Jari Marttinen

SEMANTTINEN MUUNNOS

Tietotekniikan pro gradu -tutkielma

Tietoliikenteen linja

27.7.2008

Jyväskylän yliopisto

Tietotekniikan laitos

Tiivistelmä

Marttinen, Jari Pekka

Semanttinen muunnos

Jyväskylä: Jyväskylän yliopisto, 2008

94 s.

Pro gradu -tutkielma

Tutkielma käsittelee erilaisten resurssien muuntamista semanttisen webin standardien mukaiseksi. Pääpainona ovat teollisuuteen ja erityisesti paperiteollisuuteen liittyvät dynaamiset resurssit.

Tutkielma liittyy Tekesin rahoittamaan SmartResource-projektiin ja projektin siihen osaan, missä tutkielman tekijä on ollut mukana, eli semanttisen muunnoksen toteuttamiseen.

Tutkielmassa käsitellään semanttista muunnosta ja sen tuomia haasteita. Tutkielman tavoitteena on esitellä yleinen sovitinympäristö GAF (General Adaptation Framework), minkä avulla erilaisia teollisuuden resursseja saadaan muunnettua XML-muodosta semanttisesti rikkaampaan RscDF (Resource State/Condition Description Framework) -muotoon.

Tutkielmassa tutustutaan XSLT sekä XPath -kieliin ja tarkastellaan tapoja, kuinka XML-muotoista dataa pystytään muuntamaan RDF:ksi ja sen laajennuksiksi kuten RscDF:ksi.

Tutkielman tuloksena esitetään semanttisen muunnoksen haasteisiin ratkaisuksi kaksiosaista muunnosta, minkä avulla saadaan muunnettua myös dynaamiset resurssit semanttisesti rikkaampaan muotoon. Lisäksi esitetään prototyyppisovellus, minkä avulla semanttinen muunnos pystytään toteuttamaan.

AVAINSANAT: RscDF, XML, XSLT, XPath semanttinen muunnos, semanttinen web

Abstract:

Marttinen, Jari Pekka

Semantic transformation

Jyväskylä: University of Jyväskylä, 2008

94 p.

Master's Thesis

The thesis examines how different resources could be transformed into the semantic web standards. The main accents are industrial resources, especially paper industries dynamic resources.

The thesis is based on SmartResource project, which is funded by Tekes, and to that part of the project where author of the thesis have participated.

The thesis examines semantic transformation and challenges, what come with it. Thesis goal is to present General Adaptation Framework to transform different industrial resources from XML to RscDF (Resource State/Condition Description Framework), which provides rich semantic descriptions to resource data.

Thesis explores XSLT and XPath languages and studies ways, how to transform XML based data to RDF and to its extensions like RscDF.

Thesis gives results how to beat challenges, which come from semantic transformation. Also thesis introduces two-stage transformation what collaborates the whole semantic transformation process. At the end of the thesis a prototype environment is presented to perform semantic transformation.

KEYWORDS: RscDF, XML, XSLT, XPath, semantic transformation, semantic web

Sisältö

1 Johdanto

Heterogeenisten sovellusten ja datalähteiden integrointi toistensa kanssa yhteensopiviksi järjestelmiksi on eräs suurimmista tämän päivän tietoyhteiskunnan haasteita. Globaalin ympäristön kehittäminen, mikä tukisi tiedon siirtämistä ihmisekspertiltä automaattiselle ja itse oppivalle web-palvelulle, on erittäin haasteellinen tehtävä. Teollisuuden eri laitteiden ylläpito kuuluu myös tämän haasteen piiriin.

Teollisuuden piirissä on kuitenkin huomattu, että datan semantiikan esittäminen on ensiarvoisen tärkeää sovellusten integroimiseksi. Vaikka semanttinen web tarjoaa globaalina standardin tiedon semanttisen sisällön esittämiselle, on kuitenkin liian myöhäistä, kallista ja työvoimaa kuluttavaa toimintaa siirtää metadataa paikallisesta standardista globaaliin standardiin.

Eräs ratkaisu tämän haasteen ratkaisemiseksi on suunnitella semanttisia sovittimia, mitkä toimisivat ikään kuin ”tulkkeina”, ja muuntaisivat paikallisen standardin mukaisen metadatan semanttisen webin standardien mukaisiksi. Tämä on ollut lähtökohtana yleisen sovitinympäristön kehittämiselle, mikä antaa lähtökohdat semanttisten sovittimien rakentamiselle.

Tämä tutkielma liittyy SmartResource-projektiin ja sen ensimmäiseen vuoteen. SmartResource oli kolmivuotinen projekti, minkä tarkoituksena oli antaa työkalut ja ratkaisut erilaisten teollisuuden resurssien saattamiseksi web-käyttöisiksi, proaktiivisiksi ja yhteistyökykyisiksi. Tällä tarkoitan sitä, että resurssit pystyisivät analysoimaan omaa tilaansa riippumatta muista tai tilaamaan analyysin joko etäekspertiltä tai web-palvelulta. Tällöin resurssit pystyisivät tietämään oman tilansa koko ajan ja suunnittelemaan omaa käyttäytymistään kohti tehokasta ja ennustavaa ylläpitoa. Tämä tutkielma esittelee ratkaisun semanttisen muunnoksen tekemiseen, missä muutetaan XML (eXtensible Markup Language) -muotoista dataa erityisesti teollisuuden resurssien ylläpitoon tarkoitettuun RscDF (Resource State/Condition Description Framework) -muotoon.

Tutkielman toisessa luvussa esitetään keskeiset käsitteet. Kolmannessa luvussa esitellään yleinen sovitinympäristö ja neljännessä luvussa kerrotaan semanttisesta muunnoksesta ja

sen haasteista. Viidennessä luvussa esitellään XSLT (XSL Transformations) ja XPath (XML Path Language) sekä niiden ominaisuudet. Kuudennessa luvussa esitetään kuinka XML-muotoista dataa voidaan muuntaa RscDF-muotoon. Seitsemännessä luvussa esitellään sovittimen pilottiversio. Lopuksi yhteenvedossa kootaan tutkimuksen tulokset ja keskeinen sisältö.

2 Keskeiset käsitteet

Tässä luvussa esitetään tutkielman kannalta keskeiset käsitteet semanttisesta webistä ja sen standardeista.

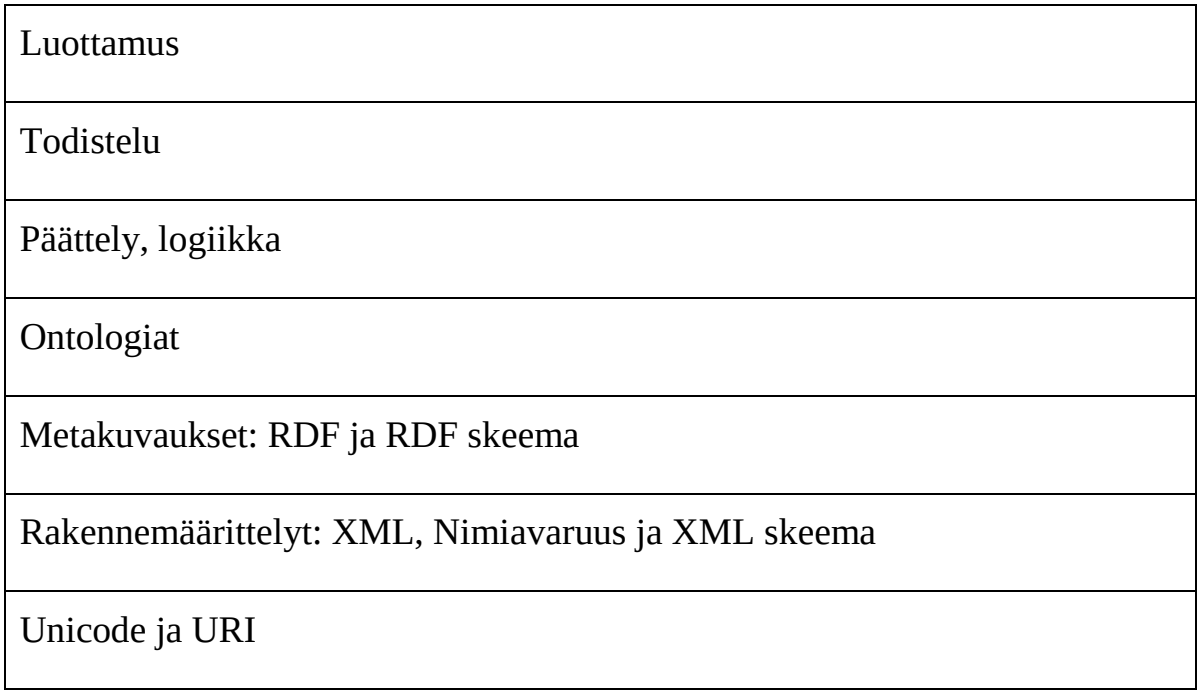
2.1 Semanttinen web

Tänä päivänä webissä oleva tieto on suunniteltu ihmisten luettavaksi, eikä tietokoneiden itsensä luettavaksi. Tietokoneet voivat kyllä prosessoida tietoa joiltain osin, kuten esim. dokumentin rakennetta ja ulkoasua, mutta ne eivät ymmärrä tiedon todellista *semanttista* merkitystä. Tämä estää tietokoneiden käyttää webissä olevaa tietoa itsenäisesti ja automaattisesti. Semanttinen web antaa tarvittavat työkalut tiedon semanttisen sisällön esittämiseen. (Berners-Lee, Hendler & Lassila 2001.)

Semanttinen web on W3C:n (World Wide Web Consortium) visio ja sen tarkoituksena on antaa tarvittavat teknologiat internetsivujen semanttisen sisällön tuottamiseen. Se ei kuitenkaan ole mikään erillinen web, vaan ainoastaan nykyisen webin seuraava kehitysaskel ja laajennus. Tieto esitetään semanttisessa webissä myös formaalissa tietokoneiden ymmärtämässä muodossa, mikä mahdollistaa tietokoneen ja ihmisten paremman yhteistyön. Tärkeimmät kielet semanttisen webin takana ovat RDF (Resource Description Framework) ja OWL (Web Ontology Language). (Paolucci M. & Sycara K. 2003)

Semanttisen webin arkkitehtuuri voidaan esittää kerrosmallin avulla, mikä on esitetty kuviossa 1. Oheisessa kuviossa alimmalla rivillä ovat Unicode ja URI (Uniform Resource Identifier) -kerrokset, mitkä takaavat kansainvälisten merkkien käytön ja ennalta määrätyn tavan resurssien osoittamiseen ja identifioimiseen webissä. Seuraavalla tasolla XML (eXtensible Markup Language), nimiavaruus ja XML-skeema määrittelevät rakenteellisen ja syntaktisen yhteensopivuuden muiden XML standardien kanssa. RDF ja RDF-skeema mahdollistavat väittämien tekemisen URI-resursseista sekä sanastojen luomisen, millä voidaan antaa resursseille tyyppejä. Ontologiataso tukee sanastojen kehittämistä. Päättely- ja logiikkakerros mahdollistaa päättelysääntöjen tekemisen alla olevan ontologian pohjalta.

Todistelukerros suorittaa säännöt ja arvioi luottamuskerroksen kanssa sovellusten luotettavuutta. (Koivunen M. & Miller E. 2001)



Kuvio 1. Semanttisen webin arkkitehtuuri. (Koivunen M. & Miller E. 2001)

2.2 XML ja XML-skeema

XML on W3C:n kehittämä metakieli. XML oli alun perin tarkoitettu vastaamaan laaja-alaisen elektronisen kustannustoiminnan haasteisiin. XML muistuttaa hyvin paljon HTML (Hypertext Markup Language) -kieltä ja käyttää elementtejä ja attribuutteja. HTML:ssä jokaisella merkillä ja attribuutilla on oma merkityksensä, kun taas XML:ssä elementtejä käytetään datan rajaamiseen. XML mahdollistaa jokaisen käyttäjän määrittää omat elementtinsä ja attribuuttinsa ja niiden tulkinta jää täysin lukijan vastuulle. (Philips 2000)

XML on joukko sääntöjä jonkin rakenteellisen tiedon kuvailemiseen. Se mahdollistaa helpon tavan tuottaa, lukea ja varmistaa, että tiedon rakenne on yksiselitteinen. XML on Unicode yhteensopiva, avorakenteinen ja alustasta riippumaton. (Harold 1999)

Jokaisella XML-dokumentilla on fyysinen ja looginen rakenne. Fyysisesti dokumentti koostuu alkioista, joita kutsutaan elementeiksi. Loogisesti dokumentti on määrityksiä,

elementtejä, merkkiviitteitä ja prosessointiohjeita, mitkä ilmaistaan ja erotellaan dokumentissa yksitulkintaisella merkkauksella. Esimerkki 1 esittää yksinkertaisen XML-dokumentin Jyväskylän yliopiston kotisivuista ja sen skeematiedoston.

