ipsum[4] -arvoja sisältävästä *lorem*-elementistä.



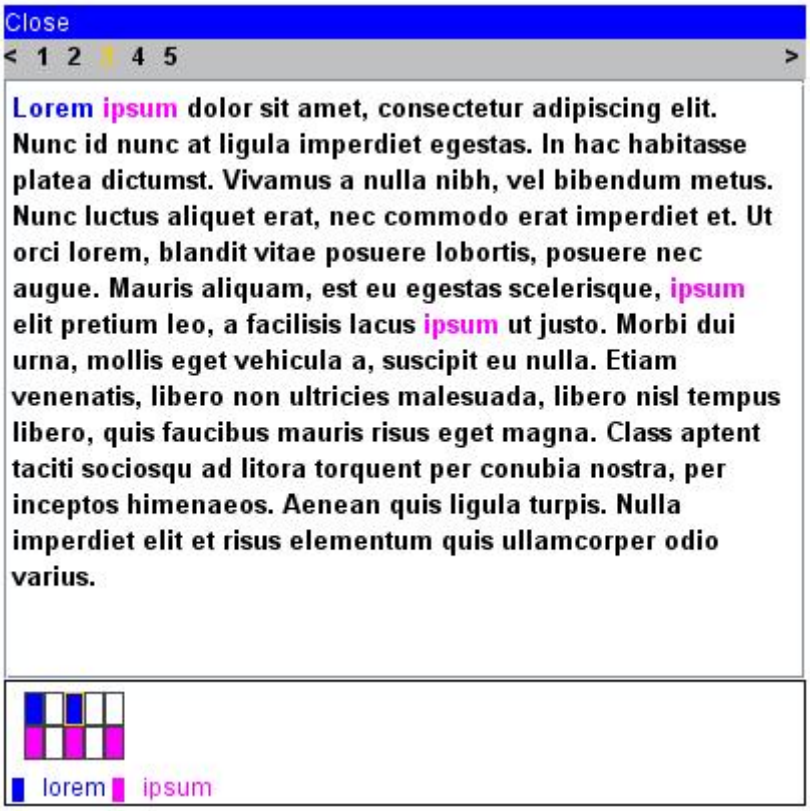
Kuva 15. Hakutermien syöttäminen



Kuva 16. Väritetyt palkit ilmaisevat löytyneet hakutermit

Käyttäjä voi selailla ilmentymien arvoja yläkulmissa olevien ohjausmerkkien avulla tai valita käsiteltävän ilmentymän numeron suoraan. Visualisoitua ilmentymää vastaava numero ilmaistaan käyttämällä keltaista väriä. Visualisoinnin alalaidassa on Tilebar-tyylinen [Hearst, 2001] valikko, joka ilmaisee hakutermien esiintyvyyden tietoalkion ilmentymissä. Palkit vastaavat ilmentymiä siten, että ensimmäinen palkki vasemmalta vastaa ensimmäistä ilmentymää, toinen toista, jne. Käyttäjä voi avata ilmentymiä klikkaamalla haluttua palkkia. Väritetty palkki ilmaisee, että sitä vastaavasta ilmentymästä löytyy ko. palkin väriä vastaava etsitty merkkijono. Kyseinen mekanismi eroaa alkuperäisestä Tilebar-visualisoinnista siten, ettei se ota huomioon termien esiintymien runsautta suhteessa ilmentymän kaikkiin merkkijonoihin.

[4] Lorem ipsum on ns. täyteteksti, jonka tarkoituksena on havainnollistaa ulkoasua.



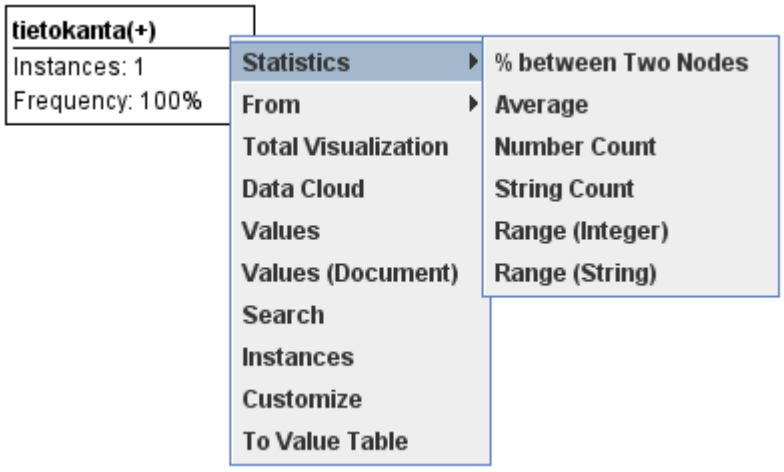
Kuva 17. Kolmannen *lorem*-elementin esiintymän sisältö

4.1.4 Dokumentin tunnusluvut

Visualisointiohjelmaan on kehitetty apuvälineitä dokumentin erilaisten ominaisuuksien selvittämiseen. Näitä ominaisuuksia kutsutaan tässä dokumentin tunnusluvuiksi ja käyttäjä valitsee ne **Statistics**-valikosta. Niiden avulla käyttäjä kykenee arvioimaan millaisia ilmentymätason komponentteja tietoalkioihin liittyy. Selvittämällä tietoalkioiden arvoja, käyttäjä kykenee tätä informaatiota hyödyntämällä lisäämään ymmärrystään tietoalkioista ja niiden keskinäisistä suhteista ja sitä kautta tietoalkioihin liittyvästä semantiikasta. Kuvassa 18 on esitelty käyttöliittymä tunnuslukujen laskemiseksi. Järjestelmän käyttäjän on mahdollista valita kuuden eri tunnusluvun väliltä.

Tunnusluvut lasketaan tietoalkiokohtaisesti, lukuun ottamatta kahden eri tietoalkion esiintymien välistä esiintymisprosenttia, joka lasketaan **% between Two Nodes** -komennolla. Sillä voidaan laskea, miten yleinen tietyssä visualisointipuussa oleva tietoalkio on suhteessa siitä riippuvaan tietoalkioon. Toisin sanoen lasketaan todennäköisyys sille, että XML-dokumentin tietyn nimisen elementin esiintymällä on jälkeläisenään tietyn nimisen tietoalkion esiintymä. Tämä tunnusluku eroaa tietoalkioissa esitetystä Frequency-arvosta siten, että sillä

voidaan laskea esiintymisprosentti kaikkien jälkeläisten suhteen, ei siis pelkästään välittömien jälkeläisten. Attribuuttien sisältönä ei XML:ssä voi olla kaaviotason tietoa. Tämän takia attribuutin esiintymisprosentti minkään elementin tai toisen attribuutin vanhempana on aina nolla. Esimerkiksi *yliopisto*-tietoalkion ja *id*-tietoalkion välille laskettu prosentti ilmaisee, kuinka monella kuinka monella *yliopisto*-elementin esiintymällä on ainakin yksi sellainen elementin esiintymä, jolla on välittömänä jälkeläisenään *id*-niminen attribuutti.



Kuva 18. Tietokanta.xml - Tunnuslukujen käyttöliittymä

Number Count -tunnusluku ilmaisee, kuinka monta numeraalista arvoa valittu tietoalkio sisältää. Sillä voidaan siis selvittää, sisältävätkö tietoalkion arvot esimerkiksi hinta- tai määräinformaatiota. Esimerkiksi solmu *id* sisältää kaksi numeraalista arvoa. Mikäli käyttäjä löytää tietyn tietoalkion arvoista numeerista tietoa, voidaan niistä laskea tietoalkiolle keskiarvo **Average**-komennolla. **Range (Integer)** ilmaisee tällaisen tietoalkion arvoina olevien kokonaislukujen vaihteluvälin. **String Count** ilmaisee, montako sanaa annetun tietoalkion arvoista löytyy ja **Range (String)** -komentoa käytetään, kun halutaan tietää, mitkä ovat tietyn tietoalkion arvojen aakkosjärjestyksessä ensimmäinen ja viimeinen sana.

4.2 Visuaalisen vastineen johtaminen XML-relaatiosta

Tässä luvussa esitellään formalismit edellisessä luvussa esitellyille XML-relaation visuaalisille esitystavoille. Tällaista visuaalisen esityksen ja relaatiotietokannan taulun (eli XML-relaation) välistä kuvausta voidaan kutsua *visuaaliseksi metaforaksi* [Haber *et al.*, 1994]. Tarkkaan ottaen tämä on visuaalisen metaforan formalismi, jonka kautta voidaan verifioida visuaalisen esityksen oikeellisuus suhteessa tietomalliin

[Catarci *et al.,* 1995]. Se määrittelee siis, mikä on visuaalisen esityksen ja tietomallin välinen analogia.

4.2.1 Rakennepuun formaali esitys

Rakennepuun visualisoinnissa käytetään hyvin pelkistettyä kuvausta. Rakennepuu esitetään vain kahden visuaalisen primitiivin avulla. Nimettyjen suorakulmioiden avulla esitetään dokumentissa olevat tietoalkiot. Viivalla yhdistetään kaksi nimettyä suorakulmiota toisiinsa seuraavalla tulkinnalla: visualisoinnissa ylempänä oleva nimetty suorakulmio on siihen viivalla alempaa yhdistetyn nimetyn suorakulmion vanhempi. Visualisointi perustuu siihen, että tällä tavalla kuvataan kaikki vanhempi-lapsi-suhteet. Visualisoinnissa lehtitietoalkioilla ei ole lapsia, jolloin hierarkian konstruointi loppuu. Strukturoidulla tietomallilla (esimerkiksi relaatiomalli) on kiinteä rakenne, jonka mukaan ilmentymätaso on organisoitu. Sen sijaan XML-dokumentissa samannimisistä tietoalkioiden esiintymistä välittömästi riippuvat tietoalkiot voivat vaihdella huomattavasti. Tämä tekee XML-informaation visualisoinnista huomattavasti haastavampaa kuin esim. relaatiotietokannan relaation visualisoinnista. Tässä tutkielmassa rakennepuun visualisoinnissa sovelletaan maksimaalisuusperiaatetta, toisin sanoen tietyn nimisen tietoalkion lapsiksi visualisoidaan kaikki sen nimiset tietoalkiot, jotka riippuvat ko. tietoalkiosta välittömästi ainakin jossakin sen esiintymässä. Oletetaan, että tietoalkiolla, jonka nimi on A, on seuraavat kolme esiintymää XML-dokumentissa. Ensimmäisessä esiintymässä A:sta riippuuvat tietoalkioiden B, C, D esiintymät; toisessa esiintymässä A:sta riippuuvat tietoalkioiden C, D, E esiintymät ja kolmannessa esiintymässä siitä riippuvat tietoalkioiden B, D, F esiintymät. Tässä tapauksessa visualisoinnissa kaikki tietoalkiot A, B, C, D, E ja F esitettäisiin nimettyinä suorakulmioina ja A:sta piirrettäisiin viivat tietoalkioihin B, C, D, E ja F. Rakennepuun visualisointia voidaan luonnehtia rekursiivisella proseduurilla

Visualize(Name, I-Set, XRel).

Siinä *Name* on visualisoitavan tietoalkion nimi, *I-Set* kyseisen tietoalkion esiintymiin liittyvien indeksien joukko ja *XRel* visualisoinnin kohteena oleva XML-dokumentin XML-relaatioesitys. Koko rakennepuun visualisointi alkaa aina juurisolmusta. Aluksi piirretään juurisolmua vastaava tietoalkio, jonka jälkeen kutsutaan visualisointiproseduuria komennolla *Visualize(R, {<1>}, XRel)*, jossa *R* on juurisolmuna olevan elementin nimi. Puun juurisolmu

vastaa XML-dokumentin juurielementtiä, ja siihen liittyy ainoastaan yksi indeksiarvo *<1>*. Visualisointi tapahtuu seuraavien vaiheiden kautta:

1) Selvitetään kaikkien joukossa *I-Set* olevien indeksien välittömät jälkeläiset (ts. *Namen* lapset), jotka voivat olla elementtejä tai attribuutteja. Välittömiin jälkeläisiin liittyvä tietoalkioiden nimien joukko on *N* ja näiden tietoalkioiden indekseistä muodostettu joukko on *Z1*. Joukon *N* muodostaminen tapahtuu poistamalla duplikaatit elementtien ja attribuuttien nimistä.
2) Muodostetaan joukko *Set1*, joka koostuu *(DataItemName, I-Set2)*-pareista siten, että *DataItemName* on joukossa *N* oleva tietoalkion nimi ja *I-Set2* siihen liittyvien indeksien joukko. Toisin sanoen *I-Set2* on joukon Z1 osajoukko.
3) Piirretään joukossa *Set1* esiintyvät tietoalkiot, jonka jälkeen piirretään viiva *Name*-nimisestä suorakulmiosta jokaiseen *Set1*-joukon aikaisemmin visualisoituun tietoalkioon.
4) Iteroidaan joukko *Set1={ (DataItemName(1),I-Set2(1)), (DataItemName(2), I-Set2(2)),…,(DataItemName(n), I-Set2(n)) }* läpi ja kutsutaan jokaiselle parille visualisointiproseduuria *Visualize(DataItemName(i), I-Set2(i), XRel)*. Mikäli *Set1* on tyhjä joukko, on piirrettävä solmu lehtisolmu ja rekursio päättyy.

Kuvassa 6 esitetyn XML-dokumentin rakennepuu visualisoidaan seuraavasti:

Piirretään tietoalkio *tietokanta* (ks. kuva 7) suorakulmiona, jonka jälkeen kutsutaan proseduuria

Visualize(tietokanta, {<1>}, Tietokanta.xml), jonka jälkeen:

1) Selvitetään kaikki välittömästi indeksiarvoon <1> liittyvät elementtien ja attribuuttien ilmentymien nimet. Näitä ilmentymiä esiintyy kaksi kappaletta nimellä *yliopisto*. Muodostetaan niistä joukko *N* = {yliopisto}. Tämän jälkeen selvitetään, mitkä indeksit vastaavat joukon *N* alkiota, ja muodostetaan niistä joukko *Z1* = {<1,1>, <1,2>}.
2) Muodostetaan joukko *Set1*={ (yliopisto, {<1,1>,<1,2>) }
3) Piirretään tietoalkio *yliopisto* suorakulmiona, ja piirretään siihen viiva proseduurin kutsussa olleesta parametrista *tietokanta* (juurielementin nimi, ks. kuva 9).