Esimerkki 1. XML-dokumentti ja sen skeema.

```
<?xml version="1.0" encoding="UTF-8"?>
<Homepage>
    <Title>Jyväskylän yliopiston kotisivut</Title>
    <Language>fi</Language>
</Homepage>

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Homepage">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Title" />
          <xs:element name="Language" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

XML-skeema on kieli, millä kuvataan XML-dokumentin rakenne. Skeema muodostuu joukosta sääntöjä, mitkä määrittelevät XML-dokumentin rakenteen, elementit, attribuutit sekä elementtien ja attribuuttien tietotyypit ja sallitut arvojoukot.

XML-skeema on myös W3C:n julkaisema. Se kehitettiin vastaamaan nykypäivän haasteita, mihin DTD (Document Type Definition) ei pelkästään riittänyt.

2.3 RDF ja RDF-skeema

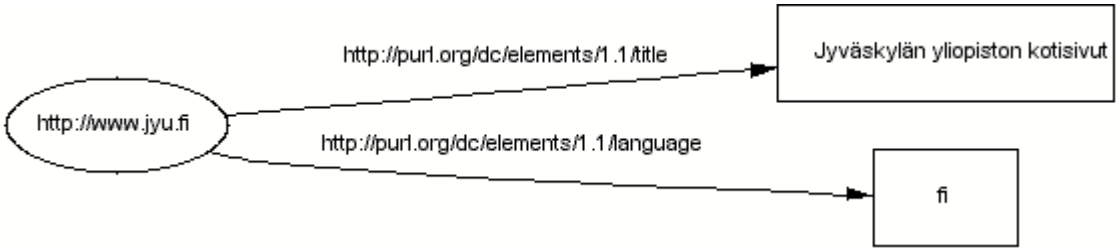
RDF on W3C:n kehittämä monikäyttöinen kieli webin resurssien, kuten www-sivujen kuvausten ja niiden metatietojen esittämiseen. Se mahdollistaa yhteensopivuuden sovellusten välillä, mitkä vaihtavat keskenään konelukuista informaatiota webissä. RDF on suunniteltu tilanteisiin, missä informaatiota käyttää sovellus, eikä sitä tarvitse näyttää ihmiselle. RDF antaa yleiset puitteet informaation ilmaisulle ja sen vaihtamiselle sovellusten välillä informaation tarkoituksen kuitenkaan häviämättä. (Klyne G. & Carroll J.)

RDF on erityisesti tarkoitettu metadatan esittämiseen webin resursseista. Resurssit voidaan kuvata tietokoneiden ymmärtämillä väittämillä. Väittämät ovat kolmikkoja (resurssi, ominaisuus, arvo) tai (subjekti, predikaatti, objekti). (Manola & Miller 2004; Becket 2004)

Esimerkki 2 on tyypillinen pieni RDF-dokumentti, missä kuvaillaan esimerkissä 1 esitettyä Jyväskylän yliopiston kotisivuja kuvaavaa XML-dokumenttia ja kuvio 2 esittää RDF-dokumentin graafin.

Esimerkki 2. RDF dokumentti.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="http://www.jyu.fi">
    <dc:title>Jyväskylän yliopiston kotisivut</dc:title>
    <dc:language>fi</dc:language>
</rdf:Description>
</rdf:RDF>
```



Kuvio 2. RDF:n graafinen esitys.

RDF-skeema on kieli, millä kuvataan RDF:n sanasto, eli se on RDF:n semanttinen laajennus. Se antaa mekanismin kuvailla RDF:ssä käytetty sanasto sekä varmistaa sen yhtenäisyyden. RDF-skeema määrittää luokkia ja ominaisuuksia lähes samalla tavalla kuin tavallinen olio-ohjelmointikieli. RDF:ssä voidaan määritellä ominaisuus `eg:Author`, minkä yläluokka on `eg:Document` ja arvojoukkona `eg:Person`. Tavallisessa olio-ohjelmointikielessä taasen määriteltäisiin luokka `eg:Document`, millä olisi tyyppiä `eg:Person` oleva attribuutti `eg:Author`.

RDF-skeeman resurssilähtöinen määrittely mahdollistaa luokkien laajentamisen myöhemmin lisäämällä lisää ominaisuuksia. Tämä voidaan tehdä siten, ettei alkuperäistä luokkaa tarvitse lähteä muuttamaan. Kuitenkaan RDF-skeema ei itsessään kerro luokkien ja ominaisuuksien tarkoitusta. Tämän vuoksi on kehitetty erilaisia ontologia-kieliä. ((Brickley & Guha 2004)

2.4 OWL

Kuten kuvio 1 osoittaa, tulee XML:n ja RDF:n jälkeen neljännellä tasolla ontologiat. Ontologia on kieli, millä kuvataan formaalilla tavalla web-sivujen luokkien ja ominaisuuksien semantiikka. Sen täytyy mennä semanttisessa kuvauksessaan pidemmälle kuin RDF-skeeman, että tietokoneet voisivat käyttää ja prosessoida verkkoon talletettua tietoa laajamittaisesti ja tehokkaasti.

OWL on W3C:n esittelemä ratkaisu ontologioiden esittämiseen. OWL on käyttökelpoinen niissä tapauksissa, kun dokumenteissa oleva informaatio on tarkoitettu erilaisten

sovellusten prosessoitavaksi. Jos dokumenttien informaatio on tarkoitettu pelkästään ihmisten luettavaksi, ei OWL tarjoa mitään lisähyötyä. OWL pystyy määrittelemään eksplisiittisesti sanastojen termien merkityksen sekä termien väliset yhteydet. (Heflin 2004)

OWL on jakautunut kolmeen eri alakieleen, mitkä ovat OWL Lite, OWL DL ja OWL Full. Näistä OWL Lite on kaikkein yksinkertaisin ja asettaa useita rajoitteita rakenteiden käytölle. OWL DL mahdollistaa monimutkaisemmat rakenteet ja OWL Full ei aseta mitään rajoituksia rakenteiden tekoon. Näistä OWL Full on ainoana täysin yhteensopiva RDF:n ja RDF-skeeman kanssa. (McGuinnes & Harmelen. 2004)

3 Yleinen sovitinympäristö

Tässä luvussa esitellään yleinen sovitinympäristö, minkä avulla semanttinen muunnos pystytään toteuttamaan. Johdanto kertoo sovitinympäristön kehittämisen taustan ja sen jälkeen esitellään sen perusperiaate. Lopuksi kuvataan RscDF-malli, sen rakenne ja käyttötarkoitus.

3.1 Johdanto

Maailmassa on tänä päivänä useita erilaisia tietojärjestelmiä, mitkä eivät ole yhteensopivia keskenään. Lisäksi useilla järjestelmillä on omat yksilöidyt tehtävänsä ja käyttötapansa, joten tällaisten erilaisten järjestelmien yhteensovittaminen on erittäin vaikeaa. Järjestelmiä on rakennettu erilaisille käyttöjärjestelmille, erilaisten protokollien ja dataformaattien varaan, mikä tekee yhteensovittamisesta entistäkin haastavampaa.

Yritykset ovat käyttäneet suuren määrän aikaa ja rahaa suunnitellakseen ja toteuttaakseen omia yrityskohtaisia järjestelmiä ja standardeja. Tästä johtuen on ymmärrettävää, etteivät yritykset halua heittää hukkaan vuosia kestäneitä kehitystöitä. Toisaalta yritykset yhdistyvät tänä päivänä entistä useammin keskenään, jolloin tietojärjestelmienkin tulisi olla yhteneviä. Kasvavat yhteentoimivuusvaatimukset ovatkin haaste monille yrityskohtaisille standardeille ja käytännöille.

Ottaen huomioon useat erityyppiset informaatioresurssit, dataformaatit ja protokollat, on tietojärjestelmien yhteensovittaminen erittäin tärkeä tutkimuksellinen haaste. (Khanna R. 2004)

Semanttinen web tarjoaa globaalin standardin tietojärjestelmien integraatiolle, mutta paikallisesta standardista globaaliin standardiin siirtyminen vaatii paljon työtä. Olisi liian kallista ja työlästä muuttaa suuri määrä dataa paikallisesta standardista globaaliin standardiin käsin, joten tarvitaan osittain automaattinen tapa datan muunnokseen. (Kaykova ym. 2005a, Terziyan 2005)

GUN-ympäristö (Global Understanding eNvironment) (Kaykova, Khriyenko, Kovalainen & Zharko 2004; Terziyan 2003) on semanttisen webin kaiken tutkimus ja

kehittämisponnistelujen perimmäinen tarkoitus. Sillä pyritään helpottamaan kaikkien erilaisten resurssien, mitä voidaan liittää webiin, proaktiivista, tarkoitushakuista ja itse itsensä ylläpitävää käyttäytymistä. Jotta kaikenlaiset, niin rakenteeltaan kuin luonnoltaan, erilaiset resurssit saataisiin toimivaan yhteensopivasti, pitää GUN-ympäristön perustua universaaliin metodologiaan resurssien integraatiosta. Tämän integraation toteuttamista varten tarvitaan yleiset puitteet, mitkä tarjoaa yleinen sovitinympäristö GAF (General Adaptation Framework). (Kaykova, Khriyenko, Kononenko, Terziyan, Zharko, 2004)

GAF:n suunnittelun lähtökohdaksi on otettu erilaiset teollisuuden resurssit, tarkoituksen tarjota ratkaisut ja työkalut resurssien saattamiseksi web-käyttöisiksi, proaktiivisiksi ja itse itsensä ylläpitäviksi. Tällä tarkoitetaan sitä, että resurssit osaavat itse analysoida omaa tilaansa riippumattomasti. Lisäksi resurssien tulisi pystyä suunnittelemaan omaa käyttäytymistään tehokkaasti ja ennustavampaa ylläpitoa kohti, käyttämällä tarvittaessa apuna etäeksperttejä tai web-palveluita. (Kaykova ym. 2005a)

3.2 GAF

3.2.1 GAF:n periaate

Useita erilaisia resursseja on tarkoitus integroida yleiseen GUN-ympäristöön. Tehokkaampaa analyysia varten kaikki resurssit on lajiteltu kolmeenperusluokkaan: laitteisiin, palveluihin ja ihmisiin. Nämä luokat edustavat reaalimaailman objekteja, joiden tulee olla vuorovaikutuksessa keskenään jollain tavalla. Tällaisten resurssien sovittaminen yhteen tarvitsee ympäristön, missä kaikki pystyvät kommunikoimaan keskenään yhtenäisellä tavalla käyttäen jotain standardoitua protokollaa.

GAF:n idea perustuu *sovittimelle* (adapter), mikä toimii siltana paikallisen esitystavan ja yleisen ympäristön välillä. Sovitin on avainasemassa kun reaalimaailman objekteja muunnetaan sopiviksi GUN-ympäristöön. Koska lähtökohdaksi on otettu teollisuuden resurssit ja niiden ylläpito, niin sovittimen tehtävänä on muuntaa objektit RscDF (Resource State/Condition Description Framework) -formaattiin. RscDF on varta vasten kehitetty teollisuuden resurssien ylläpidon tarpeisiin. (Kaykova ym. 2005b)

Päävaatimus GAF:lle on, että sen pitää pystyä tarjoamaan ainakin osittain automaattinen tapa muuntaa erilaisia datamalleja semanttisesti rikkaampaan muotoon. Tällaisia datamalleja ovat esim. XML-pohjaiset standardit ja erilaiset relaatiotietokannat.

GAF:n suunnittelu voidaan jakaa karkeasti kahteen osaan:

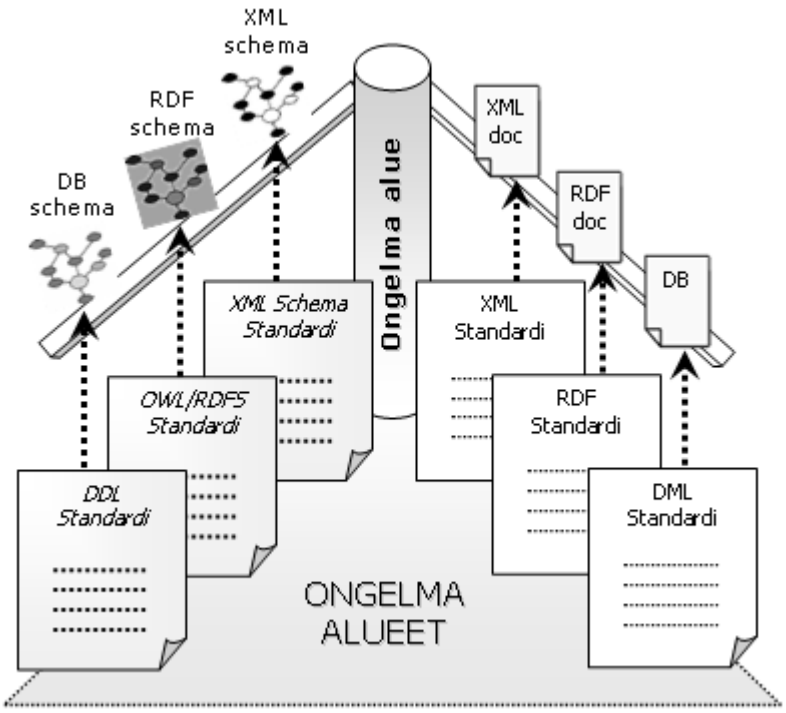
1. Erilaisten moduulien, luokkien ja protokollien suunnitteluun ja ohjelmointiin sekä
2. erilaisten datamallien semanttisen muunnoksen suunnitteluun ja toteutukseen.

Semanttinen muunnos on eräs ratkaiseva haaste suunniteltaessa ja rakennettaessa yleistä GAF-ympäristöä. Ongelman ratkaisemisen lähtökohta perustuu olettamukselle, että mikäli semanttisesti rikasta dataa käyttävät heterogeeniset ohjelmistokomponentit perustuvat yleiseen ontologiaan (Farrar S. ym. 2002), niin ne ovat keskenään yhteensopivia. (Malucelli A. & Oliveira E. 2003)

3.2.2 Datamallit

Erilaisia datamalleja löytyy tänä päivänä useita. Teollisuudessa käytetyimmät mallit ovat XML ja relaatiomalli. Varsinkin XML-muotoinen data on nostanut suosiotaan teollisuuden parissa. Uusia standardeja kuitenkin tulee esiin varsinkin semanttisen webin myötä, kuten RDF ja OWL. Nämä mallit ovat kuitenkin pääsäätöisesti keskittyneet semantiikkaan.

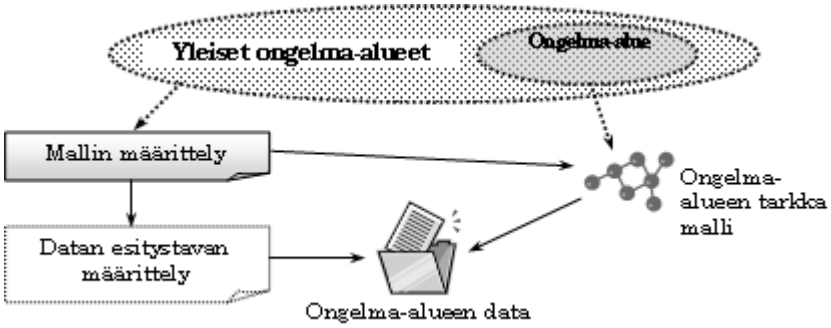
Jotta semanttinen muunnos olisi mahdollinen, täytyy kaikki datan esitysmallit tuntea, jotta pystyttäisiin ymmärtämään semanttisen muunnoksen perusolemus. Yleisimmät datamallit esitellään kuviossa 3.



Kuvio 3. Yleisimmät datamallit.

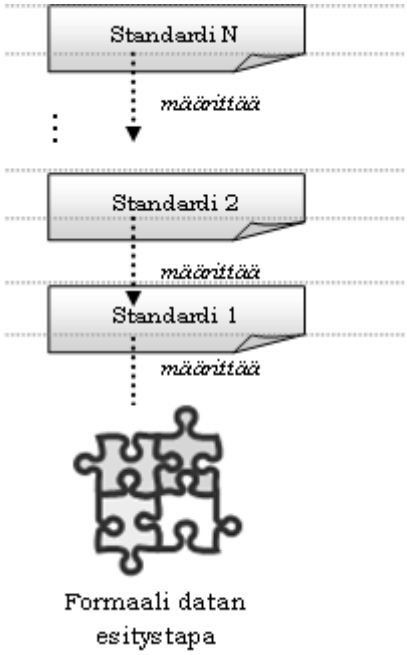
Niin kuin oheisesta kuviosta nähdään, perustuvat kaikki esitysmallit omalle määritetylle standardille. Toisaalta dokumentin, kuten XML:n, tulee perustua myös omaan skeemaansa. Peruslähtökohtana on jokaisen erilaisen tiedonmäärityskielen tulkinta erikseen. Kaikkien edellä esitettyjen esitysmallien standardit antavat suuntaviivat ongelma-alueiden formalisointiin ja ratkaisemiseen.

Yleisesti ottaen jokaisen datamallin rakenne perustuu omaan spesifikaatioonsa, mikä on jäsennetty tietyn skeeman ja skeeman oman spesifikaation antaman mallin mukaan. Kuviossa 4 on esitetty yleinen datan skeeman esitystavan malli.



Kuvio 4. Yleinen esitysmalli. (Kaykova ym. 2004a)

Erilaisilla esitysmalleilla voi olla myös yhteys toisiinsa. Abstraktimmat mallit määrittelevät täsmällisempiä. Useissa tapauksissa ovat eri mallit ketjuuntuneet toisiinsa, kuten kuviossa 5 esitetään. Tästä näkökohdasta katsottuna semanttisen muunnoksen tarkoitus on erottaa datan semantiikka esitysstandardista riippumatta. Tämä lähestymistapa on osoittautunut suunnittelun lähtökohdaksi, jotta pystytään mahdollistamaan datan muuntaminen toiseen esitystapaan kuitenkaan menettämättä datan alkuperäistä olemusta ja tarkoitusta. (Kaykova ym. 2005b)



Kuvio 5. Standardien välinen yhteys.

3.2.3 Semanttinen muunnos prosessi

Semanttisen muunnoksen aikana, muunnoksen tekevä objekti/moduuli käsittää niin datamallin metadatan eli skeeman sekä muunnossäännöt. Nämä muunnossäännöt, skeema ja ontologiakerros yhdessä muodostavat kehyksen semanttiselle muunnokselle.

Semanttisen muunnoksen täytyy määrittää toimiakseen toiminnallisuus seuraavien seikkojen kanssa:

- Sovittimen tarjoamien palvelujen kanssa.
- Sovitetun järjestelmän datan esitystavan standardien ja mallien kanssa.
- Sovitetun järjestelmän rajapintojen kanssa.
- Järjestelmän ajon aikaisen ympäristön kanssa.

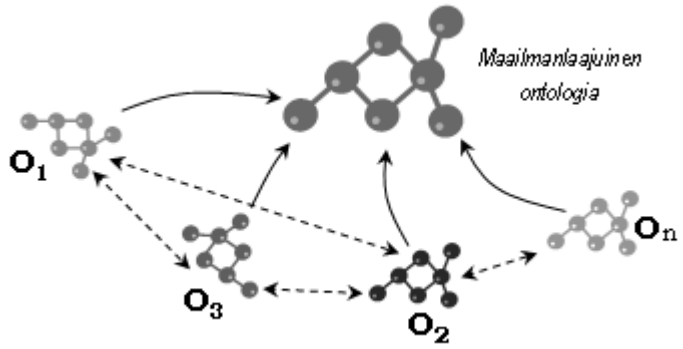
On todistettu, että täysin automaattista semanttista muunnosta ei kaikissa olosuhteissa pystytä toteuttamaan. Täysin yksiselitteiset semanttiset kuvaukset ovat kone-luettavia, joten automaattinen rakennekin pystytään ainakin joissain tapauksissa toteuttamaan. Siis herääkin kysymys, kuinka paljon automatisointia pystytään saavuttamaan. Kuitenkin ellei yksiselitteisen semanttisen kuvauksen taustalla ole yleistä ontologiaa, tarvitsee ihmisen ”opettaa” muunnokselle käsitteet ja suhteet jotta muunnos toimisi. Tästä syystä myös työkaluja tarvitaan muunnoksen helpottamiseksi. (Kaykova, Khriyenko, Kovalainen & Zharko 2004)

Automaattisen semanttisen muunnoksen taustalla on kolme välttämätöntä tapausta (Kaykova, Khriyenko, Kovalainen & Zharko 2004):

- Eksplisiittinen ihmisavusteinen muunnos.
- Jaettu ontologia, mikä tarkoittaa että kummatkin resurssit käyttävät samaa ontologiaa tai voidaan ainakin muuntaa siihen.
- Jaetusta ontologiasta tehtävät haut, mitkä voidaan toteuttaa palveluna tai sisäänrakennettuna ominaisuutena.

3.3 Ontologioiden välinen muunnos

Semanttisesta muunnoksesta puhuttaessa tulee väistämättä esiin ontologioiden välisten muunnosten haaste. Mikäli kaiken taustalla olisi yksi yhteinen maailmanlaajuinen ontologia, niin kaikkien alaontologien väliset muunnokset voitaisiin tehdä eksplisiittisesti. Kuvio 6 osoittaa ontologioiden välisen muunnoksen. Katkoviivaiset nuolet osoittavat alaontologioiden välisen muunnoksen ja tavalliset nuolet vastaavuuden yhteiseen maailmanlaajuiseen ontologiaan.



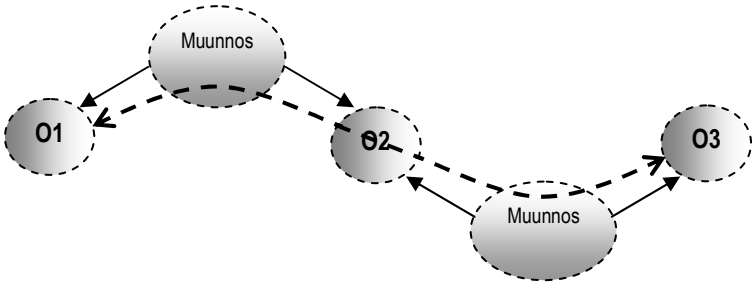
Kuvio 6. Ontologien välinen muunnos. (Kaykova ym. 2005a)

Maailmanlaajuinen ontologia olisi ylin sanasto, mikä määrittelisi minkä tahansa muun alemman ontologian kaikki käsitteet. Tietysti idea maailmanlaajuisesta ontologiasta on harhakuvaa, koska oikeassa elämässä pieninkin ongelma-alue voidaan kuvata useilla eri tavoilla. Kuitenkin maailmanlaajuisen ontologian konseptia voidaan käyttää rajatuille ongelma-alueille, kuten esimerkiksi teollisuudessa.

Vaikka maailmanlaajuinen ontologia olisikin käytössä jollain tarkoin rajatulla alueella, niin alaontologioiden heikkous voi estää niiden väliset keskinäiset muunnokset.

Toisaalta tällaisessa tapauksessa ontologioiden välinen semanttinen muunnos voitaisiin toteuttaa solmusta solmuun muunnoksena. Tällä tarkoitetaan sitä, että solmujen väliin tulisi välitysontologia, mikä määrittäisi solmujen välisen muunnoksen. Siihen olisi määritelty kummankin solmun käsitteet ja niitä vastaavat muunnossäännöt. Kuvio 7 esittää kolme

solmuontologiaa ja kaksi muunnosontologiaa. Semanttinen muunnos solmujen välillä voidaan tehdä suoraan, jos valmis muunnosontologia on olemassa. Tapauksessa missä suora muunnosontologia puuttuu, voitaisiin muunnos tehdä jaksottaisesti askel askeleelta lähtösolmusta välisolmujen kautta kohdesolmuun.



Kuvio 7. Solmusta solmuun muunnos. (Kaykova ym. 2005a)

3.4 RscDF

3.4.1 Tausta

Semanttisen webin alkuperäisenä ajatuksena oli, että normaalin sisällön lisäksi sivuilla olisi myös kone-luettavaa metadataa, mikä helpottaisi resurssien löytämistä ja automaattista prosessointia. Jotta ohjelmat pystyisivät ymmärtämään metadataa, pitää metadata kirjoittaa eksplisiittisessä muodossa. Tätä varten W3C kehitti RDF:n, mikä on perusmalli semanttisten kuvausten eli metadatan kirjoittamiseksi.