4) Koska joukossa *Set1* on ainoastaan yksi alkio, kutsutaan proseduuria *Visualize(yliopisto, {<1,1>,<1,2>}, Tietokanta.xml)*
5) Selvitetään kaikki välittömästi indeksiarvoihin <1,1> ja <1,2> liittyvät elementtien ja attribuuttien ilmentymien nimet ja näihin liittyvät indeksit, ja luodaan niistä joukot *N2* = {nimi, opiskelija} ja *Z2* = {<1,1,1>,<1,1,2>, <1,2,1>, <1,2,2>}
6) Muodostetaan joukko *Set2*={(nimi,{<1,1,1>,<1,2,1>}), (opiskelija, {<1,1,2>, <1,2,2>})}
7) Piirretään tietoalkio *nimi* sekä tietoalkio *opiskelija* suorakulmioina ja piirretään niihin viivat tietoalkiosta *yliopisto* (ks. kuva 10).
8) Koska *nimi*-tietoalkion indekseillä ei ole välittöminä jälkeläisinään attribuuttien tai elementtien ilmentymiä, päättyy visualisointi kyseisen haaran osalta. *Opiskelija*-tietoalkion indekseillä on välittöminä jälkeläisinään id-attribuuttien ilmentymiä. Muodostetaan niiden nimistä ja indekseistä joukot N3 = {id} ja Z3 = {<1,1,2,1>,<1,1,3,1>}
9) Muodostetaan joukko *Set3*={ (id, {<1,1,2,1>,<1,1,3,1>})}
10) Piirretään tietoalkio *id* suorakulmiona ja yhdistetään siihen viiva tietoalkiosta *opiskelija* (ks. kuva 11).
11) *id*-tietoalkion indekseillä ei ole välittöminä jälkeläisinään attribuutteja eikä elementtejä, joten visualisointi päättyy sen osalta. Koska kyseessä on viimeinen avoinna oleva rakennepuun haara, on visualisointi päättynyt.

4.2.2 Tietopilven formaali esitys

Tutkielmassa kehitetyn tietopilven esitystavan muodostaminen perustuu proseduuriin

Datacloud(Name, I-Set, XRel),

jossa *Name* on minkä tahansa rakennepuussa esiintyvän tietoalkion nimi ja *I-Set* on *Name:*n esiintymiin liittyvien indeksien joukko. Tietopilvi muodostetaan sanoista, jotka esiintyvät *I-Set:*ssä olevista indekseistä alkavista rakenteissa. Mikäli proseduurin parametrina annettu tietoalkio on juurielementti, muodostetaan tietopilvi tällöin koko XML-dokumentista. Proseduuri palauttaa joukon *(w, I)*-pareja, joissa *w* on yksittäinen sana *Name*-tietoalkiosta alkavassa alirakenteessa, *I* on indeksijoukko, joka sisältää kaikkien *w:*n esiintymien indeksit tarkastelun kohteena olevissa rakenteissa. Proseduuri käyttäytyy seuraavasti:

1) Muodostetaan joukko *I-Set2* niistä indekseistä, jotka esiintyvät joukossa *I-Set* olevan elementtien välittömiin tai välillisiin jälkeläisiin. Lisätään joukkoon *I-Set2* lisäksi kaikki joukon *I-Set* indeksit.
2) Muodostetaan joukko C niistä erilaisista sanoista, joiden indeksit esiintyvät joukossa *I-Set2*.
3) Muodostetaan joukko *Set1*, joka sisältää muotoa (w, *I-Set3)* olevia pareja. Parissa *w* on eräs joukossa C esiintyvä sana ja *I-Set3* on kyseiseen sanaan liittyvien indeksien joukko. *I-Set3* on joukon *I-Set2* osajoukko.

Datacloud-proseduurilla tuotettu joukko *Set1* välitetään parametrina proseduurille

VisualizeDatacloud(Set, n), missä *n* on visualisoitavien sanojen määrä.

Proseduuri visualisoi joukossa *Set* olevat sanat seuraavasti:

1) Muodostetaan joukosta *Set* osajoukko *Set1*, joka sisältää proseduurin parametrina annetun (*n*) määrän joukon *Set* indeksimäärältään suurinta paria. Mikäli *n* on suurempi kuin *Set*:ssä esiintyvien parien määrä, otetaan kaikki parit joukkoon *Set1*.
2) Asetetaan joukossa *Set1* esiintyvien parien indeksimäärältään suurin luku eli suurin frekvenssi *fmax* vastaamaan suurinta mahdollista fonttia *max*, ja pienin frekvenssi *fmin* vastaamaan pienintä mahdollista fonttia *min*.
3) Lasketaan jokaiselle sanalle *w* fontin koko *s* kaavalla *min + ((f – fmin) / (fmax – fmin)) * (max – min)*, jossa *f* ilmaisee sanan frekvenssin.
4) Muodostetaan jokaiselle sanalle *w* RGB-värimallin mukainen arvo kaavalla *255 * a / (a+b), 0, 255 * b / (a+b)*, jossa *a* ja *b* ovat pariin $(w, I) \in Set1$ joukossa *I* liittyvät kaaviotason (*a*) ja ilmentymätason (*b)* indeksien lukumäärät.
5) Piirretään tietopilvi muodostettujen fonttikokojen ja RGB-arvojen perusteella.

Kuvassa 12 esitetty tietopilvi muodostetaan seuraavasti:

Aloitetaan visualisointi kutsumalla proseduuria

Datacloud(tietokanta, {<1>}, Tietokanta.xml), jossa:

1) Muodostetaan indeksijoukko *I-Set* = {<1>, <1,1>, <1,1,1>, <1,1,1,1>, <1,1,1,2>, <1,1,2>, <1,1,2,1>, <1,1,2,1,1>, <1,1,2,2>, <1,1,2,3>, <1,1,3>, <1,1,3,1>, <1,1,3,1,1>, <1,1,3,2>, <1,1,3,3>, <1,2>, <1,2,1>, <1,2,1,1>, <1,2,1,2>, <1,2,2>, <1,2,2,1>, <1,2,2,2>, <1,2,3>, <1,2,3,1>, <1,2,3,2>}
2) Muodostetaan joukko C = {tietokanta, yliopisto, nimi, Tampereen, opiskelija, id, 12345, Veikko, Virtanen, 23456, Pekka, Pekkanen, Turun, Tapani, Timonen, Veikkola}
3) Muodostetaan joukko järjestettyjä pareja, jotka koostuvat joukossa C olevasta sanasta ja siihen liittyvistä indeksiarvoista *Set1* = {(tietokanta, {<1>}), (nimi,{<1,1,1>,<1,2,1>}),(Tampereen,{<1,1,1,1>}), (yliopisto, {<1,1,>, <1,2>, <1,1,1,2>, <1,2,1,2>}), (opiskelija,{<1,1,2>, <1,1,3>, <1,2,2>, <1,2,3>}), (id, {<1,1,2,1> ,<1,1,3,1>}), (12345, {<1,1,2,1,1>}), (Veikko, {<1,1,2,2>, <1,2,3,1>}), (Virtanen, {<1,1,2,3>}), (23456, {<1,1,3,1,1>}), (Pekka, {<1,1,3,2>}), (Pekkanen, {<1,1,3,3>}), (Turun, {<1,2,1,1>}), (Tapani, {<1,2,2,1>}), (Timonen, {<1,2,2,2>}), (Veikkola,{<1,2,3,2>})}

Tämän jälkeen kutsutaan proseduuria

VisualizeDatacloud(Set1, 30), jossa:

1) Jälkimmäisellä parametrilla (*n*) ilmaistaan, että halutaan visualisoida 30 yleisintä sanaa. Koska *n:*llä on arvo 30 ja *Set1* sisältää 16 paria, asetetaan *n = 16*. Palautetaan joukko *Set2*, joka sisältää *n* suurinta frekvenssiä omaavat sanat. Tässä tapauksessa *Set2* sisältää samat alkiot kuin joukko *Set1*.
2) Joukon *Set2* suurin frekvenssi *fmax* on 4, joka esiintyy mm. parissa (opiskelija,{<1,1,2>, <1,1,3>, <1,2,2>, <1,2,3>}). Asetetaan suurin frekvenssi vastaamaan suurinta mahdollista fonttikokoa (20) *max = 20*. Joukon *Set2* pienin frekvenssi *fmin* on 1, joka esiintyy myös useassa parissa. Asetetaan pienin frekvenssi vastaamaan pienintä mahdollista fonttikokoa (10) *min = 10*:
3) Lasketaan jokaiselle joukossa *Set2* olevalle sanalle *w* fontin koko *s* kaavalla *min + ((f – fmin) / (fmax – fmin)) * (max – min)*, jossa *f* ilmaisee sanan frekvenssin. Sanalle ’opiskelija’ fontin kooksi tulee 20 (*10 + ((4 – 1) / (4 – 1)) * (20 – 10) = 20*). Sanoille, joilla on vain yksi esiintymä, tulee fontin kooksi automaattisesti 10.

4) Lasketaan jokaiselle sanalle *w* RGB-värimallin mukainen arvo. Ainoastaan sana ’yliopisto’ esiintyy sekä kaavio- että ilmentymätasoilla, joten sen väriarvo muodostetaan kaavalla *((255 * 2 / 4), 0, (255 * 2 / 4)) = (127,0,127)*, joka vastaa violettia väriä. Muut sanat saavat väriarvokseen joko punaisen (255,0,0) (kaaviotaso) tai sinisen (0,0,255) (ilmentymätaso).
5) Visualisoidaan tietopilvi laskettujen arvojen mukaisesti.

4.3 Visualisointiohjelman toteutus

Tässä tutkielmassa kehitetty visuaalinen käyttöliittymä on toteutettu Sun Microsystemsin kehittämässä Java-kehitysympäristössä (JDK 1.6). XML-tietojen tallettamiseen ja käsittelyyn käytetty relaatiotietokantajärjestelmä on PostgreSQL (PostgreSQL 9.0.5). Grafiikan tuottamiseen käytetään pääosin The Visual Library 2.0 -kirjastoa [NetBeans Team, 2007], joka on nykyään osa NetBeans-sovellusalustaa. Kyseinen kirjasto operoi nk. *ikkunaolioiden* (engl. widget) kautta. Ikkunaolio on olio, jonka ulkoasu voidaan määritellä ja joka kyetään esittämään visuaalisessa muodossa. Se sisältää oliokohtaisia jäsenmuuttujia ja jäsenfunktiota kuten tavanomainen olio.

4.3.1 Arkkitehtuuri

Visuaalinen käyttöliittymä on toteutettu noudattaen MVC-mallia [Burbeck, 1987/1992]. MVC on ohjelmistoarkkitehtuurityyppi, jonka nimi on lyhenne sanoista Model-View-Controller (malli–näkymä-ohjain). Se on tarkoitettu etenkin vuorovaikutteisille, graafisille järjestelmille, ja siinä käyttöliittymä on erotettu sovelluslogiikasta. Tämä mahdollistaa ohjelmakoodin uudelleenkäytön ja refaktoroinnin: samaa käyttöliittymää voidaan käyttää useassa sovelluksessa, eivätkä siihen tehdyt muutokset välttämättä edellytä muutoksia sovelluslogiikassa.

Malli sisältää kaiken ohjelman käyttämän tiedon sekä prosessit ja säännöt sen käsittelemiseksi. Tämä on myös eräs tietomallin määritelmä [Ullman, 1988]. Malli voi koostua muutamasta muuttujasta, relaatiotietokannan taulusta tai niiden yhdistelmästä. Sen sisältöä ei ole rajoitettu. Mallista voidaan muodostaa erilaisia näkymiä, ja se voi olla usean eri sovelluksen käytössä. Tässä ohjelmassa mallina toimivat PostgreSQL-tietokannan taulut,

jotka sisältävät XML-relaatioita. Kaikki ohjelmassa käsiteltävä tieto on siis aina lähtöisin käytettävästä relaatiotietokannasta.

Näkymän tarkoituksena on visualisoida mallin sisältämät tiedot käyttäjälle intuitiivisella tavalla. Kehitetyssä käyttöliittymässä visualisoidaan kerrallaan aina yksi XML-relaatio tai sen osia. Visualisoinnin nimenä käytetään sen XML-relaation nimeä, josta visualisointi on muodostettu.

Ohjaimen tehtävänä on yhdistää malli ja näkymä. Sen toiminta perustuu hiirellä suoritettujen toimintojen seuraamiseen. Käyttäjä voi siten hiirellä osoittamalla ja/tai klikkaamalla sekä painamalla jotakin visuaalista oliota käynnistää siihen liittyvän aliohjelman tai muun toiminnallisuuden suorittamisen.

Ohjelman käyttämä PostgreSQL-tietokantasovellus on varustettu lisäominaisuuksilla XML-relaatioiden käsittelyä varten. Näistä lisäominaisuuksista visualisointiohjelma käyttää funktioita *relxml_parse(), relxml_toxml(), relxml_im_predecessor(), relxml_predecessors(), relxml_successors()* ja *relxml_im_successors()*. Funktio r*elxml_parse()* muodostaa XML-dokumentista XML-relaatioesityksen; *relxml_toxml()* muodostaa XML-relaatiosta tekstuaalisen XML-dokumentin; *relxml_im_predecessor()* palauttaa annetun indeksiarvon välittömän vanhemman indeksiarvon; *relxml_predecessors()* palauttaa annetun indeksiarvon kaikkien vanhempien indeksiarvot; *relxml_successors()* palauttaa annetun indeksiarvon kaikkien jälkeläisten indeksiarvot ja *relxml_im_successors()* palauttaa annetun indeksiarvon välittömien jälkeläisten indeksiarvot. Nämä lisäominaisuudet on implementoitu Niemen ja Järvelinin [2006] esittelemien formalismien mukaan, eikä niiden toteutus ole kuulunut tätä tutkielmaa varten luodun ohjelman piiriin.

4.3.2 Ikkunaoliot

Ohjelmassa on useita näkymiä, mutta tässä tutkielmassa käsitellään niistä ainoastaan kahta keskeisintä: visualisoitua rakennepuuta ja tietopilveä. Nämä molemmat ovat ikkunaolioita eli olioita, jotka näkyvät ruudulla. Rakennepuun tuottamiseen on kehitetty ikkunaolio nimeltään XMLWidget, kun taas tietopilvi tuotetaan käyttäen ikkunaoliota nimeltä DatacloudWidget. Kyseiset ikkunaoliot on konstruoitu useasta muusta (ali)ikkunaoliosta, joista jotkut ovat The Visual Library 2.0 -kirjastosta vakiona löytyviä ikkunaolioita, kun taas toiset ovat kyseisiä näkymiä varten kehitettyjä ikkunaolioita.

Visualisoidun rakennepuun tietoalkiot ovat tyyppiä XMLWidget, joka on muodostettu laajentamalla The Visual Libraryn Widget-ikkunaoliosta, joka toimii yliluokkana kaikille kyseisen kirjaston ikkunaolioille. XMLWidgetin jäsenmuuttujiin talletetaan siihen liittyvien elementtien tai attribuuttien ilmentymiä XML-relaatiossa vastaavat indeksiarvot. Instantoidun XMLWidgetin nimeksi tulee ko. elementin/attribuutin nimi. Jäsenmuuttujiin talletetut indeksiarvot ilmaisevat visualisointiohjelmalle täsmällisesti, missä kyseisen tietoalkion ilmentymät esiintyvät visualisoinnin perustana olevassa XML-relaatiossa. Jäsenmuuttujiin tallennetaan tieto siitä, onko kyseessä lehti-, sisä- vai juurisolmu ja onko kyseinen tietoalkio elementti vai attribuutti.