RDF:n perusajatuksena on, että ihminen luo web-resurssin ja on vastuussa resurssin kuvauksesta. Mikäli resurssia pitää muuttaa, ja muutos koskettaa myös resurssin kuvausta, päivittää ihminen RDF-kuvauksen samalla tarkoituksenmukaiseksi. Tämä on erittäin työlästä, jos resurssit ovat dynaamisia. Toisaalta automaattinen metadatan päivitys on erittäin hankalaa kun kyseessä ovat dynaamiset resurssit. Esimerkiksi jos web-resurssi generoituu automaattisesti jonkin alati muuttuvan tietokannan mukana, niin metadataa pitää päivittää jokaisen muutoksen jälkeen siten, että resurssin jokainen uusi tila tulisi otettua huomioon myös metadatassa. (Benjamins ym. 2002)

Hankalammaksi tilanne muuttuu kun kyseessä ovat teollisuuden resurssit, kuten esim. laitteet ja prosessit. Tämä johtuu siitä, että tämän tyyppiset resurssit ovat monesti

dynamiikaltaan luonnollisesti jatkuvia, jopa lyhyellä aikavälillä tarkasteltuna. Tällaiset resurssit voidaan kuitenkin linkittää webiin sovittimien ja sensoreiden avulla. Tällöin sensorit valvovat resurssien sisäistä tilaa ja sovittimet pitävät huolen metadatan automaattisesta päivityksestä. On selvää, että RDF ei ole kovinkaan käyttökelpoinen tekniikka tällaisia alati muuttuvia resursseja varten. (Kaykova, Khriyenko, Kononenko, Terziyan, Zharko, 2004)

RscDF on laajennus RDF:ään ja kehitetty vastaamaan teollisuuden resurssien dynamiikasta tuleviin haasteisiin. RscDF esittelee ontologian, mikä keskittyy resurssien ylläpitoon liittyvien tilojen kuvailemiseen. Siihen liittyy tieto resurssin tilasta ja siihen liittyvistä olosuhteista, tapahtuvat tilan muutokset, tieto resurssin tavoitetilasta ja historia resurssin edeltävistä tiloista. Resursseilla on oma tila-kuvauksensa esitettynä RscDF:llä, mitä voivat käyttää ulkopuoliset ohjelmistot, mitkä tukevat RscDF:ää. RscDF on tarkoitettu ratkaisemaan RDF:n ongelmat, mitkä koskevat resurssien erilaisuutta ja yhteensopivuutta. (Terziyan ym. 2005; Kaykova ym. 2005b)

3.4.2 RscDF:n käyttötarkoitus

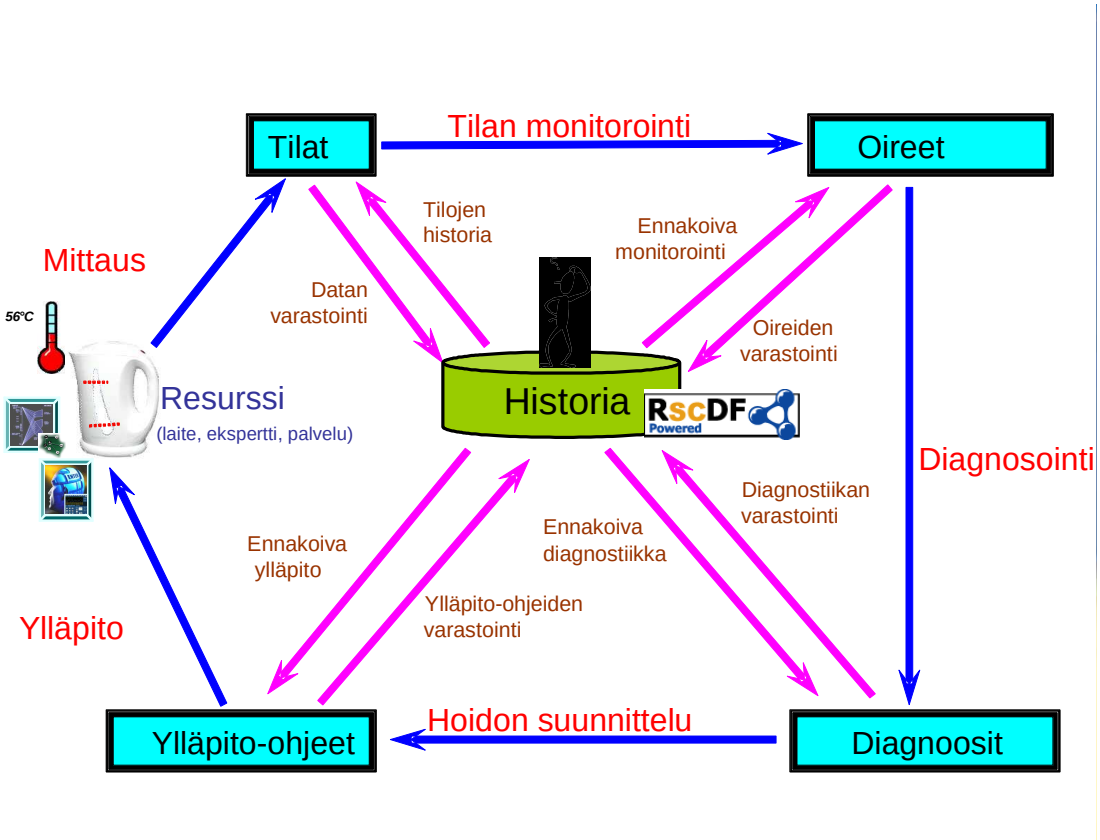
RscDF:n suunnittelun taustalla on, että erilaisten teollisuuden resurssien ylläpito olisi mahdollisimman yksinkertaista. SSDDM-malli (State-Symptom-Diagnosis-Decision-Maintenance) on luotu reaaliaikaiseen webin kautta tapahtuvaan olosuhteiden valvontaan ja ennustavaan ylläpitoon erilaisille teollisille resursseille.

SSDDM-malli sisältää kaikki tarpeelliset konseptit, niiden ominaisuudet sekä konseptien väliset suhteet tukeakseen ylläpitoresurssien kokonaisvaltaisen elämänkaaren. Malli sisältää neljä osaa (Kaykova ym. 2005b):

1. Havainnointi resurssin tilan hahmotuksesta, mikä tarkoittaa joukkoa parametrien arvoja tietyn ajan kuluessa tai niiden lukemista resurssin historiasta.
2. Ennakoinnin tekemistä tilasta eli hälytysjärjestelmä, mikä tekee ”oireiden” perusteella diagnoosin laitteesta. Tämä hälytysjärjestelmä olisi web-palvelu, mikä tekee diagnoosin resurssin tilasta oireiden perusteella.

3. Yksityiskohtainen tilan prosessointi mikä tekee lopullisen päätöksen diagnoosista. Tässä vaiheessa voi resurssin tilan selvittää ”tohtori”, joka antaa ohjeet kuinka tila saadaan takaisin normaaliksi. Nämä ohjeet ja oireet tallennetaan resurssin historiadataan.
4. Suorittamaan tarvittavat ylläpitotoimenpiteet, mutta kuitenkin pitää olla palvelu, mikä tekee ohjeen resurssille palauttaakseen sen taas normaaliin tilaan diagnoosin perusteella. Kaikki ylläpitotoimenpiteet täytyy tallentaa resurssin historiaan, sillä systeemin täytyy tietää sille tehdyt muutokset.

RscDF:n käyttö mahdollistaa agentti-pohjaisten alustojen rakentamisen jokaiselle teolliselle resurssille. Tällöin kaikki tieto liittyen resurssin monitorointiin ja diagnostiikkaan kerätään resurssin historiaan ja sitä hoidetaan resurssin agentilla. Kuvio 8 esittelee SSDDM-mallin elinkaaren. (Kaykova ym. 2005b)



Kuvio 8. SSDDM elinkaari.

3.4.3 RscDF:n aikalaajennus

Jotta RscDF pystyisi olemaan data-formaatti, mikä tukee kehittyneitä resurssien ylläpito tekniikoita, pitää sen pystyä tarjoamaan peruskäsitteet aikaan liittyvään metadataan. Ajallisesti muuttumattomat resurssien tilat eivät riitä tehokkaaseen resurssien ylläpitoon, sillä ylläpidon on otettava huomioon myös tilojen aikasuhteet. Joskus pelkästään tarkastelemalla resurssin tilaa ja vertailemalla sitä aikaisempiin tiloihin, voidaan saada tarpeeksi tietoa selville oikeaa diagnoosia varten. Yleisesti ottaen ajallinen diagnoosi on tärkeä alue ja sitä tarvitaan esim. lääketieteessä käyttävissä laitteissa. (Ryabov & Terziyan 2003)

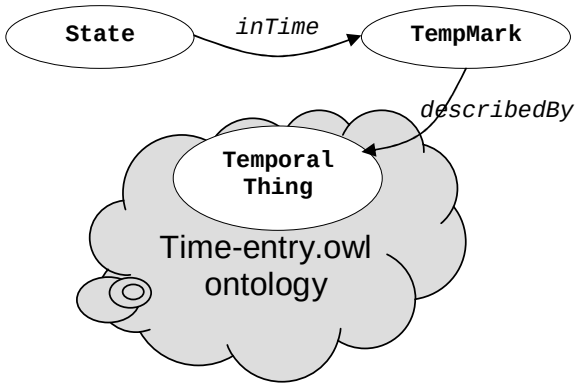
Myös muita yrityksiä standardoida aika-esitystä on olemassa. Kattavin yritys on DAML-Time -projekti tavoitteenaan kehittää kuvaava ontologia ajasta, mikä ilmaisisi aikakäsitteet ja ominaisuudet yleisesti kaikenlaisille ajan formalisoinneille. RscDF:ssä käytetään hyväksi joitain DAML-Time -projektin tuloksia. RscDF:n `rscdf:TempMark` on peritty DAML-Time -ontologian ylätason `TemporalEntity`-kasitteestä. Tämän avulla RscDF tulee yhteensopivaksi yleisesti hyväksyttyjen standardien kanssa. (DAML-Time; Terziyan ym. 2005)

Kolme tärkeintä RscDF:n aikalaajennuksen tuomaa asiaa ovat (Kaykova ym. 2005b):

- Aikamerkki (TempMark), mikä lisää aikamerkin mihin tahansa resurssin tilaan;
- Aikasuhde (Temporal relation), mikä määrittää ajallisen suhteen resurssin kahden eri tilan välillä;
- Aikajälki (Temporal trace), mitä käytetään resurssin peräkkäisten tilojen säilömiseen.

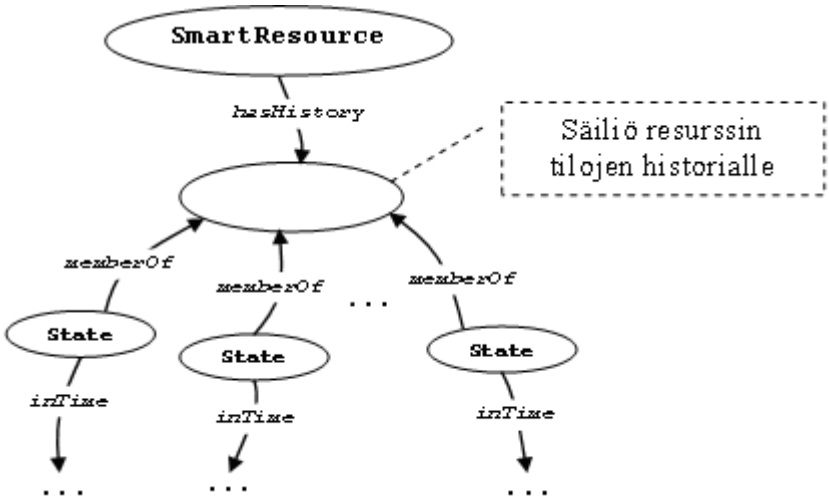
RscDF:ssä resurssia kuvataan merkinnällä `rscdf:State`, mihin ajalliset ominaisuudet lisätään yhdistämällä se ontologisen käsitteen `rscdf:TempMark` avulla. `rscdf:TempMark` on suoraan yhteydessä resurssin tilaan `rscdf:inTime`-ominaisuuden avulla. Koska `rscdf:TempMark` peritään DAML-Time-ontologiasta, niin ontologian ylimpänä luokkana toimii `time-entry:TemporalThing`. Tällöin

`rscdf:TempMark` on suoraan yhteydessä siihen ominaisuuden `rscdf:describedBy` avulla, kuten kuviossa 9 on esitetty.