Visualisoitu tietopilvi on tyyppiä DatacloudWidget oleva ikkunaolio. Se sisältää tiedot visualisoitavista sanoista. Sanat ovat DataWidget-ikkunaolioita, joiden jäsenmuuttujiin on tallennettu tieto niistä XML-relaatioesityksen indekseistä, jotka liittyvät sellaisiin kaavio- ja ilmentymätason komponentteihin, joissa kyseinen sana esiintyy. Tämän lisäksi DataWidget sisältää tiedon siitä, mikä on kyseisen sanan fontin koko ja väri.

5 HETEROGEENISTEN XML-TIETOLÄHTEIDEN KÄSITTELY

Heterogeenisten XML-tietolähteiden käsittelyllä tarkoitetaan tässä niiden käsittelyä graafisen käyttöliittymän kautta siten, että niissä esiintyvät heterogeenisyystekijät (eli tietokonfliktien aiheuttajat) poistetaan. Tätä tutkielmaa varten toteutetun visualisointiohjelman rakennepuusta (luku 4) voidaan tuottaa taulumuotoinen esitys siinä olevien elementtien ja attribuuttien ilmentymistä rakennepuuta vastaavassa XML-relaatiossa (alakohta 5.2). Tämän jälkeen kyseisiä ilmentymiä voidaan käsitellä tavoilla, jotka visuaalinen vuorovaikutusmekanismi mahdollistaa. Tutkielmaa varten on myös luotu graafinen käyttöliittymä RXQL-kyselykielelle (alakohta 5.4), jolla voidaan ottaa huomioon myös ne tietokonfliktit, joita ei kyetä poistamaan tietolähteiden suoralla muokkauksella. Dokumentteja muokataan siis joko paikallisesti tai niistä luodaan erillisiä tulosdokumentteja.

5.1 Tietokonfliktit

XML-tietolähteiden välillä saattaa esiintyä tietokonflikteja. Tietokonfliktit voivat esiintyä kaavio- ja/tai ilmentymätasoilla. Johdannossa esiteltiin viisi erilaista tietokonfliktia. Näistä tietokonfliktityypeistä 1 (arvojen välinen konflikti) ja 5 (elementtien/attribuuttien keskinäinen konflikti) kyetään poistamaan yksinkertaisilla uudelleenesittämis- tai nimeämisoperaatioilla. Dokumenttien välinen tietokonflikti (tyyppi 3) voidaan poistaa näillä operaatioilla, mikäli heterogeenisyys johtuu edellämainituista tyypin 1 tai tyypin 5 tietokonflikteista. Tietoalkioiden ja niiden arvojen uudelleennimeämisellä ei kuitenkaan kyetä siirtämään tietoa kaaviotasolta ilmentymätasolle tai päinvastoin. Tämä vaatii erillisten harmonisointioperaatioiden käyttämistä. Näitä operaatioita tarkastellaan tarkemmin alakohdissa 5.3 ja 5.4. Kuvissa 19–21 esitetyissä XML-dokumenteissa esitetään sama informaatio eri tavoilla. Niiden välillä esiintyy kaikkia viittä tietokonfliktin tyyppiä. Yksittäisten esimerkkidokumenttien sisällä ei esiinny heterogeenisyystekijöitä.

```
1. <Joukkue>
2. <Pelaaja pelinumero="10">
3. Virtanen P
4. </Pelaaja>
5. <Pelaaja pelinumero="12">
6. Koskinen S
7. </Pelaaja>
8. </Joukkue>
```

Kuva 19. Joukkue1.xml

```
1. <Joukkue>
2. <Pelaaja>
3. <pelinumero>
4. 10
5. </pelinumero>
6. Virtanen
7. </Pelaaja>
8. <Pelaaja>
9. <pelinumero>
10. 12
11. </pelinumero>
12. Koskinen
13. </Pelaaja>
14. </Joukkue>
```

Kuva 20. Joukkue2.xml

```
1. <Joukkue>
2. <Peluri>
3. <_10>
4. Virtanen P
5. </_10>
6. </Peluri>
7. <Peluri>
8. <_12>
9. Koskinen S
10. </_12>
11. </Peluri>
12. </Joukkue>
```

Kuva 21. Joukkue3.xml

5.1.1 Arvojen välinen konflikti

Arvojen välinen konflikti (tyyppi 1) esiintyy, kun sama tieto esitetään eri tavalla, jolloin on kyse synonymiasta. Se edustaa syntaktista heterogeenisyyttä, ja se voidaan poistaa uudelleenkirjoittamisen avulla ts. esittämällä samaa tarkoittavat tiedot yhdellä yhdenmukaisella tavalla. Tällainen konflikti voi esiintyä esimerkiksi henkilöiden esittämisen yhteydessä, kuten kuvissa 19 (Joukkue1.xml) ja 20 (Joukkue2.xml) esitetyissä XML-dokumenteissa on tapahtunut. Käyttäjä voi havaita nämä konfliktit visuaalisten primitiivien avulla, jotka mahdollistavat tietoalkioiden arvojen tutkimisen. Kuvassa 22 on kuvan 19 dokumentissa esiintyvän *Pelaaja*-tietoalkion ilmentymätason sisältö, kun taas kuvassa 23 on kuvan 20 dokumentin *Pelaaja*-tietoalkion ilmentymätason sisältö. Käyttäjä saa nämä tiedot esiin valitsemalla ponnahdusvalikosta komennon **Values** em. tietoalkioiden kohdalla. Tarkastelemalla näitä visualisointeja käyttäjä huomaa, että kuvan 19 dokumentissa pelaajien nimet esitetään muodossa sukunimi, jota seuraa etunimen ensimmäinen kirjain. Sen sijaan kuvan 20 dokumentissa pelaajista ilmaistaan vain heidän sukunimensä. Oletetaan, että käyttäjä tietää samojen pelaajien esiintyvän molemmissa dokumenteissa. Kyseinen konflikti poistuu, kun kuvan 19 dokumentin pelaajien tiedot muutetaan yksitellen kuvan 20 dokumentin muotoon. Toisin sanoen arvo ”Virtanen P” muutetaan muotoon ”Virtanen” ja arvo ”Koskinen S” muotoon ”Koskinen”. Visuaalinen primitiivi uudelleenkirjoittamiseen esitellään kohdassa 5.2.

Tyypillisesti arvojen välinen konflikti esiintyy myös sellaisten dokumenttien välillä, joista löytyy numeerisia arvoja kuten hintoja ja mittoja ja jotka perustuvat eri mittayksiköihin. Hinnat voivat olla ilmaistuna eri valuuttayksiköillä (esim. dollari ja euro) ja mitta-arvot eri järjestelmillä (esim. tuuma-pauna-järjestelmä ja metrijärjestelmä). Tällaisten konfliktien poistaminen saattaa olla ongelmallista, ellei alkuperäistä yksikköä ole ilmaistu riittävän selkeästi käsittelyn kohteena olevissa arvoissa. Tällöin on tarpeellista tutkia tietolähdettä ja löytää sen arvojen mittayksiköille oikea tulkinta. Esimerkiksi jos dokumentin tiedot koskevat selvästi tiettyä maata, esimerkiksi Englantia, niin hintoja ilmaisevat arvot ovat todennäköisesti puntia. Numeeristen arvojen muuttaminen järjestelmästä toiseen on mahdollista automatisoida. Tutkielman pohjalla olevassa järjestelmässä on toteutettu visuaalinen primitiivi, jonka avulla valuuttakonversiot yleisimpien valuuttayksiköiden välillä voidaan suorittaa.

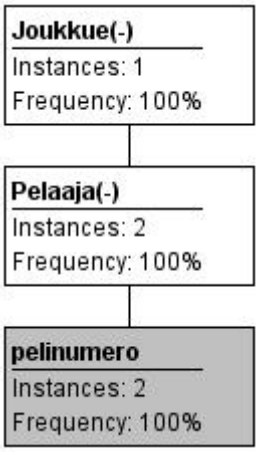
Kuva 22. Kuvan 19 Pelaaja-tietoalkion ilmentymätason sisältö

Kuva 23. Kuvan 20 Pelaaja-tietoalkion ilmentymätason sisältö

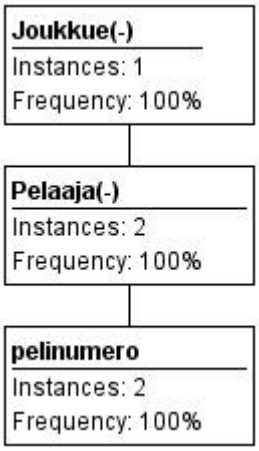
5.1.2 Attribuutin ja elementin välinen konflikti

Attribuutin ja elementin välinen konfliktityyppi (tyyppi 2) esiintyy, kun sama tieto esitetään dokumenteissa sekä elementtien että attribuuttien avulla. Tämä konflikti syntyy, kun kaksi tai useampia dokumentteja on laadittu samaan tarkoitukseen toisistaan riippumatta. Kyseinen konfliktityyppi esiintyy kuvissa 19 (Joukkue1.xml) ja 20 (Joukkue2.xml) esitettyjen dokumenttien välillä. Kuvan 19 XML-dokumentissa esiintyy attribuutti nimeltä *pelinumero*, kun taas kuvassa 20 samaa tarkoittava tieto esitetään alirakenteettomana elementtinä. Nämä konfliktit voidaan havaita rakennevisualisoinnin avulla. Kuvassa 24 on kuvan 19 XML-dokumentin täydellinen rakennevisualisointi ja kuvassa 25 on vastaava visualisointi perustuen kuvan 20 XML-dokumenttiin. Visualisoinneista on helppo huomata *pelinumero*-tietoalkioiden välinen tietokonflikti: tummennettu suorakulmio kuvassa 24 ilmaisee, että siinä esitetty *pelinumero* on attribuutti, kun taas kuvassa 25 *pelinumero* esitetään valkoisella pohjalla, mikä ilmaisee kyseisen tietoalkion olevan elementti.

Jos elementtien ilmentymiä edustava tietoalkio on lehtisolmu, mahdollistaa XML-relaatioesitystapa kyseisen tietokonfliktin poistamisen muuttamalla ko. elementtien ilmentymissä niiden tyypit *a:ksi* tai vastaavasti attribuuttien ilmentymissä niiden tyypit *e*:ksi. Tämän jälkeen entiselle attribuutille olisi mahdollista luoda alirakenne ja attribuutiksi muutetulla entisellä elementillä ei enää olisi tätä mahdollisuutta. Toisin sanoen attribuuttina esitetyn tietoalkion ominaisuuksia kasvatettaisiin ja elementtinä esitetyn tietoalkion ominaisuuksia rajoitettaisiin. Tällaisen operaation suorittaminen muuttaisi siten dokumentin uudelleenstrukturointimahdollisuuksia.



Kuva24. Joukkue1.xml



Kuva 25. Joukkue2.xml

5.1.3 Dokumenttien välinen konflikti

Dokumenttien välinen konflikti tarkoittaa tilannetta, jossa sama tieto esitetään kahdessa tai useammassa dokumentissa erilaisia rakenteita käyttäen. Kaikkien kuvissa 19–21 olevien dokumenttien välillä esiintyy tämänkaltainen tietokonflikti. Niemi *et al.* [2009] huomauttavat, että dokumenttien välinen konflikti voi esiintyä, vaikka sama tieto olisi esitetty käyttäen samoja rakenteita. Tämä on mahdollista tilanteessa, jossa on esitetty dokumentit A, B ja C siten, että dokumentti A:lla on sama tietosisältö kuin dokumenteilla B ja C yhteensä. Esimerkiksi jos dokumentti A sisältää jalkapallojoukkueen pelaajien nimi- ja pelinumerotiedot, dokumentti B sisältää pelaajien nimitiedot (mutta ei pelinumeroita) samoilla rakenteilla esitettynä kuin dokumentissa A, ja dokumentti C sisältää pelaajien pelinumerotiedot (mutta ei nimitietoja) käyttäen myös samoja rakenteita A:n kanssa. Toisin sanoen kaikkien dokumenttien A, B, C välillä esiintyy tietokonflikti, koska ei voida löytää kahta dokumenttia, joilla olisi sama tietosisältö. Tämä tietokonflikti voidaan havaita tutkimalla A:n, B:n ja C:n rakennepuuvisualisointeja.

5.1.4 Arvojen ja tietoalkioiden nimien välinen konflikti

Arvojen ja tietoalkioiden nimien välinen konflikti esiintyy, mikäli jonkin dokumentin tietoalkion (attribuutti tai elementti) arvot esitetään tietoalkion niminä jossain toisessa dokumentissa. Tällainen konflikti esiintyy dokumenttien 20 (Joukkue2.xml) ja 21 (Joukkue3.xml) välillä. Kuvan 21 dokumentissa esiintyvät elementit *_10* ja *_12* (alaviivat numeron edessä johtuvat XML:n elementtien nimeämiskäytännöistä). Kuvan 20

dokumentissa vastaava informaatio esitetään *pelinumero*-elementtien ilmentymien arvoissa. Numeroitujen elementtien käyttö on perusteltua, koska jalkapallojoukkueella saattaa olla pelinumeroihin pohjautuva taktiikka. Tällaisessa pelistrategiassa tiettyä paikkaa pelaa aina tietty numero ja numeroon liittyvä henkilö voi vaihdella. Tätä konfliktia ei kyetä poistamaan uudelleennimeämisoperaatioilla. Se voidaan kuitenkin havaita tutkimalla dokumenttien rakennepuita ja ilmentymätasoja käyttöliittymän edellä esitetyillä visualisointiprimitiiveillä. Tämän jälkeen havaittu tietokonflikti voidaan poistaa käyttäen järjestelmän visuaalista kyselypiirrettä (alakohta 5.4).

5.1.5 Tietoalkioiden nimien keskinäinen konflikti

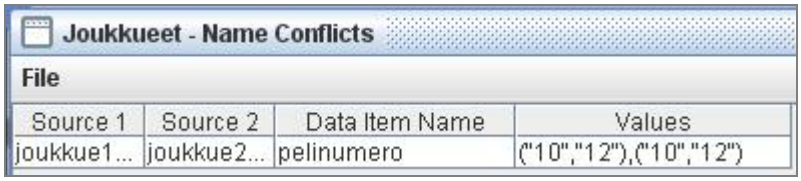
Tämä konfliktityyppi syntyy silloin, kun semanttisesti samaa tarkoittavat tietoalkiot nimetään eri tavoilla eri dokumentissa tai kun eri asioita tarkoittavat tietoalkiot nimetään samoiksi. Tietoalkioiden nimien keskinäinen konflikti voidaan poistaa uudelleennimeämisen avulla. Kyseinen konfliktityyppi esiintyy kuvan 21 XML-dokumentin ja kuvissa 19 ja 20 esitettyjen XML-dokumenttien välillä. Kuvan 21 XML-dokumentissa pelaajien tiedot esitetään *Peluri*-elementteinä, kun taas muissa dokumenteissa samat tiedot kuvataan *Pelaaja*-elementeillä. Tiedetään kuitenkin, että kyseessä ovat semanttisesti samaa tarkoittavat tiedot. Dokumenttien väliset tietokonfliktit poistuvat, kun *Peluri*-elementin ilmentymät nimetään *Pelaaja*-elementeiksi. Tämä operaatio voidaan automatisoida koskemaan koko dokumenttia (alakohta 5.2).