Kuvio 9. RscDF:n aikamerkki. (Kaykova ym. 2005b)

Aikajäljen RscDF tuo ominaisuuden `rscdf:hasHistory`-avulla, mikä kuuluu `rscdf:SmartResource`:lle ja viittaa SmartResource:n tiloihin mitkä pystyvät lisääntymään pysyvästi resurssin eliniän aikana kuten kuvio 10 esittää. (Kaykova ym. 2005b)

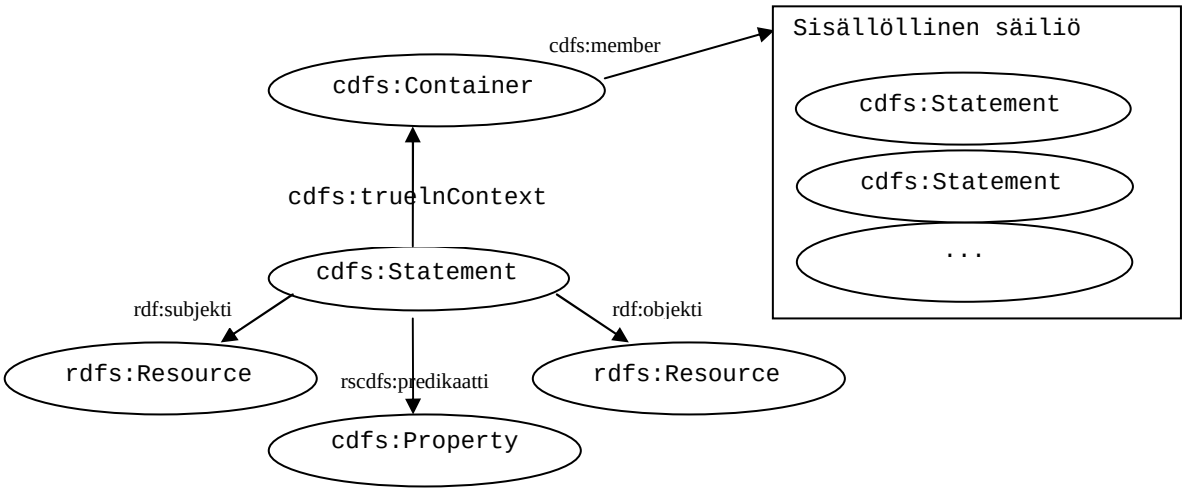


Kuvio 10. Aikajälki SmartResource:ssa.

3.4.4 Sisällöllinen laajennus

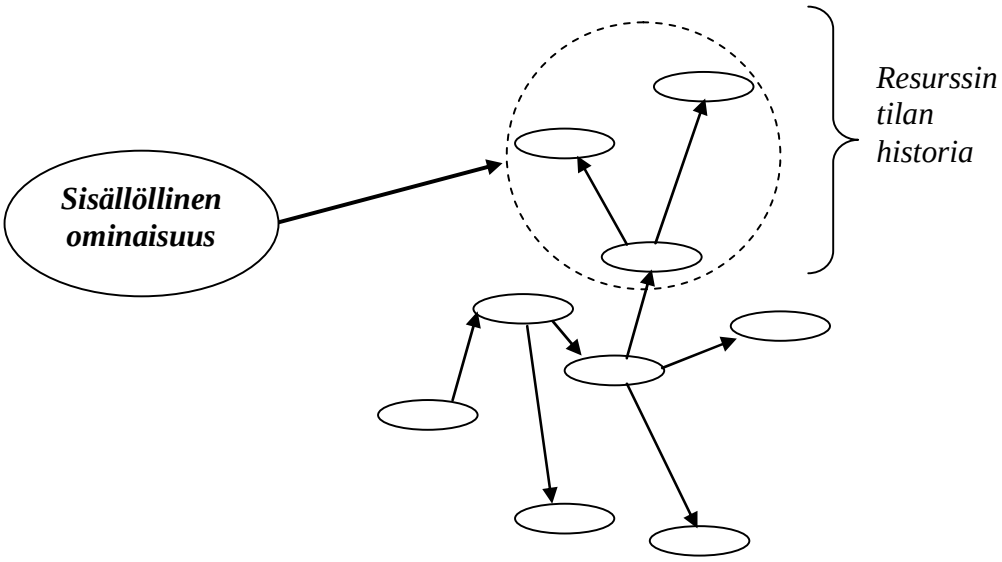
Normaalista RDF:stä on huomattavan hankalaa esittää ja tehdä kyselyjä väittämistä kontekstissaan, sillä se muodostuu kolmiosaisesta mallista. Tämän johdosta on esitelty joitain tapoja tämän ongelman ratkaisemiseksi. CDF:ää (Context Description Framework) on esitetty yhtenä vaihtoehtona ratkaisemaan sisällölliset parannustarpeet RscDF:ään. (Khriyenko & Terziyan 2005)

CDF:n vision mukaan jokaisella ominaisuudella on jotain tietoa viitekehyksestään. Jokainen väittämä voisi olla tosi tai epätosi koskiessaan erilaisia olosuhteita ympäristössään. Tässä tapauksessa väittämän sisältö olisi nippu muita väittämiä, mitkä kuvailisivat ympäristön tilan. Tällaiset kuvaukset voisivat sisältää myös tiedon väittämien kuvailujen lähteestä. Visiona on nelinkertainen malli, mikä esitetään kuviossa 11. Mallissa neljäs komponentti on säiliö kontekstuaalisille väittämille. (Khriyenko & Terziyan 2005)



Kuvio 11. Nelinkertainen visio väittämälle.

Sisällöllinen RscDF:n laajennus antaa mahdollisuuden lisätä resurssin ominaisuuksiin lisäominaisuuksia. Laajennus toimii kuin se olisi RDF:n alagraafi. Tämä mahdollistaa lisätä väittämiin tietoa resurssin historiasta kuten kuvio 12 esittää. (Kaykova 2005b)



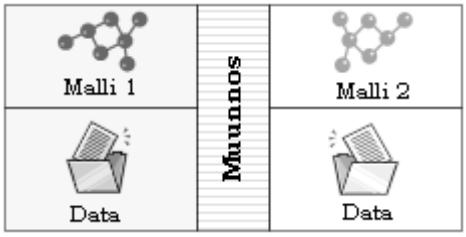
Kuvio 12. Sisällöllinen RscDF lisäys.

4 Semanttinen muunnos

Tässä luvussa esitellään aluksi semanttiseen muunnokseen liittyviä ongelmia. Tämän jälkeen esitellään SmartResource-projektissa kehitetty ratkaisu näiden ongelmien poistamiseksi.

4.1 Muunnosongelma

Kuten aikaisemmin on mainittu eräs pääongelmista GAF:n suunnittelussa ja toteuttamisessa on semanttinen muunnos. Semanttinen muunnos toteutetaan kahden eri standardin välillä, kuten kuviossa 13 esitetään, mitkä ovat tässä tapauksessa XML ja RscDF. Koska RscDF on RDF:n laajennus, palautuu muunnos ainakin osittain samaksi kuin muuttaisi XML-muotoista dataa RDF-muotoon.



Kuvio 13. Semanttinen muunnos.

Muunnosongelma voidaan jakaa karkeasti neljään eri kategoriaan (Sperberg-McQueen & Miller 2004):

1. Konkreettinen datan rakenteen kartoitusongelma, millä tarkoitetaan että halutaan määrittää millä tavalla joku ohjelmisto sisäisesti esittää XML-muotoista dataa. (Krupnikov & Thompson)

2. Abstrakti datan rakenteen kartoitusongelma määrittää miten muuttaa XML dokumentti sarakkeiksi, riveiksi ja taulukoiksi SQL-pohjaisissa tietokantajärjestelmissä. (Vorthman & Buck 2000a; Vorthman & Buck 200b)

3. FOPC-ongelma (First-Order Predicate Calculus) määrittelee miten tulkitaan merkkausta oikein. (Sperberg-McQueen ym. 2001)

4. RDF muunnosongelma, millä tarkoitetaan mielivaltaisen XML:n muuntamista RDF:ksi. (Hazaël-Massieux & Connolly 2005)

Nämä neljä ongelmaa voidaan yhdistää keskenään, jos kohdemallilla on yhteensopiva formalismi. Tällöin kaikki neljä edellä mainittua ongelmaa yhdistyvät, kun muunnetaan informaatiota yhdestä syntaksista (XML) johonkin toiseen syntaksiin (RDF). (Sperberg-McQueen & Miller 2004)

Sperberg-McQueen ja Miller ovat löytäneet vielä yhden ongelman lisää, mitä he kutsuvat *dokumentointiongelmaksi (documentation problem)*. Dokumentointiongelmalla he tarkoittavat, että XML skeema ei sisällä tarpeeksi ihmisen ymmärtävää informaatiota, jotta ihmiset osaisivat käyttää elementtejä ja attribuutteja oikein. Tämän ongelman tulisi olla hyvinkin helposti ratkaistavissa, eikä sen oikeastaan pitäisi olla ongelma lainkaan. Skeemasuunnittelijoiden tulisi kirjoittaa sanastosta selväkielinen selostus ja skeemaa käyttävien ihmisten tulisi osata tulkita ohjeet oikein.

Dokumentointiongelmaa ei kuitenkaan pystytä koskaan täysin poistamaan. Niin kuin hyvin tiedetään, on täydellisten ohjeiden kirjoittaminen miltei mahdotonta. Vaikka kuinka pyrittäisiin täydelliseen dokumentointiin jää aina jotain silti tulkinnanvaraiseksi, sillä sama teksti voidaan helposti tulkita usealla eri tavalla. Vielä hankalammaksi tilanne tulee silloin, kun skeeman suunnittelija ja käyttäjä tulevat täysin eri kulttuureista.

Myös useita sanastoja rakenteellisten kielien tulkitsemiseen on kehitetty ja käytetty jo vuosien ajan. Tällaisten sanastojen käyttäminen tekee rakenteellisten kielien käyttämisen kätevämmäksi ja helpommaksi, mutta ne eivät pysty tekemään siitä mekaanista. Tämä aiheuttaa suurimman ongelman, mikä liittyy semanttisen muunnoksen suunnitteluun.

Muunnosongelmaa voidaan lähestyä myös toiseltakin kantilta. Swick & Henry (1999) katsovat, että muunnosongelma ja sen ratkaisu on ainoastaan uuden kielen suunnitteluun ja käyttöön liittyvä ongelma. He katsovat, että XML-skeeman `xsd:annotation`–elementin sisälle suunniteltaisiin uusi kieli kuvaamaan muunnoksia XML:stä johonkin toiseen syntaksiin. `xsd:annotation`–elementti mahdollistaa skeeman dokumentoinnin.

Sperberg-McQueen ja Miller tuovat kuitenkin kaksi asiaa esille edellä mainittua ratkaisua vastaan:

- Koska muunnosongelmat voidaan pelkistää ongelmaan muuntaa informaatiota yhdestä syntaktista toiseen, niin ei ole välttämätöntä kehittää uutta kieltä, sillä jo nykyäänkin olevia voidaan käyttää hyödyksi.
- Ja vaikka olemassa olevat kielet ovat monisanaisempia, yleisempiä ja turhan monimutkaisia, niin muunnoskielen tulisi perustua oikeiden muunnosten analyysiin.

Tällä hetkellä käyttökelpoisin ja useimmiten käytetyin tapa muuntaa XML-muotoista dataa toiseen syntaksiin, on käyttää XSLT-muunnoksia. XSLT:n avulla voidaan XML-muotoista dataa muuntaa suoraviivaisesti RDF:ksi. (Ogbuji 2001)

XSLT:n käyttö yleisissä tapauksissa kuitenkin tuottaa Sperberg-McQueenin ja Millerin esittelemän kategoria 4:ään kuuluvan ongelman, sillä XML-tiedostot voivat olla täysin mielivaltaisia. Jos XSLT-muunnos tehtäisiin tuottamaan RDF:ää jo alussa esitellystä yliopiston kotisivuja kuvaavasta dokumentista, mutta tämän jälkeen tiedoston rakennetta muutettaisiin, niin lopputulos olisi täysin erilainen. Mikäli `<Title>` muutettaisiin vaikka muotoon `<Otsikko>`, niin XSLT-muunnos olisi kirjoitettava uudelleen. Myöskään XSLT-muunnos ei sovellu suoraan XML:n muuntamiseen RscDF:ksi, sillä XSLT:n avulla ei pystytä ottamaan huomioon RscDF:n toiminnallisuutta.

Edellä mainittu kotisivuja kuvaava XML-tiedosto oli hyvin lyhyt ja yksinkertainen. Normaalisti XML-tiedostot ja niiden skeemat ovat paljon monimutkaisempia ja laajempia. Tällöin myös metadatan erottamiseksi kirjoitettava XSLT-tiedosto on huomattavasti monimutkaisempi. Tällaisten XSLT-tiedostojen tekeminen vaatii huomattavan määrän tietoa XML:stä, sen skeemasta ja RDF:stä.

Kun XML:ää käytetään kuvailemaan yleisesti internetsivuilla olevia resursseja, kuten *title, author, copyright* jne., niin ne kirjoitetaan pääsääntöisesti samalla tavalla. Tällaisissa tapauksissa XSLT-muunnos on erittäin käytännöllinen tapa tehdä suora semanttinen muunnos. Toisaalta myös muunnettaessa XML-tiedosto toiseksi XML-tiedostoksi on XSLT-muunnos käytännöllinen.