5.2 Tietokonfliktien havaitseminen ja visuaalinen muokkaus

Jotta tietokonfliktit voidaan ottaa huomioon ja käsitellä, pitää ne ensin havaita. Tietokonfliktien havaitsemisessa käyttäjää voidaan avustaa. Tätä varten käyttöliittymä sisältää primitiivejä, joiden avulla kokoelmasta voidaan hakea potentiaalisesti yhteensopivia dokumentteja (kokoelman dokumentit, joiden välillä ei esiinny dokumenttikonfliktia) sekä potentiaalisia taso- (attribuutin/elementin nimien ja arvojen välinen) ja nimikonflikteja (attribuutin/elementin nimien välinen).

Kuvassa 26 on käyttöliittymällä tuotettu taulumuotoinen esitys potentiaalisista nimikonflikteista. Konfliktitaulu tuotetaan valitsemalla haluttu kokoelma ja käynnistämällä nimikonfliktien etsintä siihen tarkoitetulla **Name Conflicts** -panikkeella. Kokoelma, johon

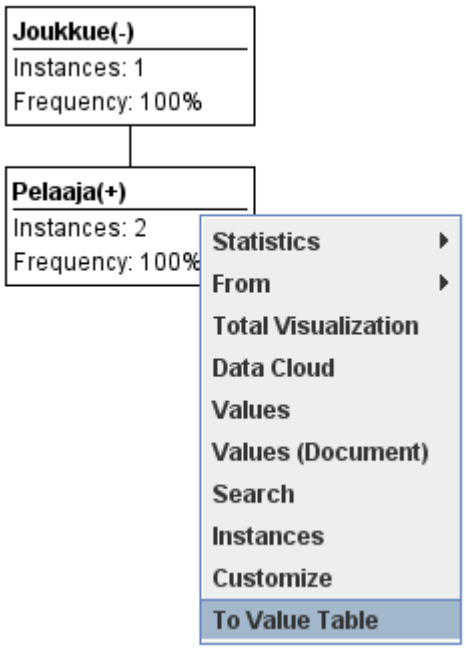
haku kohdistuu, on muodostettu kuvien 19–21 XML-dokumenteista. Tuloksena on löytynyt tietoalkion nimi *pelinumero* dokumenteista Joukkue1.xml sekä Joukkue2.xml. Lähemmin tarkasteltuna (esimerkiksi rakennepuun avulla) huomataan, että Joukkue1.xml:ssä *pelinumero* on attribuutti ja Joukkue2.xml:ssä se on elementti. Kyseessä on siis attribuutin ja elementin välinen tyypin 2 konflikti.



Kuva 26. Nimikonfliktit

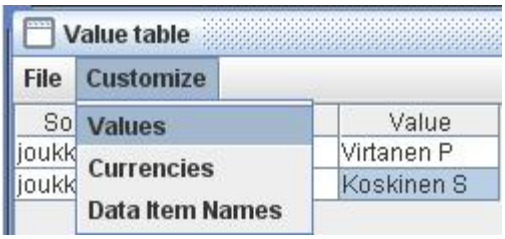
Uudelleennimeämistä edellyttäviä tietokonflikteja on mahdollista käsitellä useampi samalla kertaa. Tämä edellyttää kohteena olevien XML-tietolähteiden käymistä läpi ja haluttujen tietoalkioiden merkkaamista niistä. Arvojen kerääminen tapahtuu visuaalisessa käyttöliittymässä avaamalla (hiiren toissijainen painike) halutun tietoalkion kohdalta ponnahdusvalikko, josta valitaan komento **To Value Table**. Vaihtoehtoisesti käyttäjä voi valita komennon **Customize**, jonka kautta on mahdollista muokata yhtä tietoalkiota välittömästi. Kuvassa 27 kyseinen komento käynnistetään kuvassa 19 esitetyn XML-dokumentin *Pelaaja*-elementin visuaalisen primitiivin (suorakulmio) kohdalta. Kun käyttäjä on valinnut muokattavat tietoalkiot, suoritetaan käyttöliittymän komento **Create Value Table**, joka luo niistä taulumuotoisen esityksen (kuva 28).

Taulumuotoisesta esityksestä voidaan **Customize**-valikon kautta valita muokkausvaihtoehto koskemaan arvoja (**Values**), tietoalkioiden nimiä (**Data Items**) tai valuuttoja (**Currencies**) (kuva 29). Tässä tapauksessa käyttäjä on valinnut muokattavaksi Joukkue1.xml:ssä esiintyvän *Pelaaja*-elementin ilmentymän arvon ”Koskinen S”, jonka hän muuttaa muotoon ”Koskinen” (kuva 30) poistaen siten kohdassa 5.1.1 esitellyn arvojen välisen konfliktin. Käyttäjä voi valita myös useampia ilmentymiä muokattavaksi esimerkin avulla. Esimerkiksi kohdassa 5.1.5 esitelty tietoalkioiden nimien keskinäinen konflikti voidaan poistaa valitsemalla muokattavaksi kaikki *Peluri*-elementin ilmentymät, määrittelemällä uudelleenkirjoitus koskemaan tietoalkioiden nimiä ja antamalla esimerkkiarvoksi ”Pelaaja”.

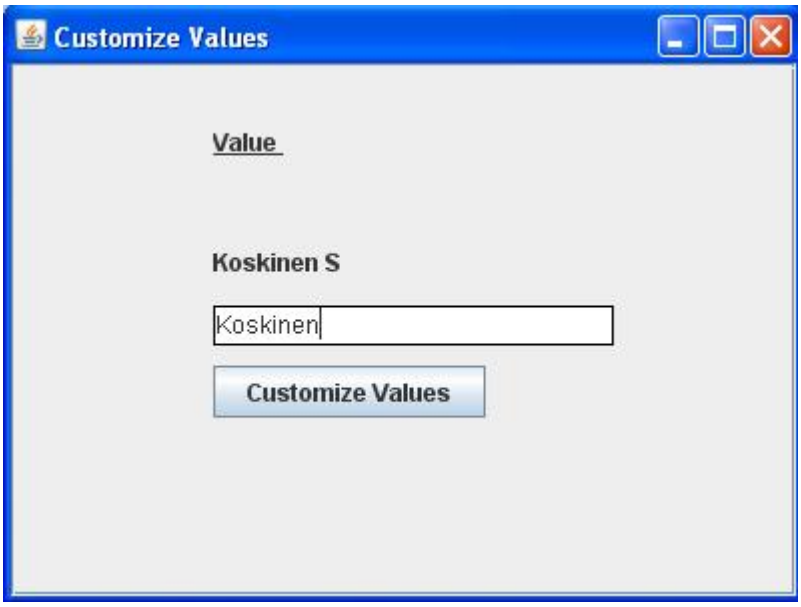


Kuva 27. Arvojen kerääminen

Kuva 28. Taulumuotoinen esitys ilmentymistä



Kuva 29. Muokkausvaihtoehdon valinta



Kuva 30. Arvojen uudelleenkirjoitus

5.3 XML-tietolähteiden harmonisointi kyselykielillä

Kyselykielillä suoritettu XML-tietolähteiden harmonisointi tarkoittaa tietolähteiden välillä esiintyvien heterogeenisten tekijöiden poistamista kyselyn lopputuloksena syntyvästä dokumentista. Nämä heterogeeniset tekijät voidaan poistaa joko asteittain tai kaikki yhdellä kertaa. Tietolähteiden muokkauksen lisäksi (alakohta 5.2 edellä) harmonisointi voi kohdistua myös tietokonfliktityyppien 2 (attribuuttien/elementtien välinen konflikti) ja 4 (attribuuttien/elementtien ja arvojen välinen konflikti) poistamiseen. Nämä harmonisointitilanteet muistuttavat tiedon kääntämistä siinä, että niissäkään ei muuteta alkuperäisen tietolähteen sisältöä vaan tuotetaan täysin uusi tulosdokumentti. Toisin kuin tiedon kääntämisessä harmonisoinnilta edellytetään kykyä siirtää tietoa kaaviotasolta ilmentymätasolle ja toisinpäin. Tarve tiedon siirtämiselle tasojen välillä syntyy esimerkiksi tilanteessa, jossa jossakin XML-dokumentissa oleva tieto on esitetty elementin nimenä ja toisessa dokumentissa samaa tarkoittava tieto on esitetty elementin arvona. Käyttäjä haluaa kuitenkin koota nämä tiedot molemmista lähteistä uuteen dokumenttiin, jossa ne esitetään yhdenmukaisessa muodossa. Heterogeenisten XML-tietolähteiden harmonisointiprosessi on olennainen osa satunnaisten tietotarpeiden tyydyttämistä, koska se mahdollistaa tietojen yhteiskäytön ilman tietolähteiden välisiä kaaviokuvauksia. Tässä luvussa tarkastellaan erilaisten kysely- ja käsittelykielten käyttöä XML-tietolähteiden harmonisoimisessa ja arvioidaan, miten hankalaa/helppoa niiden käyttö on.

5.3.1 Polkuorientoituneet XML-kielet

XML-tietolähteiden harmonisointi on periaatteessa mahdollista toteuttaa käyttämällä XML-dokumenttien käsittelyyn tarkoitettuja tekstuaalisia kieliä kuten XQuery [W3C, 2010a] ja XSLT [W3C, 2007]. Ne ovat W3C:n XML-dokumenttien yleiseen käsittelyyn suosittelemia (kysely)kieliä. Molemmat niistä perustuvat XPathiin [W3C, 2011b], joka on niiden molempien aito osajoukko. Periaatteessa harmonisointi voidaan suorittaa myös millä tahansa muulla XML-muotoisten tietojen käsittelyyn tarkoitetulla kielellä, jolla kyetään siirtämään tietoa kaavio- ja ilmentymätasojen välillä ja muokkaamaan tietolähteissä ilmentymä- ja kaaviotasoilla olevia tietoja. Edellä mainittujen kyselykielten lisäksi harmonisointi voidaan suorittaa millä tahansa laskennallisesti täydellisellä ohjelmointikielellä, vaikka se ei tukisikaan XML-dokumenttien käsittelyä. Laskennallisesti täydellisellä ohjelmointikielellä tarkoitetaan kieltä, jolla kyetään simuloimaan mitä tahansa yksinauhaista Turingin konetta [Turing, 1937].

XDM (XQuery 1.0 and XPath 2.0 Data Model) [W3C, 2010b] on XPathin, XQueryn ja XSLT:n yhteinen tietomalli. Sen perustuu XPath 1.0:n tietomalliin, jonka pohjana on abstrakti tietojoukko XML Information Set [W3C, 2004b], joka sisältää johdonmukaiset määritelmät konkreettisten määritelmien konstruoimiselle XML-tietoihin viittamiseen. XDM esittää alkioista koostuvan puurakenteen. Alkiot puolestaan ovat joko atomisia arvoja tai solmuja. Atomiset arvot ovat XML Scheman [W3C, 2004a] tietotyyppien mukaisia arvoja. Solmuja on seitsemää erilaista tyyppiä: dokumentti/juuri-, attribuutti-, elementti-, nimiavaruus-, prosessointiohje-, kommentti- ja tekstisolmuja. Tämän lisäksi solmut voivat olla sekvenssejä, jotka sisältävät edellä kuvattuja alkioita tietyssä järjestyksessä.

5.3.2 XPath

XPath (XML Path Language Version 2.0) on polkuilmaisuihin perustuva kyselykieli XML-tietolähteiden käsittelemiseen. Polku on reitti XML-dokumentin juuresta haluttuun solmuun. XPath operoi siis XML-tietolähteissä puumallin mukaisesti ja palauttaa niistä XDM:n mukaisia alkioita. Polkuilmaisuja tehdessään käyttäjän pitää tietää solmun nimi ja muuttujien alustaminen polkuun liittyvän mallin sovituksen kautta.

Haluttu polku voidaan esittää absoluuttisena tai suhteellisena. Absoluuttinen polku on suhteellinen juurielementtiin nähden. Sen konteksti on siis koko dokumentti. Absoluuttinen

polku alkaa aina etukenoviivalla (/) tai kaksoisetukenoviivalla (//). Etukenoviivalla ilmaistu polku ilmaisee sen solmun, jonka välittömiä jälkeläisiä käsitellään, kun taas kaksoisetukenoviiva ilmaisee, että käsittelyn kohteena ovat kaikki ko. solmun jälkeläiset. Suhteellisissa polkuilmauksissa kyselyt evaluoidaan siinä kontekstissa, joka on kulloinkin valittuna, eikä siinä ekspliittisesti ilmaista polkua. Käsiteltäessä kuvan 20 dokumenttia (Joukkue2.xml) pelkkä solmun nimi *pelinumero* on suhteellinen polkuilmaus ja se valitsee kaikki kyseisen nimisten solmujen esiintymät kyseisessä kontekstissa (tässä tapauksessa 2). Absoluuttisena polkuilmauksena sama voidaan esittää joko muodossa *//pelinumero* (kontekstina koko dokumentti) tai */Joukkue/Pelaaja/pelinumero* (konteksti ekspliittisesti määritelty). Polkuilmaisuja on mahdollista ketjuttaa.

Pistenotaatiolla (.) valitaan käsiteltävä solmu, kahden pisteen notaatiolla (..) valitaan käsittelyn kohteena olevan solmun vanhempi, ja ilmaisu *@attribuutin_nimi* valitsee attribuutin käsittelyn kohteena olevasta (elementti)solmusta. Näitä ilmaisuja voidaan tarkentaa predikaateilla. Näin voidaan rajata tulosjoukkoa valitsemalla tietyn solmun ilmentymien osajoukko. XML-tietolähteestä Joukkue2.xml voidaan valita ensimmäinen *pelinumero*-elementin ilmentymä ilmaisulla *//pelinumero[1]* ja pelinumeroltaan kymmentä suurempi oleva pelaaja ilmaisulla *//Pelaaja[pelinumero > 10]*, joka valitsee toisen *pelinumero*-elementin esiintymän dokumentista Joukkue2.xml**.** XPath sisältää lisäksi yli 100 valmista funktiota, joilla voidaan muun muassa vertailla ja ketjuttaa arvoja sekä käsitellä solmuja suoraan.

XPath 2.0:ssa voidaan soveltaa sekä heikkoa että vahvaa tyypitystä. Mikäli solmun sisältämälle arvolle on määritelty tyyppi ulkopuolisessa kaaviossa, sen käsittelyssä on käytettävissä vahva tyypitys. Vahva tyypitys ei salli operaatioiden suorittamista kahden erityyppisen muuttujan välillä. Täten yritys suorittaa esimerkiksi yhteenlaskuoperaatio päivämäärän ja kokonaisluvun välillä tuottaa järjestelmävirheen. Heikossa tyypityksessä pyritään konvertoimaan operaatioissa käytettävät muuttujat yhteensopiviksi. Esimerkiksi edellä mainittu yhteenlaskuoperaatio konvertoisi sekä päivämäärän että kokonaisluvun merkkijonoiksi, jonka jälkeen se yhdistäisi ne esimerkiksi ketjuttamalla.

5.3.3 XQuery

XQuery (XML Query Language) on XML-tietojen käsittelyyn suunniteltu kieli. Se käyttää XPathin polkuilmaisuja liittämään käsittely koskemaan tiettyä osaa XML-dokumentista.

XPathin polkuilmaisut toimivat siten XQueryssa navigointimekanismina. Se soveltuu hyvin suurten tietolähteiden käsittelyyn, koska XSLT:sta (alakohta 5.3.4) poiketen käsittelyn kohteena ei oletuksena ole koko tietolähde.

XQuery on kehitetty usean (kysely)kielen kuten Quilt [Chamberlin *et al.,* 2001], XQL [Robie *et al.,* 1998] ja XML-QL [Deutsch *et al.,* 1998] pohjalta. XQuery lainaa näistä kielistä soveltuvia osia. XQueryn vaatimuksissa sanotaan, että sillä voi olla useampi kuin yksi syntaksimuoto. Vaatimuksissa määritellään myös, että yhden syntaksin tulee olla helposti ihmisille helppo tulkittava ja yksi syntaksi täytyy kyetä esittämään XML:n muodossa. Kuvassa 31 esitetty kysely käyttää helposti tulkittavissa olevaa syntaksia. Tulkinnan helppous on kuitenkin varsin abstrakti ja subjektiivinen toteamus, eikä sille aseteta mitään selkeää mittaustapaa. XML-muotoista syntaksia varten käytetään erillistä XQueryX-syntaksia [W3C, 2010c].

XQuery on polkuorientoitunut kyselykieli. Tämä edellyttää sitä, että kyselyn laatijan täytyy tuntea XML-tietolähteen rakenne, sisältö ja semantiikka löytääkseen halutut tiedot. Toisin sanoen käyttäjän tulee selvittää aikaisemmin tuntemattoman XML-tietolähteen rakenne ja sisältö sekä muodostaa niiden kautta semanttinen tulkinta kyseiselle XML-tietolähteelle. XQuery on myös proseduraalinen kieli. Siinä siis määritellään miten lopputulos saadaan aikaiseksi. Tästä seuraa, että käyttäjän tulee hallita proseduraalisen ohjelmoinnin periaatteet kuten myös iteratiiviset kontrollirakenteet ja hävittävä muuttujakäsite (iteraation aikana muuttujan arvot vaihtuvat, jolloin vanha muuttujan arvo korvaantuu uudella arvotuksella). Kuvassa 31 on esitettynä XQuery-kysely, jolla tuotetaan dokumenteista Joukkue1.xml ja Joukkue3.xml tulosdokumentti (kuva 32), joka ei sisällä heterogeenisiä tekijöitä ts. tiedot esitetään tulosdokumentissa yhdellä yhdenmukaisella tavalla. Kyseisissä lähtödokumenteissa esiintyvät samojen pelaajien tiedot esitettyinä käyttäen erilaisia rakenteita. Esimerkkikyselyä voidaan kuitenkin soveltaa mihin tahansa vastaavat rakenteet sisältäviin XML-dokumentteihin ts. niissä voi esiintyä useampia vastaavalla tavalla organisoitujen tietojen ilmentymiä.

XQuery käyttää ns. FLWOR-ilmaisuja (*for, let, where, order by, return*), joilla kyetään iteroimaan XML-dokumenttia (*for*), arvottamaan muuttujia solmuilla (*let*), vertailemaan (*where*), palauttamaan (*return*) ja järjestämään (*order by*) arvoja ja alkioita. Nämä ilmaisut ovat osittain samantyylisiä kuin mitä relaatiotietokantojen käsittelyyn tarkoitettu SQL käyttää.

XQuery ei kuitenkaan tue dokumenttien paikallista päivitystä tai käsittelyä. W3C suosittelee tähän tarkoitukseen XQuery Update Facilitya [W3C, 2011c], joka mahdollistaa *insert* ja *update* -ilmaisujen käytön. Monet XML-tietokannat kuten BaseX ja eXist tukevat tätä laajennusta. XQueryssa on mahdollista toteuttaa ehdollinen haarautuminen soveltaen if-then-else-kontrollirakenteita. Tämän lisäksi on mahdollista kutsua käyttäjän XQuerylla määrittelemiä funktioita, jotka voidaan määritellä kyselytiedostossa tai ulkopuolisessa kirjastossa. Nämä ominaisuudet tekevät XQuerysta Turing-täydellisen kielen [Kepser, 2004].

```
1.  <Joukkue>
2.
3.  {
4.  for $n1 in doc("Joukkue1.xml")/Joukkue/Pelaaja
5.  return <pelaaja nimi="{$n1/text()}">
6.  <pelinumero> { $n1/@pelinumero/data() }
7.  </pelinumero>
8.  </pelaaja>
9.  }
10.
11. {
12. for $n4 in doc("Joukkue3.xml")/Joukkue/Peluri
13. return <pelaaja nimi="{$n3/text()}"><pelinumero> { $n3/node-name(node()) }
14. </pelinumero>
15. </pelaaja>
16. }
17.
18. </Joukkue>
```

Kuva 31. Query1.xd

Rivillä 1 (kuva 31) esiintyy juurielementin *Joukkue* alkutunniste ja rivillä 18 on sen lopputunniste. Riveillä 3–9 käsitellään dokumentin Joukkue1.xml tietoja. Kysely käy läpi jokaisen *Pelaaja*-elementin ilmentymän, jonka välittömänä vanhempana on *Joukkue*-elementin ilmentymä. Jokainen näistä ilmentymistä talletetaan yksi kerrallaan muuttujaan *$n1*. Jokaista talletettua ilmentymää kohdin palautetaan XML-osadokumentti [W3C, 2001], jonka juurielementtinä on *pelaaja* (alkaen riviltä 5). Se sisältää attribuutin *nimi,* joka saa arvokseen kyseisellä hetkellä muuttujassa *$n1* olevan *Pelaaja*-elementin ilmentymän sisällön tekstuaalisen arvon. Tämän jälkeen tulosdokumenttiin muodostetaan *pelinumero*-elementti, joka sisältää ko. elementin ilmentymän *pelinumero*-attribuutin arvon. Riveillä 11–16 käsitellään dokumentin Joukkue3.xml tietoja. Kyseiseen dokumenttiin sovelletaan yllä

mainitun kaltaisia operaatioita. Kysely käy läpi molemmat *Peluri*-elementin esiintymät, joiden välittöminä vanhempina on *Joukkue*-elementti. Joukkue3.xml:ssä pelinumero on esitetty elementin nimenä ja pelaajan nimi tämän elementin tekstuaalisena sisältönä.

```
1.  <Joukkue>
2.  <pelaaja nimi="Virtanen P">
3.  <pelinumero>10</pelinumero>
4.  </pelaaja>
5.  <pelaaja nimi="Koskinen S">
6.  <pelinumero>12</pelinumero>
7.  </pelaaja>
8.  <pelaaja nimi="Virtanen P">
9.  <pelinumero>_10</pelinumero>
10. </pelaaja>
11. <pelaaja nimi="Koskinen S">
12. <pelinumero>_12</pelinumero>
13. </pelaaja>
14. </Joukkue>
```

Kuva 32. Tulosdokumentti Query1.xd:stä

XQuery 3.0 [Robie *et al.*, 2011] on XQuery 1.0:n laajennus, jolla ei vielä ole W3C:n suositus-statusta. Se tukee *korkean kertaluvun funktioita* (engl. high-order functions). Niillä tarkoitetaan funktioita, jotka hyväksyvät parametreikseen toisia funktioita tai jotka kykenevät palauttamaan niitä. Tämän lisäksi XQuery 3.0 sisältää uusia FLOWR-tyylisiä ilmaisuja kuten ryhmittely- (*group by*) ja lukumääräilmaisut (*count*). Nämä ilmaisut tukevat etenkin tieto-orientoituneiden XML-dokumenttien käsittelyä.

5.3.4 XSLT

XSLT (XSL Transformations) on XML-tietojen käsittelyyn tarkoitettu kieli. Se suunniteltiin alunperin XML-dokumenttien esittämiseksi toisenlaisella tavalla. Se on osa XSL-perhettä. XSL (EXtensible Stylesheet Language) on vastaava tyylipohja XML:lle kuin CSS on HTML:lle. XSLT pohjautuu XML:ään ja on siten itsessään XML-sovellus. Tästä seuraa, että XSLT-sovelluksien on noudatettava XML:n syntaksia.

XSLT käyttää tietojen navigointiin eli solmujen käsittelyyn XQueryn tavoin XPathia. XSLT:lla esitetty harmonisointi tuottaa annetuista tietolähteistä uuden dokumentin käyttäen ennalta määriteltyä tyylipohjamoduulia. Tyylipohjamoduuli sisältää ohjeet siihen, miten alkuperäisen lähteen tiedot tulee esittää tulosdokumentissa. Tulosdokumentti voi erota

lähtödokumentista esitysformaatiltaan ja rakenteeltaan. XSLT soveltuu siis muihinkin kuin XML-dokumenttien välisiin muunnoksiin. Tyylipohja voidaan linkata suoraan XML-dokumenttiin. Tämän jälkeen ko. dokumentti voidaan esittää, esimerkiksi WWW-selaimissa, kuvauksen mukaisella tavalla. Linkkaaminen tapahtuu ilmaisulla

<?xml-stylesheet type="text/xsl" href="pohja.xsl"?>

Siinä pohja.xsl sisältää XSLT-ilmaisut tyylipohjan määrittelemiseksi. Tyylipohja voi prosessoida myös ulkoisia XML-lähteitä. Tätä varten on olemassa *document()*-funktio (kuva 33; rivit 6 ja 14).

```
1.  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
2.  <xsl:template match="/">
3.
4.  <Joukkue>
5.
6.  <xsl:for-each select="document('Joukkue1.xml')/Joukkue/Pelaaja">
7.  <pelaaja nimi="{.}">
8.  <pelinumero>
9.  <xsl:value-of select="@pelinumero" />
10. </pelinumero>
11. </pelaaja>
12. </xsl:for-each>
13.
14. <xsl:for-each select="document('Joukkue3.xml')/Joukkue/Peluri">
15. <pelaaja nimi="{.}">
16. <pelinumero>
17. <xsl:value-of select="name(./node())" />
18. </pelinumero>
19. </pelaaja>
20. </xsl:for-each>
21.
22. </Joukkue>
23.
24. </xsl:template>
25. </xsl:stylesheet>
```

Kuva 33. XSLT-Query.xml

XSLT-sovellus määritellään nimeämällä se juurielementissä (xsl:stylesheet) XSL-tyylipohjaksi ja valitsemalla tietty nimiavaruus (kuva 33; rivi 1). Tämän jälkeen XSLT:n

ilmaisuja annetaan kyseiseen XSL-nimiavaruuteen perustuen. XSL-tyylipohja koostuu säännöistä (template), jotka ilmaisevat, miten tietynlaisten solmujen esiintymiä pitää käsitellä. Säännöt rakennetaan xsl:template-elementillä, jonka match-attribuutti ilmaisee XML-lähteestä käsiteltävän osan. Kuvassa 33 rivillä 2 määritellään, että sääntöjen vaikutusalueena on koko dokumentti. Käsiteltävä lähde voidaan iteroida läpi xsl:for-each-ilmaisulla, ja sen select-attribuutissa oleva XPath-ilmaisu kiinnittää käsiteltävän solmun (kuva 33; rivi 6). Käsiteltävän solmun arvo poimitaan xsl:value-of-elementillä. Myös tähän elementtiin voidaan sijoittaa select-attribuutti (kuva 33; rivit 9 ja 17) ja käyttää XPathin predikaatteja tulosjoukon rajoittamiseen. Tulosjoukon järjestäminen on mahdollista XSLT:ssa käyttäen xsl:sort-elementtiä.

Kuvassa 33 on esitettynä XSLT-sovellus, joka tuottaa Joukkue1.xml:stä ja Joukkue3.xml:stä kuvassa 32 esitetyn tulosdokumentin. XSLT-sovellus on hyvin samankaltainen kuin XQuerylla toteutettu vastaava harmonisointi (kuva 31). Sekä XSLT:lla että XQuerylla voidaan toteuttaa samat asiat, ja ne molemmat ovat W3C:n suosituksia. XQuery on yleinen valinta tietokantatyyppisille käsittelytarpeille, kun taas XSLT soveltuu paremmin dokumenttien visuaaliseen esittämiseen [Kay, 2006]. Harmonisointi ei yksiselitteisesti ole kumpaakaan näistä. Testihenkilöille, joilla ei ollut kokemusta kummastakaan kielestä, XQueryn käytettävyys yksinkertaisissa tehtävissä havaittiin paremmaksi kuin XSLT:n [Graaumans, 2004]. Yleisimmät WWW-selaimet kuten Mozilla Firefox ja Internet Explorer tukevat XSLT:tä, mutta eivät XQuerya. Tämä tekee XSLT:stä luontaisen vaihtoehdon XML-tietojen WWW-julkaisemiseen.

5.3.5 RXQL

RXQL (Relational XML Query Language) [Näppilä *et al.*, 2011] on Tampereen yliopistossa kehitetty tekstuaalinen kyselykieli XML-tietojen käsittelyyn. RXQL perustuu luvussa 3 esiteltyyn XML-relaatioon. RXQL:n deklaratiivisuuden aste on hyvin korkea verrattuna polkuorientoituneisiin XML-kyselykieliin. Korkea deklaratiivisuuden aste tarkoittaa, että käyttäjä voi kuvata halutun lopputuloksen, eikä hänen tarvitse ilmaista sitä käyttäen esim. polkuilmaisuja ja iteratiivisia rakenteita. Tästä seuraa, että RXQL:n käyttäjän ei tarvitse hallita proseduraalisen ohjelmoinnin periaatteita, XPath-tyylisiä polkuilmaisuja tai XML-notaatiota, vaan käyttäjä voi ilmaista kyselyn lopputuloksen suoraviivaisesti ja selkeästi. RXQL on erityisesti heterogeenisten tieto-orientoituneiden XML-tietolähteiden käsittelyyn

kehitetty kieli, ja se tukee ilmauksiensa kautta tietojen harmonisointia, uudelleennimeämistä, uudelleenorganisointia ja erilaisia aggregointeja.