Kun tämän tapaisia muunnoksia tekevät sanastojen suunnittelijat tai skeemojen tekijät on selvää, että he pystyvät tekemään muunnokset käyttämällä XSLT-muunnoksia hyväkseen. Mutta vaikka XSLT on tehokas kieli XML:n muuttamiseen, tarvitaan muitakin keinoja parantamaan skeemojen dokumentointia, minkä avulla muunnoksia pystytään helpottamaan. Dokumentointia tarvitaan varsinkin jos dataa pitää prosessoida muuntamisen ohella.

Tällä hetkellä tutkimusta XML:n muuntamiseksi RDF:ksi yleisissä tapauksissa ei ole tarpeeksi tehty, eikä ole olemassa valmiita ratkaisuja ongelmien poistamiseksi. (Sperberg-McQueen & Miller 2004)

4.2 Kaksiosainen muunnos

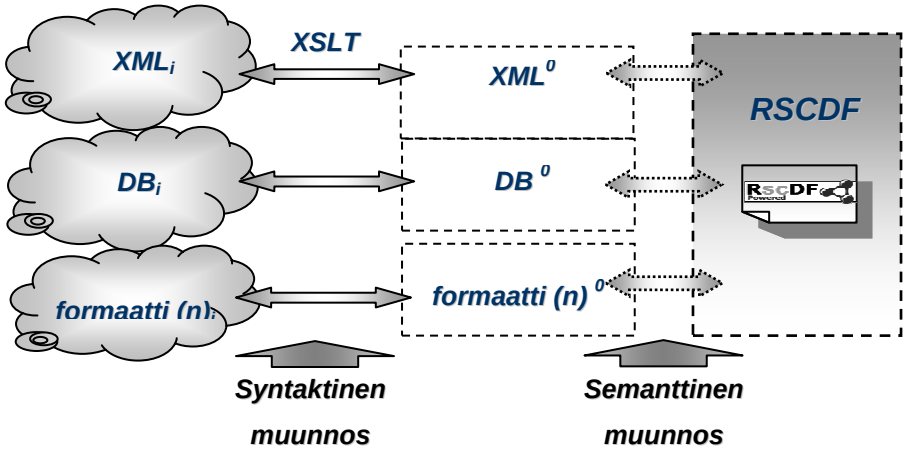
Aikaisemmin on esitetty useita haasteita, mitkä liittyvät XML-muotoisen datan muuntamiseen RDF-muotoon. Yksittäisissä muunnoksissa itse perusmuunnoshan on erittäin helposti toteutettavissa XSLT-muunnoksen avulla. Mutta kuten aikaisemmin on todettu, tämä tapa ei sovellettavissa kaikissa tilanteissa. Varsinkaan kun XML-muotoista dataa muunnetaan RscDF-muotoon, jolloin joudutaan ottamaan huomioon RscDF:n toiminnallisuus, eli resurssien dynaamisuus. Toisaalta RscDF sisältää myös tiedon resurssin tilasta ja siihen liittyvistä olosuhteista, tapahtuvat tilan muutokset, tiedon resurssin tavoitetilasta ja historian resurssin edeltävistä tiloista. Tällöin ei XSLT yksistään anna riittäviä työkaluja muunnoksen toteuttamiseksi. Ratkaisuna tähän ongelmaan on *kaksiosainen muunnos*.

Kaksiosaisella muunnoksella tarkoitetaan sitä, että koko muunnosprosessi jaetaan kahteen täysin itsenäiseen osaan:

- syntaktiseen muunnokseen ja
- kanoniseen semanttiseen muunnokseen.

Idea kaksiosaisen muunnoksen taustalla on se, että kun on uniikki RDF-muotoinen rakenne, kuten esim. RscDF tai Dublin Core, niin silloin pystytään myös rakentamaan sitä

vastaava uniikki XML-muotoinen rakenne XML^0, kuten kuviossa 14 esitetään. Tämä XML^0-rakenne on vain apua tuova välimuoto muunnosprosessissa. Tällaisen rakenteen avulla päästään yksinkertaisesti eroon XML:n rakenteen tuomista ongelmista, mitkä on mainittu aiemmin.



Kuvio 14. Kaksiosainen muunnos.

XML^0-rakenne pitää sisällään kaiken tarvittavan tiedon, mitä sitä vastaava RDF-rakenne vaatii. Lisäksi rakenne pitää sisällään myös kaiken tarpeellisen alkuperäisestä XML-dokumentista tarvittavan tiedon, kuten esimerkissä 3 esitetään. XML^0-rakenteen suunnittelijoiden tuleekin olla hyvin perillä ongelma-alueesta.

Esimerkki 3. Rakenteiden vertailu.

Kirjailija	Tekijä	dc:creator
Otsake	Nimeke	dc:title
XML	XML^0	RDF

Muunnoksen jako kahteen erilliseen osaan myös helpottaa koko muunnosprosessia ja tekee siitä yksinkertaisen käyttää. Tällöin kumpaakin muunnosta voidaan kehittää eteenpäin erikseen ja tekniikka muunnoksen toteuttamisessa on vapaasti valittavissa.

Tämä lähestymistapa ei kuitenkaan ole käyttökelpoinen kaikkien erilaisten datamallien ollessa kyseessä. Kaksiosainen muunnos on mahdollista kuitenkin niissä tapauksissa, missä työkalut syntaktisen muunnokseen tekemiseen ovat olemassa, kuten XML:ssä on XSLT.

Kaksiosaisen muunnoksen ensimmäisessä syntaktisessa osassa muutettava XML-rakenne muunnetaan XSLT:n avulla XML^0-muotoon. Tämä tapa mahdollistaa erilaisten XML-muotoisten rakenteiden muuntamisen yksinkertaisella muunnoksella, mihin on myös kaupallisia työkaluja olemassa. Tällaisesta kaupallisesta työkalusta voidaan esimerkkinä mainita Altova MapForce, mikä ottaa muunnettavien XML-dokumenttien skeemat ja tekee muunnoksen skeemojen pohjalta. Kuitenkin tämäkin työkalu tarvitsee käyttäjän yhdistämään solmut keskenään, sekä tekemään myös käsin muutoksia muunnokseen jo yksinkertaisten skeemojen ollessa kyseessä.

Lisäksi XSLT mahdollistaa muunnokset XML:n ja relaatiotietokantojen välillä. Tällaisissa tapauksissa on mahdollista muuntaa relaatiotietokannat suoraan XML^0-muotoon mistä edelleen RDF:ksi.

Datan muuntamisen ja muokkaamisen mahdollisuus muunnoksen aikana on erittäin tärkeää, mikäli pyritään saamaan aikaan yleismaailmallinen sovellus. Maailmassa on myös olemassa useita erilaisia kieliä sekä tapoja ilmaista mittoja, kuten esim. SI-järjestelmä ja anglosaksinen järjestelmä. Tämä aiheuttaa useita yhteensopivuusongelmia eri järjestelmien kesken ja mahdollistaa sekaannukset eri käyttäjien välillä. Kaksiosainen muunnos on myös ratkaisu tähän ongelmaan. Syntaktisen muunnoksen vaiheessa voidaan tarvittaessa esim. mitat muuntaa järjestelmästä toiseen, jolloin järjestelmät tulevat yhteensopiviksi keskenään, eikä loppukäyttäjän tarvitse edes tietää muutoksesta.

Kanonisessa semanttisessa muunnoksessa XML^0-muotoinen data muunnetaan vastaavaksi RDF-muotoiseksi dataksi. Tässä muunnoksessa käytettävä tekniikka riippuu muunnoksen monimutkaisuudesta. Mikäli muunnetaan Dublin Core -muotoon, niin muunnoksessa on

helppo käyttää suoraan XSLT:ä. Mutta esim. RscDF-muotoon muunnettaessa tulee ottaa huomioon RscDF:n toiminnallisuus, joten siihen XSLT ei suoraan sovellu.

Tällä hetkellä ei ole työkaluja suoran semanttisen muunnoksen tekemiseksi XML:n ja RDF:n välille. Kaupallisia työkaluja ei ole jotka pystyisivät, ainakaan tällä hetkellä, käsittelemään RDF-skeemoja sillä tavalla, että pystyttäisiin tekemään suoraan XSLT-muunnos XML:stä RDF:ään. Tästä syystä suorien muunnosten tekeminen käsin on haastavaa ja erittäin työlästä laajojen XML-skeemojen ollessa kyseessä. Tästä johtuen on huomattavasti yksinkertaisempaa tehdä yksi muunnos, mikä mahdollistaa kaikkien XML-rakenteiden liittämisen järjestelmään käyttämällä XML^0-rakennetta välissä. Tämän välimuodon hyväksikäyttäminen johtuu vain siitä, että XML-rakenteiden väliset muunnokset ovat nykypäivän tekniikoita käyttämällä yksinkertaisempia toteuttaa.

Jotta muunnos olisi täydellinen, tulisi kaksiosaisen muunnoksen toimia myös takaisinpäin RscDF:stä XML:ksi. Tämä on myös tarpeellista toteuttaa. On olemassa joitain projekteja, joilla on yrityksiä RDF-muotoisen datan muuntamiseksi takaisin XML-muotoon. Koska RscDF on RDF:n laajennus, niin oletuksena on, että näitä metodeja voidaan hyödyntää myös tässä tapauksessa. (Sperberg-McQueen & Miller 2004; XR)

5 XSLT-muunnokset

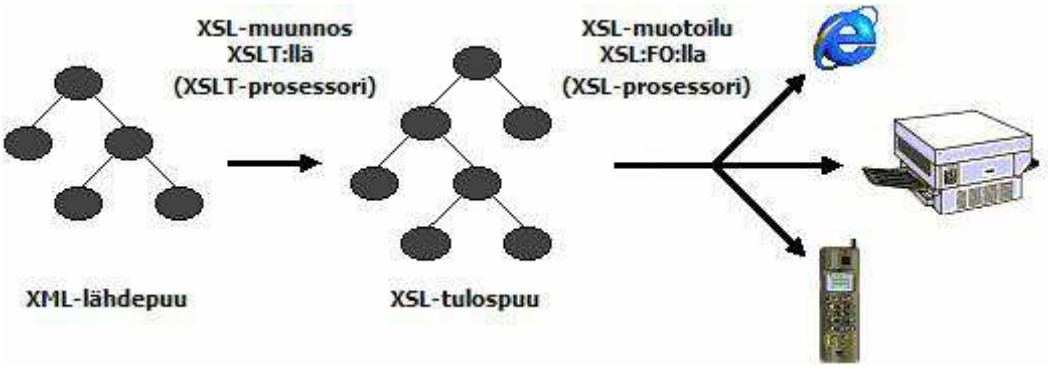
Tässä luvussa esitellään XSLT-muunnokset keskittyen varsinkin muunnoksen kahteen tutkielman kannalta tärkeimpään kieleen XSLT ja XPath. Ensin esitellään XSL ja sen jälkeen XSLT ja XPath. Lopuksi esitetään muutamia esimerkkejä XSLT:n ja XPath:n käytöstä.

XSLT:stä puhuttaessa päälähteenä on käytetty XSL Transformation (XSLT) Version 1.0 -spesifikaatiota (Clark 1999) ja XPath:n kohdalla päälähteenä on ollut XML Path Language (XPath) Version 1.0 (Clark & DeRose 1999), joten lähdeviitteet on merkitty vain näistä poikkeavista lähteistä.

5.1 XSL

Ennen kuin W3C sai ensimmäisen XML 1.0 suosituksen valmiiksi, niin syksyllä 1997 W3C sai ensimmäiset moitteet siitä, ettei ollut mitään tapaa kuinka XML-dokumenttien sisältö tulisi esittää. Tähän liittyen piti kehittää kieli, mikä mahdollistaisi XML-dokumenttien esittämisen, tyylin ja ulkoasun sekä sivutuksen esim. Web-selaimessa, taskutietokoneessa tai vaikkapa kirjassa. Alkuperäistä esitystä kielestä kutsuttiin nimeltä XSL (Extensible Stylesheet Language). W3C aloitti kielen kehittämisen 1998 ja se saavutti suosituksen aseman lokakuussa 2001. (Simpson 2001)

Kun XSL projekti julkaistiin, niin XSL-kielen uskottiin koostuvan kahdesta eri osiosta kuten kuvio 15 osoittaa. Ensimmäinen on nimeltään *tree formatting* (puumuunnos) ja toinen *formatting* (muotoilu). Puumuunnos mahdollistaa lähdepuun rakenteen olevan huomattavasti erilainen kuin kohdepuun rakenteen. Tämä antaa mahdollisuuden uudelleen järjestää lähdedataa tai suodattaa osia lähteestä pois kohdepuusta. Toinen osio antaa hienomman säätövaran kohdepuun elementtien ja attribuuttien ulkoasun esittämiselle. (Adler ym. 2001)



Kuvio 15. XSL prosessi. (Adler ym. 2001)

XSL-kielen kehittelyn alkuvaiheessa kuitenkin huomattiin, että XSL:n täytyy jakaantua erillisiin alakieliin. Nämä alakielet kuitenkin kehittyivät eri tahtia ja niiden väliset suhteet muuttuivat nopeasti. Tänä päivänä XSL koostu kolmesta eri kielestä (Nachimovsky & Myers 1999):

1. XSLT on XML-dokumenttien muunnoskieli.
2. XPath on kieli XML-dokumenteissa navigoimiseen.
3. XSL-FO kieltä käytetään XML-dokumentin ulkoasun kuvaamiseen.