Hakusanoihin perustuvat kyselykielet eivät vaadi kohteena olevien XML-dokumenttien rakenteiden tuntemista. Niissä kysely palauttaa hakusanoihin liittyvät (merkitykselliset) tiedot. Tunnetuin poimintastrategia tällaisten tietojen poimimiseen on LCA (Lowest Common Ancestor) [Aho *et al.*, 1973], ja sen variantit kuten SLCA (Smallest LCA) ja MLCA (Meaningful LCA). LCA-poimintastrategia palauttaa solmut, jotka ovat annettujen hakusanojen esiintymien pienimpiä yhteisiä esivanhempia ts. palautetuilla solmuilla ei voi olla jälkeläistä, joka olisi annettujen avainsanojen esiintymien esivanhempi. XML-lähteiden epäsäännöllisyydestä johtuen tämä tapa ei ole aina tarkoituksenmukainen. RXQL käyttää tähän tarkoitukseen SPC-semantiikkaa (Smallest Possible Context) [Näppilä *et al.*, 2008] ja siten myös XML-relaatiota. SPC-semantiikka palauttaa aina pienimmän mahdollisen kontekstin, jossa annettujen avainsanojen esiintymät ovat. Siinä palautetaan indeksoidun XML-lähteen pohjalta solmujen sijasta niiden hakusanoja vastaavien tietoalkioiden indeksit, jotka esiintyvät pienimmässä mahdollisessa kontekstissa. Tämä indeksijoukko määrittelee pienimmän mahdollisen kontekstin.

RXQL-kysely koostuu kahdesta pakollisesta ja kolmesta vapaaehtoisesta osiosta. Pakolliset osiot ovat CONSTRUCT ja FROM. CONSTRUCT-osiossa kuvataan kyselyn lopputuloksen rakenne ja sisältö. Sulutuksella esitetään tietoalkioiden hierarkkinen suhde toisiinsa nähden. Attribuuteilla on etuliitteenään @-merkki. Etuliite + ilmaisee ne tietoalkiot, joilla voi olla useampia kuin yksi ilmentymä suhteessa tietoalkion vanhemman ilmentymään. Tietoalkioiden uudelleennimeäminen suoritetaan käyttäen AS-ilmaisua. INTO-ilmaisulla voidaan useampiin tietoalkioihin viitata yhdellä nimellä. Kyseisen nimen omaavan tietoalkion arvoihin viitataan Value()-ilmaisulla, missä argumenttina on ko. tietoalkion nimi. Tietoalkion nimi hakasuluissa valitsee arvoksi tietoalkion nimen, joten esimerkiksi ilmaisu *[tietoalkioXYZ]* palauttaa arvon ’tietoalkioXYZ’. FROM-osiossa määritellään kyselyssä käytettävät tietolähteet. Vapaaehtoiset osiot ovat WHERE, GROUP BY ja ORDER BY. WHERE-osiossa määritellään ehdot tiedon valinnalle ja aggregoinnille. RXQL:ssa on viisi aggregaattifunktiota: COUNT (lukumäärä), SUM (summa), AVG (keskiarvo), MIN (minimi) ja MAX (maksimi). GROUP BY ja ORDER BY -osioissa määritellään, minkälaisissa ryhmissä ja missä järjestyksessä tiedot tulosdokumentissa esitetään. Osiot liitetään yhteen

käyttämällä niissä samoja tietoalkioiden nimiä. Tämän takia jokaisessa käytetyssä osiossa pitää olla vähintään yksi FROM-osiosta löytyvä tietoalkion nimi, joka liittää kyseisen osion muihin osioihin.

Kuvassa 34 on esitettynä RXQL-kysely, joka muodostaa kuvassa 32 esitetyn tulosdokumentin XML-lähteistä Joukkue1.xml (kuva 19) ja Joukkue3.xml (kuva 21). Tämä RXQL:lla suoritettu harmonisointi on rivi- ja merkkimääräisesti selkeästi lyhyempi kuin XQuerylla ja XSLT:lla suoritetut vastaavat harmonisoinnit.

```
1. CONSTRUCT
2. Joukkue(+pelaaja(numero, @nimi))
3. FROM
4. Joukkue1.xml[Pelaaja AS @nimi, pelinumero AS numero],
5. Joukkue3.xml[_10,_12] INTO numero, VALUE(numero) AS @nimi]
```

Kuva 34. RXQL-kysely

5.4 Tietotarpeiden tyydyttäminen visuaalisella käyttöliittymällä

Tutkielmassa on toteutettu visuaalisen käyttöliittymän (myöhemmin käyttöliittymä) piirteenä mahdollisuus esittää visuaalisesti kyselyjä, joilla on RXQL:n ilmaisuvoima. Sen tarkoituksena on edelleen helpottaa käyttäjää tietotarpeiden tyydyttämisessä. Käyttöliittymä tarjoaa useita etuja XML:n käsittelyyn tarkoitettuihin polkuorientoituneisiin kieliin (XQuery, XSLT) verrattuna. Näissä kielissä käyttäjä joutuu määrittelemään tulosdokumentin proseduraalisesti, käyttäen polkuilmaisuja ja iteratiivisia rakenteita, kun taas käyttöliittymän kautta kysely voidaan määritellä visuaalisesti ts. käyttöliittymässä voidaan ilmaista tulosdokumentin rakenne ja sen tuottamiseen tarvittavat operaatiot. Käyttöliittymä tarjoaa visuaalisia primitiivejä, joilla voidaan määritellä aggregointeja ja suorittaa tietoalkioiden uudellennimeämisiä sekä siirtää tietoa kaavio- ja ilmentymätasojen välillä. Nämä primitiivit jakavat siten kyselyn helposti ymmärrettäviin osiin, eikä käyttäjän tarvitse hallita yllä mainittujen operaatioiden proseduraalista konstruointia.

Kyselyn tekemiseen tarkoitettu käyttöliittymä koostuu kuudesta nimetystä laatikosta (Construct, From, Aggregations, Where, Group ja Order), jotka käyttäjä täyttää tekstimuotoisella tiedolla. Näillä laatikoilla on ilmeiset analogiat RXQL:n osioihin pois lukien Aggregations-laatikko, joka sisältyy RXQL:ssa Where-osioon. Käyttöliittymän visuaalisten

primitiivien avulla käyttäjä kykenee hiirellä osoittamalla ja klikkaamalla generoimaan kyselyn osioita, eli edellä mainittuja laatikkoja. Näihin laatikoihin on mahdollista kirjoittaa kyselyn osia myös suoraan tekstuaalisessa muodossa. Molempia näistä tavoista voidaan käyttää samassa kyselyssä. Visuaalisen käyttöliittymän kautta on mahdollista suorittaa kaikki samat operaatiot kuin puhtaasti tekstuaalisella kyselyllä ts. niillä on sama ilmaisuvoima.

Esimerkkikyselyssä käytetään seuraavia kahta XML-tietolähdettä: Varasto.xml (kuva 35) ja Kauppa.xml (kuva 36). Näissä tietolähteissä on kuvattu kaikki tietyn yrityksen varastossa ja myymälässä olevat tuotteet. Käyttäjä haluaa tehdä inventaarion tuotteista, eli saada aikaiseksi XML-dokumentin, josta selviää yksiselitteisesti, kuinka monta kappaletta tiettyä tuotetta yrityksellä on. Tietolähteiden välillä esiintyy kuitenkin heterogeenisiä tekijöitä, eikä haluttua tietoa (tuotteiden kokonaiskappalemäärä) ole mahdollista saada selville tietolähteisiin kohdistuvalla suoralla kyselyllä. Toisin sanoen informaatiotarvetta tyydyttäessään käyttäjä joutuu määrittelemään erilaisia harmonisointioperaatioita ja aggregointeja.

```
1.  <varasto>
2.  <artikkeli>
3.  <nimi>
4.  Tietokone
5.  </nimi>
6.  <maara>
7.  423
8.  </maara>
9.  </artikkeli>
10. <artikkeli>
11. <nimi>
12. Pesukone
13. </nimi>
14. <maara>
15. 123
16. </maara>
17. </artikkeli>
18. </varasto>
```

Kuva 35. Varasto.xml

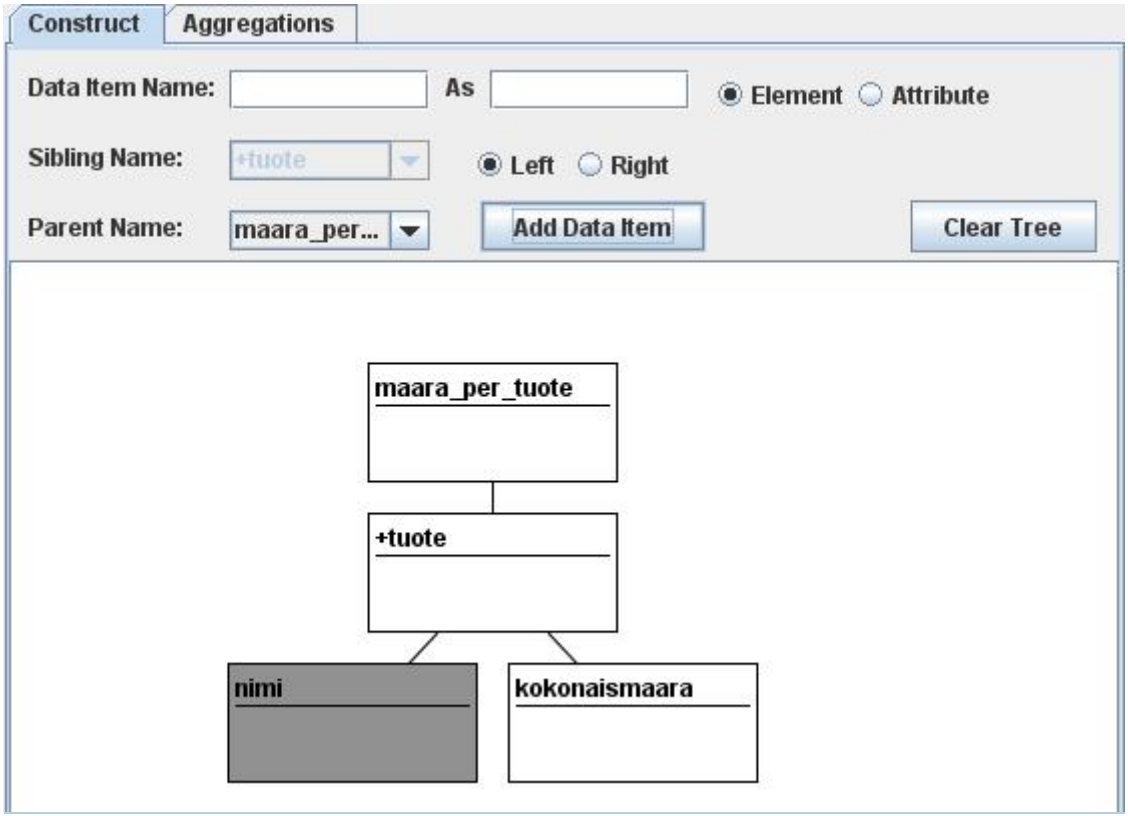
```
1.  <kauppa>
2.  <tuote tuotenimi="Tietokone">
3.  <kappaleet>
4.  101
5.  </kappaleet>
6.  </tuote>
7.  <tuote tuotenimi="Pesukone">
8.  <kappaleet>
9.  77
10. </kappaleet>
11. </tuote>
12. </kauppa>
```

Kuva 36. Kauppa.xml

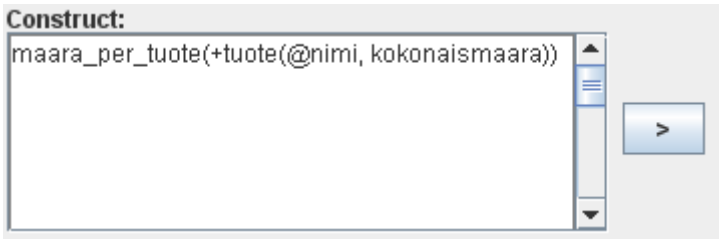
Visuaalisen kyselyn tekeminen aloitetaan määrittelemällä tulosdokumentin rakenne. Se tehdään kyselyikkunan Construct-välilehdessä (kuva 37). Koska tulosdokumentit ovat XML-muotoisia, niiden rakenne voidaan kuvata nimettyinä järjestettyinä puina. Käyttäjän tulee siis nimetä tulosdokumentissa esiintyvät tietoalkiot ts. elementtien ja attribuuttien nimet sekä määritellä niiden väliset hierarkkiset suhteet. Tietoalkion nimen ei tarvitse olla sama kuin sitä jossakin XML-tietolähteessä vastaavan elementin tai attribuutin nimi, vaan se voidaan uudelleennimetä käyttöliittymän kautta. Visualisoitavan tietoalkion hierarkiataso osoitetaan, juurielementtiä lukuunottamatta, klikkaamalla hiirellä tietoalkion vanhempaa. Mikäli kyseisellä tietoalkiolla voi olla useita ilmentymiä suhteessa sen vanhempaan, varustetaan sen nimi plus-merkillä. Tietoalkio visualisoidaan rakennepuuhun painikkeella **Add Data Item**. Tulosdokumentin rakennepuu noudattaa luvussa 4 esiteltyä maksimaalisuusperiaatetta ts. siinä ei visualisoida lainkaan ilmentymiä. Rakennepuun konstruoinnissa on mahdollista määrittää saman hierarkiatason tietoalkioille keskinäinen järjestys. Tämä tapahtuu radiopainikkeilla, joilla määritellään, tuleeko uusi tietoalkio ennen (Left) vai jälkeen (Right) aiemmin visualisoitua sisartietoalkiota.

Kuvassa 37 on esitettynä rakennepuu tietotarpeen tyydyttävälle tulosdokumentille. Se on visualisoitu käyttämällä 3 kertaa **Add Data Item** -painiketta. Tietoalkion harmaa väri ilmaisee, että ko. tietoalkio tullaan esittämään attribuuttina. Käyttöliittymä konstruoi tulosdokumentin rakennepuusta automaattisesti Construct-laatikon (kuva 38) sisällön eli tulosdokumentin tekstuaalisen kuvauksen RXQL:ssä. Käyttäjä voi myös syöttää tekstuaalisen kuvauksen suoraan laatikkoon ja visualisoida rakennepuun sen perusteella. Tämä tapahtuu

painamalla laatikon oikealla puolella olevaa nuolikuvaketta. Kyseinen rakennepuu ilmaisee, että tulosdokumentin juurielementin nimi on *maara_per_tuote*, jolla on mahdollista olla välittöminä jälkeläisinään useampia *tuote*-elementin ilmentymiä, joilla on jälkeläisinään elementtiesiintymä *kokonaismaara* sekä attribuuttiesiintymä *nimi*.



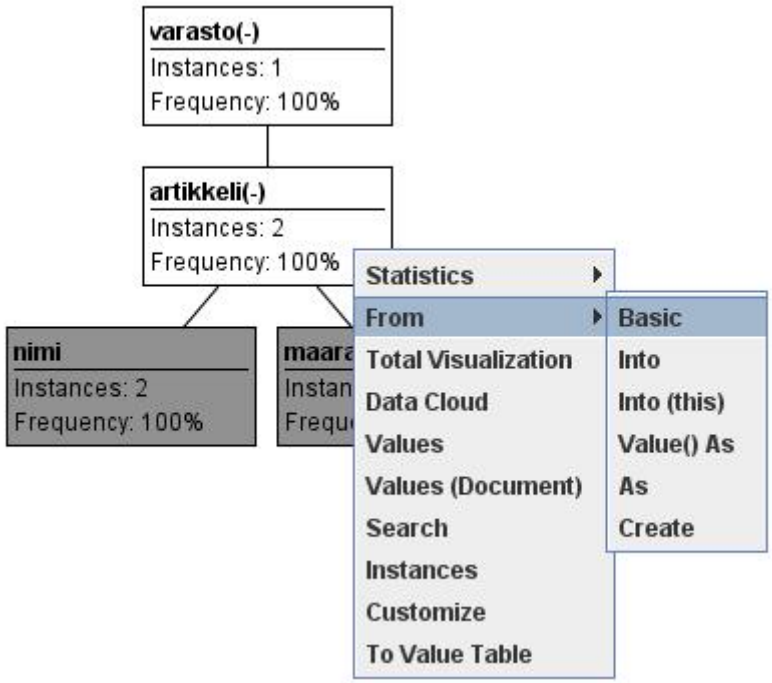
Kuva 37. Tulosdokumentin rakenteen määrittely



Kuva 38. Construct-laatikko, joka sisältää rakennepuun tekstimuodossa

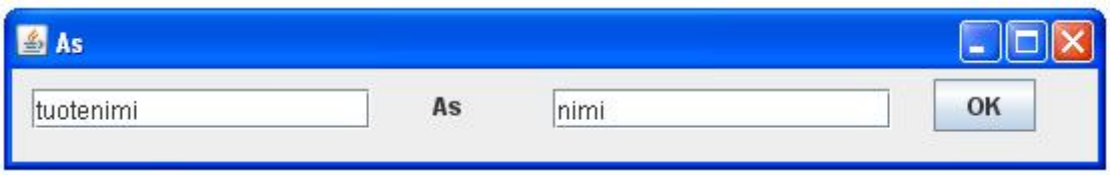
Tulosdokumentin rakenteen määrittelemisen jälkeen käyttäjä ilmaisee tietotarpeen tyydyttämisessä käytettävät tietolähteet ja ne tietoalkiot, joita kysely koskee. Tämä tapahtuu avaamalla halutun XML-dokumentin visualisointi ja käynnistämällä ponnahdusvalikko rakennepuun tietyn tietoalkion yhteydessä (kuva 39). Esimerkissä käyttäjä valitsee Varasto.xml:n visualisoinnista *artikkeli*-tietoalkioon liittyen **Basic**-komennon, joka puolestaan valitsee kaikki kyseisen tietoalkion jälkeläiset, jotka ovat rakennepuussa lehtisolmuina.

Kyselyssä aktivoiduksi tulevat siten tietoalkiot *nimi* ja *maara*. Tietolähteen Kauppa.xml-visualisoinnista käyttäjä valitsee tietoalkioiden *tuotenimi* ja *kappaleet* yhteydessä komennon **As**. Tällä komennolla asetetaan kysely koskemaan kyseisiä tietoalkioita siten, että kyselyssä ja tulosdokumentissa käytetään nimiä *maara* ja *nimi* vastaamaan lähtödokumentin nimiä *kappaleet* ja *tuotenimi* (ks. komennon **As** generoimaa dialogia kuvassa 40, jonka käyttäjä täyttää tietojen uudelleennimeämiseksi).

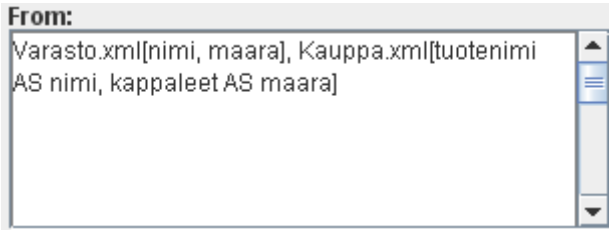


Kuva 39. Ponnahdusvalikko kyselyn tietolähteiden määrittelyyn

Muut mahdolliset komennot ovat **Into**, **Into (this)** ja **Value As()**. Näillä komennoilla voidaan sijoittaa tietoalkion tai sen jälkeläisten nimiä ja arvoja prosessointiaikaiseen tietoalkioon, jota kyselyssä käytetään. Tietolähteiden valinta ilmaistaan avaamalla From-valikko uudestaan ja valitsemalla sen generoimasta alivalikosta komento **Create**, joka puolestaan generoi From-laatikon (kuva 41).

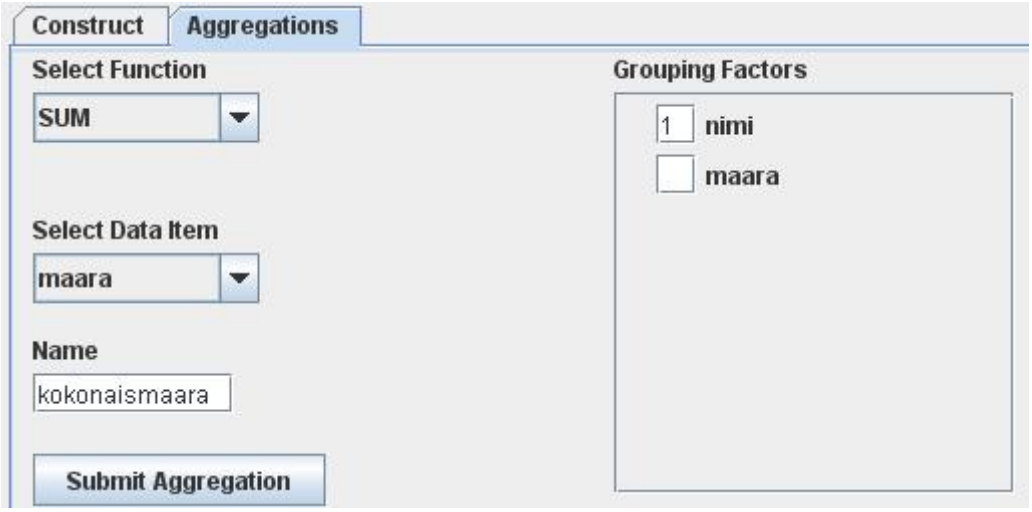


Kuva 40. Uudelleennimeäminen

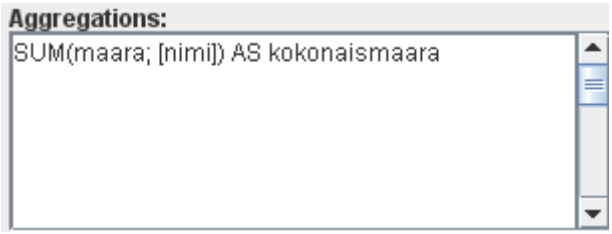


Kuva 41. Käyttöliittymän generoima From-laatikko, jonka käyttäjä täyttää

Inventaariolistauksen tuottaminen lähtödokumenteista edellyttää aggregointien ilmaisemista. Aggregoinnit on mahdollista määritellä niille käyttöliittymässä varatussa osiossa, joka saadaan esiin valitsemalla käyttöliittymästä Aggregations-välilehti (kuva 42). Ensimmäisenä käyttäjän tulee valita käytettävä aggregointifunktio, joka tässä tapauksessa on *SUM* eli yhteenlasku. Seuraavaksi käyttäjä valitsee sen tietoalkion nimen, johon liittyvien elementtien ja attribuuttien ilmentymien arvoja lasketaan yhteen (*maara*) ja antaa nimen aggregoinnin lopputulokselle (*kokonaismaara*). Tämän jälkeen määritellään numerojärjestyksessä ne tietoalkioiden nimet, joiden mukaan ryhmittely tapahtuu (tässä ainoastaan *nimi*). Aggregointi määritellään komennolla **Submit Aggregation**. Se myös konstruoi Aggregations-laatikon sisällöksi vastaavan RXQL-ilmauksen (esimerkkiin liittyen kuva 43). Tulosdokumentin rakenteen, tietolähteiden ja aggregointien määrittelyn jälkeen käyttäjä voi käynnistää koko kyselyn suorittamisen valitsemalla komennon **Submit Query**.



Kuva 42. Aggregointivalikko



Kuva 43. Käyttöliittymän generoima Aggregations-laatikko.

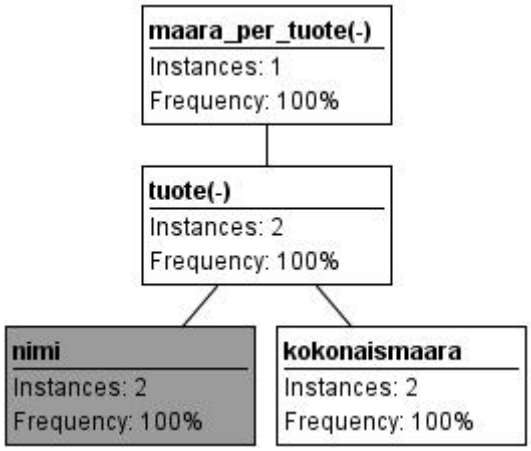
Kyselyn lopputuloksen (kuva 44) visualisointiin on kehitetty uusi esitystapa (kuva 45). Siinä visualisoidaan sekä kaavio- että ilmentymätason tiedot. Rakenteelliset elementit kuvataan laatikkoina, joissa niiden nimet esitetään sinisellä pohjalla. Alirakenteettomien elementtien sekä attribuuttien nimet sekä niiden ilmentymien arvot ovat näiden laatikkojen sisällä. Käyttäjän viedessä hiiren laatikossa sijaitsevan arvokentän päälle muuttuvat kyseinen kenttä ja sen kanssa samassa kontekstissa esiintyvät kentät keltaisiksi. Esimerkkikyselyn kohdalla käyttäjä saa siis tiedon kyseisen tuotetyypin kappalemäärästä (esimerkissä *Pesukone*; kuva 45) viemällä hiiren tuotetyypin nimen päälle.

Kuvassa 46 kyselyn lopputulos on esitetty visualisoidun rakennepuun muodossa. Kyselyn lopputulokselle on siis olemassa kolme erilaista esitysmuotoa: tekstuaalinen XML-dokumentti (kuva 44), arvovisualisointi (kuva 45) ja visuaalinen rakennepuu (kuva 46).

```
1.  <maara_per_tuote>
2.  <tuote nimi="Tietokone">
3.  <kokonaismaara>524.0000
4.  </kokonaismaara>
5.  </tuote>
6.  <tuote nimi="Pesukone">
7.  <kokonaismaara>
8.  200.0000
9.  </kokonaismaara>
10. </tuote>
11. </maara_per_tuote>
```

Kuva 44. Inventaario.xml

Kuva 45. Tulosdokumentin graafinen esitys (arvovisualisointi)



Kuva 46. Tulosdokumentin graafinen esitys (rakennevisualisointi)

6 POHDINTA

Toimivan tietoyhteiskunnan perusedellytys on kyky välittää tietoa eri toimijoiden välillä. XML on suosittu ja avoin esitysformaatti tähän tarkoitukseen, ja sitä käytetään yleisesti esimerkiksi B2B-sovellutuksissa [Cardoso and Bussler, 2011]. Yleisesti ottaen tietolähteet suunnitellaan itsenäisesti (autonomisesti) organisaatioiden omaan käyttöön. Tästä seuraa, että eri organisaatioiden luomissa tietolähteissä tietojen välillä esiintyy heterogeenisiä tekijöitä, jotka ilmenevät tietokonflikteina, kun niitä pitäisi käyttää yhdessä. Esimerkkinä tällaisesta tilanteesta on *avoimen tiedon* (engl. Open Data) yhteiskäyttö. Avoimella tiedolla tarkoitetaan viranomaisten tai muiden vastaavien tahojen tuottamaa ja julkaisemaa tietoa, joka on avoimesti (esim. verkon kautta) saatavilla ja jonka käytölle ei ole asetettu rajoitteita. Suurin osa näistä tiedoista on julkaistu myös XML-formaatissa [Brainschweig *et al,* 2012]. Käyttäjä saattaa haluta yhdistää kahden eri viranomaisen julkaisemaa tietoa, mutta tiedot on esitetty erilaisilla rakenteilla. Satunnaisen tietotarpeen tyydyttäminen tämankaltaisista lähteistä voi siten osoittautua hankalaksi, koska saatavilla ei ole liiketoimintatiedon hallintajärjestelmiä (engl. business intelligence tools), jotka mahdollistaisivat tietolähteiden integroinnin [Eberius *et al.,* 2012].

Dataspace on tietovaranto, johon on koottu tiettyä aihepiiriä koskevaa relevanttia informaatiota. Dataspacen sisältämät tietolähteet ovat usein käyttäjälle etukäteen tuntemattomia muun kuin aiherelevanssinsa suhteen, koska ne on noudettu käyttämällä avainsanahakuja. Käyttäjä tietää siis, että kyseiset lähteet käsittelevät tiettyä aihetta, mutta niiden käyttäjärelevanssi, eli miten sisältö tyydyttää käyttäjän tietotarpeen, on tuntematon. XML-dataspace on tietovaranto, jossa tietolähteet esitetään XML-formaatissa. XML-dataspace-järjestelmän tulee tarjota työkaluja, jotka auttavat käyttäjää analysoimaan tietolähteen sisältöä, siinä vallitsevia rakenteita sekä avustaa häntä liittämään tietolähteessä oleviin tietoihin oikean semantiikan [Näppilä *et al.,* 2011]. Siinä tapauksessa, että tietolähteet eivät ole keskenään yhteensopivia, tulee niiden väliset tietokonfliktit poistaa ennen kuin niiden yhteiskäyttö on mahdollista. Jotta tietokonfliktit voidaan poistaa, pitää ne ensin havaita. Tässä tutkielmassa on kehitetty visuaalisia primitiivejä tietokonfliktien havaitsemiseen ja poistamiseen ts. XML-dataspacen profilointiin ja harmonisointiin.

Suurten XML-tietolähteiden visualisointi saattaa aiheuttaa ongelmia: näyttöruudusta voi loppua tila rakennepuun ollessa tarpeeksi leveä, tietopilvi ei ota huomioon sanojen semanttista valikoivuutta ja laskentaprosessit voivat viedä paljon aikaa. Tästä syystä esitettyjen profilointi- ja harmonisointityökalujen ominaisuuksia voisi vielä parantaa esim. kehittämällä periaatteita, joilla usealle näyttöruudulle levittyvä visualisointi voidaan esittää intuitiivisesti ja järjestelmällisesti. Tämän lisäksi rakennepuun visualisoitavaa sisältöä voisi suodattaa: käyttäjä määrittelisi ennalta (esim. säännöllisillä lausekkeilla) millaisia tietoalkiota haluaa järjestelmän visualisoivan ts. suodattavan kaiken muun sisällön pois. Kehitetty tietopilvi ei ota huomioon semanttisia painotuksia. Tästä seuraa se, että tiettyjä yleisiä sanoja (merkkijonoja) korostetaan muita useammin. Käyttäjän määrittelemillä sulkusanalistoilla voitaisiin semanttisesti ei-valikoivat sanat ja muut käyttäjän mielestä epärelevantit sanat jättää huomioimatta. Semanttiset painotukset (esim. tf*idf) taas mahdollistaisivat tietopilven tehokkaamman käytön kokoelmia käsiteltäessä. Relaatiotietokantaa voisi hyödyntää yhä lisää tehostamaan käsittelyä käyttämällä esiprosessoituja tauluja, joilla pienennettäisiin asiakasohjelman ajonaikaista kuormitusta.