Yllämainitut XSL:n kolme osaa ovat täysin itsenäisiä kieliä. Kuitenkaan ei pidä unohtaa, että XSLT käyttää XPath:ia XML-dokumentissa navigoimiseen. Tästä johtuen usein XML-dokumentteja muunnettaessa puhutaan käytettävän ainoastaan XSLT:ä, vaikka pitäisi muistaa myös XPath:n olevan mukana, sillä ne ovat kuitenkin aivan erillisiä kieliä (Nachimovsky & Myers 1999).

5.2 XSLT

5.2.1 Johdanto

XSLT on XML-pohjainen kieli, minkä tarkoituksena on muuntaa XML-dokumentit toiseen muotoon. XSLT 1.0 julkistettiin marraskuussa 1999 ja XSLT 2.0 tammikuussa 2007. (Kay 2007)

XSLT kieli esitetään kuten hyvin muotoiltu XML-dokumentti, mikä vain käyttää XSLT-nimiavaruutta. XSLT-tyylisivu kuvaa säännöt, joiden avulla XSLT prosessori muuntaa lähdepuun kohdepuuksi. Elementit ja attribuutit lähdepuusta voidaan kopioida tai suodattaa ja sen lisäksi ylimääräisiä elementtejä ja attribuutteja voidaan lisätä kohdepuuhun. Tämä mahdollistaa kohdepuun rakenteen olevan huomattavasti erilaisempi kuin lähdepuun.

XSLT suunniteltiin XSL:n osaksi, mutta se pystyy toimimaan myös täysin itsenäisesti. XSLT käyttää XPath kieltä navigoimiseen XML-dokumentissa.

XSLT-tyylisivu koostuu joukosta määrityksiä ja mallinteita (template). Määrityksillä vaikutetaan tyylisivun tuottamaan dokumenttiin liittyviä asioita tai määritellään muuttujia myöhempää tarkoitusta varten. Mallineet voivat sisältävät dokumentin muuntamisessa käytettäviä sääntöjä, muutettavan dokumentin käsittelyohjeita ja tuotetun dokumentin elementtien rakenteeseen liittyviä sääntöjä.

XSLT-tyylisivu koostuu mallinesäännöistä, mitkä jakaantuvat kahteen osaan:

- lausekkeeseen, mitä verrataan lähdepuun solmuihin ja laukaisee mallineen toiminnan sekä
- mallineen sisällöstä.

Mallineen sisältö määrää tuotettavan dokumentin rakenteen ja sisällön. Se voi mm. tehdä uusia elementtejä ja attribuutteja. Malline voi myös valita sisään luetun dokumentin elementtejä ja attribuutteja ja käsitellä niitä itse tai kutsua uusia mallinteita elementtien pohjalta.

XSLT-prosessori käsittelee aina yhtä XML-dokumentin solmua kerrallaan ja tätä solmua kutsutaan *kontekstisolmuksi* (current node/context node). Tämän solmun käsittelyyn valitaan XSLT-tyylisivun mallineista se, minkä laukaiseva sääntö osuu siihen tarkimmin. Tämä mahdollistaa tyylipohjan olevan käyttökelpoinen useille dokumenteille, joilla on samantyyppinen lähdepuun rakenne.

XSLT-prosessori sisältää erillisen algoritmin, mikä määrittää sääntöjen valinta järjestyksen. Vain yksi mallinne valitaan kerralla ja tämä mallinne ohjaa toimintaa ja käskee XSLT-prosessoria jossakin vaiheessa ottamaan uuden elementin tai muun XML-dokumentin osasen nykyiseksi solmuksi. Tämän jälkeen XSLT-prosessori etsii uuden mallineen. Prosessori aloittaa aina käsittelyn sisään tulevan dokumentin juurielementistä, mihin kontekstisolmu viittaa käsittelyn alussa, ja tätä käsittelyä jatketaan edellä kuvatulla tavalla siihen saakka kunnes kaikki solmut on läpikäyty.

Yleisimmin XSLT:ä käytetään muunnettaessa XML-dokumentteja sellaiseen muotoon, että web-selaimet tunnistaisivat ne ja pystyisivät näyttämään dokumentit selkeällä tavalla käyttäjälle. Tämä tapahtuu muuntamalla XML-dokumentti joko HTML-muotoon tai XHTML (eXtensible HyperText Markup Language) -muotoon. Tämä tapahtuu muuntamalla jokainen XML-elementti joko HTML tai XHTML -elementiksi.

5.2.2 Tyylisivun rakenne

Kun tarkastellaan XSLT-tyylisivun rakennetta, niin se on kuin XML-dokumentti ja sen täytyy alkaa XML julistuksella `<?xml version=”1.0”?>`. Suurin ero näiden välillä on se että tyylisivu käyttää elementtejä, eli komentoja, mitkä löytyvät XSLT:n nimiavaruudesta, minkä URI on `http://www.w3.org/1999/XSL/Transform`.

Normaalisti XSLT:n nimiavaruuteen viitataan etuliitteellä `xsl:`. Tämä ei kuitenkaan ole pakollista, koska tyylisivut voivat käyttää mitä tahansa etuliitettä. Toisen etuliitteen käyttäminen kuitenkin edellyttää, että nimiavaruuden julistuksessa etuliite sidotaan käyttämään XSLT:n nimiavaruuden URI:a.

Juurielementti mikä julistaa dokumentin olevan XSL tyylisivu, on `<xsl:stylesheet>` tai `<xsl:transform>`. Ne ovat täysin synonyymejä ja kumpaakin voidaan käyttää. Ainoat tavat julkistaa tyylisivu esitetään esimerkissä 4. XSLT:n versio on myös julistettava tässä vaiheessa ja se tapahtuu attribuutilla `version`.

Esimerkki 4. Oikea tapa julistaa tyylisivu.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Juurielementin `<xsl:stylesheet>`-lapsielementtejä kutsutaan ylätason elementeiksi. Taulukossa 1 esitellään ylätason elementit, mitä `<xsl:stylesheet>` voi sisältää.

Elementti	Kuvaus
`xsl:import`	Lataa mallineet ulkoisesta tyylisivusta.
`xsl:include`	Sisällyttää mallineet ulkopuolisesta lähteestä kuten ne olisivat osa tuovaa dokumenttia.
`xsl:strip-space`	Mahdollistaa määrittää minkä elementtien välilyönnit poistetaan.
`xsl:preserve-space`	Mahdollistaa määrittää minkä elementtien välilyönnit säilytetään.
`xsl:output`	Määrittää ulos tulevan dokumentin formaatin.
`xsl:key`	Julistaa nimetyn avaimen mitä voidaan käyttää tyylisivulla.

`xsl:decimal-format`	Määrittää mitä merkkejä ja symboleita käytetään muunnettaessa numeroita merkkijonoiksi `format-number()`-funktiolla.
`xsl:namespace-alias`	Korvaa tyylisivun nimiavaruuden toiseksi nimiavaruudeksi ulos tulevaan dokumenttiin
`xsl:attribute-set`	Määrittää nimetyn attribuuttijoukon.
`xsl:variable`	Julistaa lokaalin tai globaalin muuttujan.
`xsl:param`	Julistaa lokaalin tai globaalin parametrin.
`xsl:template`	Määrää muunnoksessa käytettävät säännöt tietyssä solmussa.

Taulukko 1. Tyylisivun ylätason elementit.

Ylätason elementtien järjestys tyylisivulla ei ole `xsl:import`-elementtejä lukuun ottamatta tärkeä. Elementit voivat olla missä järjestyksessä tahansa, kunhan `xsl:import`-elementit ovat ensimmäisinä. Esimerkki 5 esittää tyylisivun rakenteen ja jokaisen ylätason elementeistä. Paikat, mistä attribuuttien arvot ja sisältö on poistettu, esitetään (`...`) merkinnällä.

Esimerkki 5. Tyylisivun rakenne.

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="..."/>
  <xsl:include href="..."/>
  <xsl:strip-space elements="..."/>
  <xsl:preserve-space elements="..."/>
  <xsl:output method="..."/>
  <xsl:key name="..." match="..." use="..."/>
```