7 YHTEENVETO

Tutkielmassa esiteltiin ja kehitettiin mekanismeja XML-tietolähteiden visualiseen käsittelyyn ja analysointiin. Näiden mekanismien taustalla on Niemen ja Järvelinin [2006] kehittämä XML-relaatio, joka mahdollistaa XML-tietolähteiden tallettamisen perinteisillä relaatiotietokantajärjestelmillä kadottamatta niistä mitään informaatiota.

Tutkielmassa kehitettyjen mekanismien ytimenä on visuaalinen rakennepuu. Se noudattaa ns. maksimaalisuusperiaatetta, eli siinä analysoidaan kaaviotasolle kuuluvat tiedot visualisoimalla ne siten, että tietyllä hierarkiatasolla kaikki samannimiset elementtien ja attribuuttien ilmentymät kuvataan yhtenä suorakulmiona, jonka nimenä käytetään ko. elementtien/attribuuttien yhteistä nimeä. Kyseisiä nimettyjä suorakulmioita kutsutaan tietoalkioiksi. Tutkielmassa annetaan visuaalinen metafora XML-relaation visualisoimiseksi rakennepuuna. Rakennepuun kautta käyttäjä voi myös muokata tietoalkioiden sisältöjä ja tuottaa niistä erilaisia aggregointeja. Tietoalkioihin liittyvä semantiikka voidaan selvittää tutkimalla ilmentymätason tietoja. Näitä tietoja voidaan analysoida käyttämällä tietopilveä. Se muodostetaan käyttäjän valitsemasta rakennepuun tietoalkiosta alkavaan alipuuhun liittyen. Tietopilvi on mahdollista muodostaa joko koko tietolähteestä valitsemalla rakennepuun juuri tai vain osasta siitä. Tietopilvessä sanojen koko ilmaisee esiintymien runsauden ja väri sen, esiintyykö sana kaavio- vai ilmentymätasolla vai molemmilla. Käyttäjä kykenee myös navigoimaan rakennepuussa osoittamalla tietopilvessä esiintyviä sanoja. Aktivoimalla hiirellä tietty sana ilmaisee järjestelmä keltaisella värillä ne tietoalkiot, joissa ko. sana rakennepuussa esiintyy. Tutkielmassa kehitettiin sekä visuaalinen metafora tietopilven konstruoimiseksi että mekanismit sen käsittelyyn.

RXQL on kyselykieli, joka soveltuu etenkin heterogeenisten ja tieto-orientoituneiden XML-dokumenttien käsittelyyn. Tutkielmassa on kehitetty visuaalinen käyttöliittymä mahdollistamaan RXQL:n ilmaisuvoima XML-dataspace-järjestelmän yhteydessä. Tämä käyttöliittymä pyrkii hyödyntämään ihmisen visuaalista havainnointikykyä tekstuaaliseen syntaksiin perustuvan määrittelyn sijasta ts. se helpottaa syntaksin hallitsemisen taakkaa. Käyttöliittymän avulla voidaan ilmaista sekä dataspace-järjestelmän tyypilliset analysointioperaatiot että heterogeenisten tietolähteiden käsittelyyn perustuva tietotarpeiden tyydyttäminen yhdellä yhdenmukaisella tavalla.

LÄHDELUETTELO

[Abiteboul *et al.*, 1999] Serge Abiteboul, Sophie Cluet, Tova Milo, Pini Mogilevsky, Jerome Siméon, and Sagit Zoharz, Tools for data translation and integration. Bulletin of the Technical Committee on Data Engineering 22, 1 (Mar. 1999), pp. 3–8.

[Aho *et al.*, 1973] Alfred Aho, John Hopcroft, and Jeffrey Ullman, On finding the lowest common ancestors in trees. In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pp. 253-265.

[Arvola, 2011] Paavo Arvola, The role of context in matching and evaluation of XML information retrieval. University of Tampere, School of Information Sciences, June 2011.

[Bateman *et al.*, 2008] Scott Bateman, Carl Gutwin, and Miguel Nacenta, Seeing things in the clouds: The effect of visual features on tag cloud selections. In: *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pp. 193–202.

[Braunschweig *et al.*, 2012] Katrin Braunschweig, Julian Eberius, Maik Thiele, and Wolfgang Lehner, OPEN – Enabling non-expert userts to extract, integrate and analyse open data. *Datenbank-Spektrum* **12**, 2 (2012), 121–130.

[Burbeck, 1987/1992] Steve Burbeck, Applications programming in Smalltalk-80™: How to use Model-View-Controller (MVC). URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[Cardoso and Bussler, 2011] Jorge Cardoso and Christoph Bussler, Mapping between heterogeneous XML and OWL transaction representations in B2B integration. Data & Knowledge Engineering, **70**, 12 (Dec. 2011).

[Catarci *et al.*, 1995] Tiziana Catarci, Maria F. Costabile, and Maristella Matera, Which metaphor for which database? In: *Proceedings of the HCI'95 Conference on People and Computers*, pp. 151–165.

[Chamberlin *et al.*, 2001] Don Chamberlin, Jonathan Robie, and Daniela Florescu, Quilt: An XML query language for heterogeneous data sources. *Lecture Notes in Computer Science*, **1997** (2001), pp. 1–25.

[Codd, 1969] Edgar F. Codd, Derivability, redundancy, and consistency of relations stored in large data banks. IBM Research Report, 1969.

[Codd, 1970] Edgar F. Codd, A relational model of data for large shared data banks. *Communications of the ACM* **13,** 6 (Jan. 1970), 377–387.

[Codd, 1985a] Edgar F. Codd, Is your DBMS really relational? *Computerworld* (Oct. 1985).

[Codd, 1985b] Edgar F. Codd, Does your DBMS run by the rules? *Computerworld* (Oct. 1985).

[Cruz and Xiao, 2009] Isabel F. Cruz and Huiyong Xiao, Ontology driven data integration in heterogeneous networks. Complex Systems in Knowledge-based Environments **168** (2009), pp. 75–98.

[Date, 2003] Christopher J. Date, *An Introduction to Database Systems.* Addison-Wesley, 2003.

[Deutsch *et al.,* 1998] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu, XML-QL: A Query Language for XML. URL: http://www.w3.org/TR/NOTE-xml-ql/

[Eberius *et al.*, 2012] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. Enabling business analysts to explore the web of open data. In: *Proceedings of the VLDB Endowment* **5**, 12 (Aug. 2012), pp. 1978–1981.

[Florescu and Kossmann, 1999] Daniela Florescu and Donald Kossmann, Storing and querying XML data using RDBMS. *Bulleting of the Technical committee on data engineering* **22**, 3 (Sep. 1999), pp. 27–34.

[Franklin *et al.*, 2005] Michael Franklin, Alon Halevy, and David Maier, From databases to dataspaces: A new abstraction for information management. *ACM SIGMOND* **34**, 4 (Dec. 2005), pp. 27–33.

[Goh, 1997] Chen Hiang Goh, Representing and reasoning about semantic conflicts in heterogeneous information Systems. Massachusetts Institute of Technology, Sloan School of Management, 1997.

[Graaumans, 2004] Joris Graaumans, A qualitative study to the usability of three XML query languages. In: *Proceedings of Dutch HCI '04,* pp. 6–9.

[Haber *et al.*, 1994] Eben M. Haber, Yannis E. Ioannidis, and Miron Livny, Foundations of visual metaphors for schema display. *Journal of Intelligent Information Systems* **3,** 3-4 (1994), pp. 263–298.

[Halevy *et al.*, 2006] Alon Halevy, Michael Franklin, and David Maier, Principles of dataspace systems. In: *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 243–252.

[Halverson *et al.*, 2004] Alan Halverson, Vanja Josfifovski, Guy Lohman, Hamid Pirahesh, and Mathias Mörschel, ROX: Relational over XML. In: Proc*eedings of the 30th international conference on Very large data bases*, pp. 264-275.

[Hearst, 2009] Marti A. Hearst, *Search User Interfaces*. Cambridge University Press, 2009.

[Howe *et al.*, 2008] Bill Howe, David Maier, Nicolas Rayner, and James Rucker, Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In: *Proceedings of the 24th International conference on data engineering workshops*, pp. 270–277.

[Humphrey, 1988] Watts S. Humphrey, Characterizing the software process: A maturity framework. *IEEE Software* **5**, 2 (Mar. 1988), pp. 73–79.

[ISO, 1986] ISO 8879:1986, Information processing — Text and office systems —Standard Generralized Markup Language (SGML).

[ISO, 2003] ISO/IEC IS 9075-14:2003, Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML). URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc/catalogue_detail.html?csnumber=5386

[ISO, 2008] ISO/IEC IS 9075-1:2008, Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498

[Kay, 2006] Michael H. Kay, Choosing between XSLT 2.0 and XQuery 1.0. URL: http://www.w3.org/2006/Talks/0525-www2006-Kay.pdf

[Kepser, 2004] Stephan Kepser, A proof of Turing-completeness of XSLT and XQuery. University of Tübingen, Report **SFB 441**, 2004.

[Kim and Seo, 1991] Won Kim and Jungyun Seo, Classifying schematic and data heterogeneity in multidatabase systems. *Computer* **24**, 12 (Dec. 1991), pp. 12–18.

[Lee *et al.*, 1995] Chiang Lee, Chia-Jung Chen, and Hongjun Lu, An aspect of query optimization in multidatabase systems. *ACM SIGMOND* **24**, 3 (Sep. 1995), pp. 28–33.

[NetBeans Team, 2007] NetBeans Team, The Visual Library 2.0. URL: http://platform.netbeans.org/graph/

[Niemi and Järvelin, 2006] Timo Niemi and Kalervo Järvelin, Another look at XML. University of Tampere, Dept. of Computer Science, Report **A-2006-1**, June 2006.

[Niemi *et al.*, 2009] Timo Niemi, Kalervo Järvelin, and Turkka Näppilä, A relational data harmonization approach to XML. *Journal of Information Science* **24**, 5 (Oct. 2009), pp. 571–601.

[Näppilä *et al.*, 2008] Turkka Näppilä, Kalervo Järvelin, and Timo Niemi, A tool for data cube construction from structurally hetergenous XML documents. *Journal of American Society for Information Science and Technology* **59**, 3 (Feb. 2008), pp. 435–439.

[Näppilä *et al.*, 2011] Turkka Näppilä, Katja Moilanen, and Timo Niemi, A query language for selecting, harmonizing, and aggregating heterogeneous XML data. *International Journal of Web Information Systems* **7**, 1 (2011), pp. 62–99.

[Näppilä and Niemi, 2012] Turkka Näppilä and Timo Niemi, An approach for developing a schemaless XML dataspace profiling system. In: *Journal of Information Science* **38**, 3 (2012), pp. 234–257.

[Pal *et al.*, 2004] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Glakoumakis, and Vasili Zlotov, Indexing XML data stored in a relational database. In: *Proceedings of the Thirtieth international conference on Very large data bases*, pp. 1134–1145.

[Papakonstantinou *et al.*, 1995] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom, Object exchange across heterogeneous information sources. In: *Proceedings Eleventh International Conference on Data Engineering*, pp. 251–260.

[Rivadeneira *et al.*, 2007] A.W. Rivadeneira, Daniel M. Gruen, Michael J. Muller, and David R. Millen, Getting our head in clouds: Toward evaluation studies of taglouds. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 995–998.

[Robie *et al.*, 1998] Jonathan Robie, Joe Lapp, and David Schach, XML Query Language (XQL). URL: http://www.w3.org/TandS/QL/QL98/pp/xql.html

[Robie *et al.*, 2011] Jonathan Robie, Don Chamberlin, Michael Dyck, and John Snelson, XQuery 3.0: An XML Query Language. URL: http://www.w3.org/TR/xquery-30/

[Shanmugasundaram *et al.*, 1999] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton, Relational databases for querying XML documents: limitations and opportunities. In: *Proceedings of the 25th international conference on Very large data bases*, pp. 302–314.

[Turing, 1937] A.M. Turing, On computable numbers, with an application to the entscheidungsproblem. In: *Proceedings of London Mathematical Society* **42**, 2 (Jan. 1937), pp. 230–265.

[Ullman, 1988] Jeffrey D. Ullman, *Principles of Database and Knowledge-base Systems 1, Classical Database Systems*. Computer Science Press, 1988.

[W3C, 2001] The World Wide Web Consortium (W3C), XML Fragment Interchange. URL: http://www.w3.org/TR/xml-fragment

[W3C, 2002] The World Wide Web Consortium (W3C), XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). URL: http://www.w3.org/TR/xhtml1/

[W3C, 2004a] The World Wide Web Consortium (W3C), XML Schema. URL: http://www.w3.org/TR/xmlschema-1/

[W3C, 2004b] The World Wide Web Consortium (W3C), XML Information Set (Second edition). URL: http://www.w3.org/TR/xml-infoset/

[W3C, 2007] The World Wide Web Consortium (W3C), XSL Transformations (XSLT) Version 2.0. URL: http://www.w3.org/TR/xslt20/

[W3C, 2008] The World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.0 (Fifth Edition). URL: http://www.w3.org/TR/xml/

[W3C, 2009] The World Wide Web Consortium (W3C), Namespaces in XML 1.0 (Third Edition). URL: http://www.w3.org/TR/REC-xml-names/

[W3C, 2010a] The World Wide Web Consortium (W3C), XQuery 1.0. URL: http://www.w3.org/TR/xquery/

[W3C, 2010b] The World Wide Web Consortium (W3C), XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second edition). URL: www.w3.org/TR/xpath-datamodel

[W3C, 2010c] The World Wide Web Consortium (W3C), XML Syntax for XQuery 1.0 (XQueryX). URL: www.w3.org/TR/xqueryx

[W3C, 2011a] The World Wide Web Consortium (W3C), Scalable Vector Graphics (SVG) 1.1 (Second edition). URL: www.w3.org/TR/SVG/

[W3C, 2011b] The World Wide Web Consortium (W3C), XML Path Language (XPath) Version 2.0 (Second edition). URL: http://www.w3.org/TR/xpath20/

[W3C, 2011c] The World Wide Web Consortium (W3C), XQuery Update Facility 1.0. URL: http://www.w3.org/TR/xquery-update-10/