```
  <xsl:decimal-format name="..."/>
  <xsl:namespace-alias stylesheet-prefix="..."
    result-prefix="..."/>
  <xsl:attribute-set name="..."> ... </xsl:attribute-set>
  <xsl:variable name="..."> ... </xsl:variable>
  <xsl:param name="..."> ... </xsl:param>
  <xsl:template match="..."> ... </xsl:template>
  <xsl:template name="..."> ... </xsl:template>
</xsl:stylesheet>
```

Koska XML:n rakenne vaatii jokaiselle elementille myös lopetuksen, täytyy XSL tyylisivu lopettaa `</xsl:stylesheet>`-elementillä. Tämä elementin lopetusehto koskee myös muitakin elementtejä. Se voidaan toteuttaa kuten tyylisivun lopettamisessa tai lisäämällä elementin sisällön perään `/>`, jolloin kuten edellisestä esimerkistä huomataan, saadaan syntaksia lyhennettyä ja tyylisivua selvemmäksi.

5.2.3 XSLT:n elementit ja funktiot

W3C:n suositus XSLT:lle sisältää myös muitakin elementtejä kuin ylätason elementit, mitkä esitellään taulukossa 2. Näiden elementtien avulla voidaan suoraan luoda tulospuuhun elementtejä ja attribuutteja, tehdä kommentteja, lajitella solmuja jne.

Elementti	Kuvaus
`xsl:apply-imports`	Ottaa mallineen säännöt käyttöön tuodusta tyylisivusta.
`xsl:apply-templates`	Soveltaa mallineen sääntöä nykyiseen elementtiin tai sen lapsiin.
`xsl:attribute`	Generoi attribuutin kohde dokumenttiin.

`xsl:call-template`	Kutsuu nimetyn mallineen.
`xsl:choose`	Käytetään choose/when/otherwise -rakenteiden kanssa.
`xsl:comment`	Luo kommenttisolmun.
`xsl:copy`	Kopioi kontekstisolmun, mutta ei lapsia tai attribuutteja.
`xsl:copy-of`	Kopioi kontekstisolmun ja sen lapset ja attribuutit.
`xsl:element`	Luo elementtisolmun.
`xsl:fallback`	Määrittää toiminnan, mikä suoritetaan, ellei XSLT-prosessori tue elementin vanhemman komentoa.
`xsl:for-each`	Käy läpi jokaisen solmun määritellystä solmujoukosta.
`xsl:if`	Suorittaa sisältävän elementin vain jos testi on tosi.
`xsl:message`	Kirjoittaa varoitus- ja virheviestit.
`xsl:number`	Määrittää integer-muotoisten numeroiden esitystavan muodon sekä esitystapaan liittyvien pilkkujen paikat.
`xsl:otherwise`	Käytetään choose/when/otherwise -rakenteiden kanssa.
`xsl:processing-instructions`	Luo prosessointiohjeen ulostuloon.
`xsl:sort`	Lajittelee ulostulon.
`xsl:text`	Kirjoittaa tekstiä.
`xsl:value-of`	Poimii valitun solmun arvon.
`xsl:when`	Käytetään choose/when/otherwise -rakenteiden kanssa.

`xsl:with-param`	Määrittää parametrin arvon.

Taulukko 2. XSLT:n elementit.

XSLT sisältää myös joitain sisäänrakennettuja funktioita, mitkä esitellään taulukossa 3. Saatavilla on näiden lisäksi myös yli 1000 funktiota lisää, mitkä löytyvät omasta nimiavaruudestaan. Näiden funktioiden oletusetuliitteenä on `fn`, ja nimiavaruuden URI on <`http://www.w3.org/2005/xpath-functions`>. Näitä funktioita on mm. merkkijonojen käsittelyyn, numeroiden käsittelyyn ja ajan ja päivämäärän vertailuun. (Malhotra A., Melton J. & Walsh N. 2007)

Functio	Kuvaus
`current()`	Palauttaa nykyisen solmun.
`document()`	Antaa mahdollisuuden käyttää ulkopuolisen dokumentin solmuja.
`element-available()`	Testaa tukeeko XSLT-prosessori määritettyä elementtiä.
`format-number()`	Muuntaa numeron merkkijonoksi.
`function-available()`	Testaa tukeeko XSLT-prosessori määritettyä funktiota.
`generate-id()`	Generoi yksilöllisen id-arvon solmulle.
`key()`	Palauttaa solmujoukon käyttämällä indeksiä mikä on määritetty <`xsl:key`>-elementillä.
`system-property()`	Palauttaa järjestelmän ominaisuuksien arvot.

`unparsed-entity-uri()`	Palauttaa ei jäsennettyjen yksilöiden URI:n.

Taulukko 3. XSLT:n funktiot.

5.3 XPath

5.3.1 Johdanto

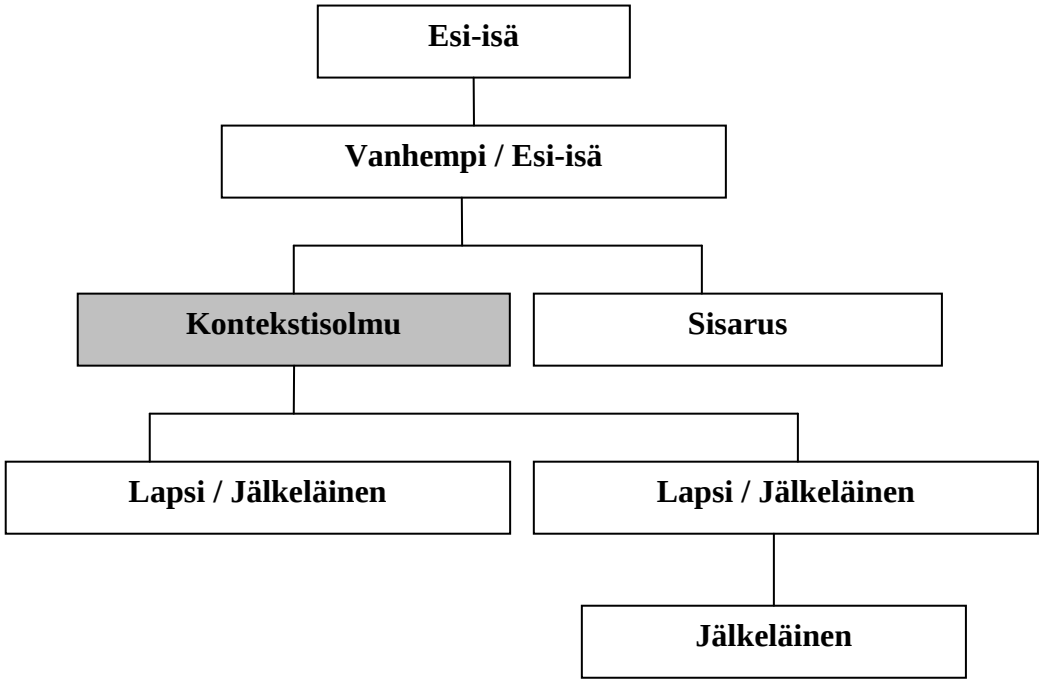
XPath versio 1.0 on ollut W3C:n suositus marraskuulta 1999 ja XPath 2.0 (Berglund ym. 2007) tammikuusta 2007. XPath:n päätarkoituksena on XML-dokumentin tiettyjen osien paikantaminen ja valinta. Sen avulla pystytään paikallistamaan esim. kaikki tietyn tyyppiset elementit, kaikki tiettyä elementtiä edeltävät tai seuraavat elementit. Toisena tarkoituksena oli taata perustarpeet merkkijonojen, numeroiden ja boolean käsittelyyn. XPath on suunniteltu käytettäväksi yksinään, mutta se on tärkeässä osassa tehtäessä XSLT-muunnoksia. Myös monet muutkin W3C:n teknologiat käyttävät XPath:ia kuten XPointer/XLink ja XQuery. (DeRose ym. 2001; DeRose ym. 2002; Boag ym. 2007)

XPath käyttää kompaktia ei-XML syntaksia tehdäkseen kanssakäymisen URI-viittausten ja XML:n attribuuttien arvojen kanssa helpommaksi. XPath operoi abstraktilla XML-dokumentin loogisella rakenteella, kuin itse syntaksi tasolla. XPath saa nimensä käyttämällä polkuesitysmuotoa navigoidessaan XML-dokumentissa.

XPath mallintaa XML-dokumentin puuksi, mikä koostuu solmuista. XPath:ssa on seitsemän erityyppistä solmua: elementti, attribuutti, teksti, nimiavaruus, prosessointiohje, kommentti ja dokumentin juurisolmu. XPath:ssa on määritelty tapa saada string-arvo jokaisesta eri solmutyypistä. Joillekin solmutyypeille string-arvo on osa solmua ja lopuille string-arvo saadaan laskemalla alenevien solmujen string-arvo. Muita mahdollisia arvotyyppejä XPath:ssa ovat boolean, solmujoukko sekä numero, mikä esitetään liukulukuna.

XPath määrittelee solmujen väliset suhteet ja navigoi puussa niiden avulla kuten kuvio 16 esittää. Jokaisella elementillä ja attribuutilla täytyy olla yksi vanhempi. Jokaisella

elementillä voi olla nolla, yksi tai useampia lapsia. Sisaruksiksi kutsutaan solmuja, joilla on sama vanhempi, esi-isäksi solmun vanhempia ja vanhempien vanhempia jne. ja jälkeläisiksi solmun lapsia ja lapsenlapsia jne.



Kuvio 16. Solmujen väliset suhteet.

5.3.2 XPath:n syntaksi

Kun navigoidaan puussa mikä koostuu solmuista, niin sijainti puussa on tärkeä tieto. XPath kyselyä ei automaattisesti tehdä koko puuhun, mutta kyselyllä on aina lähtöpiste. Lähtöpisteestä on helppo tehdä kyselyitä kuten esim. ”anna minulle lapsesi” tai ”anna minulle sisaruksesi”. Tällaisissa kyselyissähän on järkeä vain silloin, kun lähtösolmu on tiedossa. Kyselyn lähtösolmu voi olla myös dokumentin juurisolmu, jolloin kysely käy koko puun läpi.

XPath käyttää *saantipolkua* (path expression) solmun tai solmujoukon valintaan. Saantipolku voi koostua yhdestä tai useammasta askeleesta. Saantipolku voi olla *absoluuttine*n tai *suhteellinen*. Absoluuttinen saantipolku alkaa ”/”-merkillä, jolloin

yksinäinen ”/”-merkki viittaa dokumentin juurisolmuun ja tämän jälkeen sitä voi seurata suhteellinen saantipolku. Suhteellisessa tapauksessa lähtösolmuna on kontekstisolmu ja se voi sisältää useita askeleita, mitkä erotetaan toisistaan ”/”-merkillä. Uudeksi kontekstisolmuksi tulee aina saantipolulla saavutettu solmu. Esimerkki 6 esittää saantipolun lausekkeen.

Esimerkki 6. Saantipolun lausekkeen muoto.

```
Absoluuttinen saantipolku: /askel/askel/...
Suhteellinen saantipolku: askel/askel/...
```

Saantipolku koostuu yhdestä tai useammasta *askeleesta* (location step) ja jokaista askelta verrataan nykyisen solmujoukon solmuihin. Askel koostuu kolmesta osasta:

- Akselista, mikä määrittää solmujen väliset suhteet.
- Solmutestistä, mikä määrittää valittujen solmujen nimen ja tyypin.
- Nollasta tai useammasta predikaatista, mitkä määrittävät askeleen valitsemia solmuja.

Askeleen syntaksi muodostuu akselin nimestä ja solmutestistä, mitkä erotetaan toisistaan tuplakaksoispisteellä `::` sekä mahdollisista predikaateista ja on muodoltaan `akseli::solmutesti[predikaatit]`. Taulukko 4 esittää XPath:n sisältämien akselien nimet ja niiden valitsemat solmut.

Akselin nimi	Kuvaus
`ancestor`	Kontekstisolmun esi-isät.
`ancestor-or-self`	Kontekstisolmu ja sen esi-isät.
`attribute`	Kontekstisolmun attribuutit.

`child`	Kontekstisolmun lapset.
`descendant`	Kontekstisolmun jälkeläiset.
`descendant-or-self`	Kontekstisolmu ja sen jälkeläiset.
`following`	Kaikki kontekstisolmua seuraavat.
`following-sibling`	Kaikki kontekstisolmua seuraavat sisarukset.
`namespace`	Kontekstisolmun nimiavaruussolmut.
`parent`	Kontekstisolmun vanhempi.
`preceding`	Kaikki kontekstisolmua edeltävät.
`preceding-sibling`	Kaikki kontekstisolmua edeltävät sisarukset.
`self`	Kontekstisolmu itse.

Taulukko 4. XPath:n akselit.

Jokaisella XPath:n akselilla on pääsolmutyyppi. Mikäli akseli voi sisältää elementin, on sen pääsolmutyyppi myös elementti. Muutoin akselin pääsolmutyyppi on se, mitä se voi sisältää eli `attribute`-akselille attribuutti ja `namespace`-akselille nimiavaruus.

Askeleen solmutesti voi olla tosi, jos ja vain jos, akselin solmutyyppi vastaa akselin pääsolmutyyppiä. Esimerkiksi solmutesti `child::para` valitsee kontekstisolmun `para`:n elementtilapset, mikäli niitä on. Muutoin valitaan tyhjä solmujoukko. Solmutesti `attribute::lang` valitsee kontekstisolmun `lang`-attribuutin tai palauttaa tyhjän joukon ellei attribuuttia ole. Solmutesti `*` on tosi jokaiselle pääsolmutyypille. Esimerkiksi `child::*` valitsee kaikki kontekstisolmun elementtilapset ja `attribute::*` kaikki attribuutit.

Askeleen predikaatteja käytetään etsittäessä tarkoin määriteltyä solmua tai etsittäessä solmua, mikä sisältää tarkoin määritellyn arvon tai arvoja. Periaatteessa tämä tapahtuu tuloksen suodattamisella. Esimerkiksi `child::para[1]` valitsee kontekstisolmun ensimmäisen `para`:n elementtilapsen ja `child::para[last()]` valitsee viimeisen `para`:n elementtilapsen.

Saantipolkujen kirjoittamiseen voi myös käyttää lyhennettyä syntaksia. Tärkein lyhennys on, että `child::` on määritelty oletusakseliksi. Esimerkiksi saantipolku `div/para` tarkoittaa samaa kuin `child::div/child::para`. Taulukko 5 esittää XPath:n käytössä olevat lyhennykset.

Lyhennys	Kuvaus
`@`	`attribute::`
`//`	`/descendant-or-self::node()`
`.`	`self::node()`
`..`	`parent::node()`

Taulukko 5. XPath:n lyhennykset.

XPath sisältää myös operaattoreita, mitkä esitetään taulukossa 6. Näitä operaattoreita voidaan käyttää XPath ilmaisuissa.

Operaattori	Kuvaus	Palautusarvo
`|`	Yhdistää solmujoukot	Solmujoukko
`+`	Yhteenlasku	Liukuluku
`-`	Vähennyslasku	Liukuluku
`*`	Kertolasku	Liukuluku

`div`	Jakolasku	Liukuluku
`=`	Yhtä suuri kuin	Tosi/epätosi
`!=`	Eri suuri kuin	Tosi/epätosi
`<`	Pienempi kuin	Tosi/epätosi
`<=`	Pienempi tai yhtä suuri kuin	Tosi/epätosi
`>`	Suurempi kuin	Tosi/epätosi
`>=`	Suurempi tai yhtä suuri kuin	Tosi/epätosi
`or`	Tai	Tosi/epätosi
`and`	Ja	Tosi/epätosi
`mod`	Modulus	Liukuluku

Taulukko 6. XPath:n operaattorit.

Lisäksi XPath sisältää useita erilaisia funktioita mm. numeroiden ja merkkijonojen käsittelyyn. (Malhotra A., Melton J. & Walsh N. 2007)

5.4 XSLT Esimerkkejä

Seuraavassa esitellään kuinka XSLT-muunnoksia käytetään. XML-dokumenttina toimii osa lähdeluetteloa, mistä on otettu kaksi lähdettä käsittelyyn. Esimerkissä 7 esitellään XML-dokumentti `lähdeluettelo.xml`. XML-dokumentista on jätetty skeema pois selvyyden takia.

`Esimerkki 7. XML-dokumentti lähdeluettelo.xml.`

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

```
<Lähdeluettelo>
   <Lähde Julkaisuvuosi="1999">
   <Tekijä>Harold E. R.</Tekijä>
   <Nimeke>XML Bible</Nimeke>
   <Julkaisija>IDG Books Worldwide</Julkaisija>
   <ISBN>951-762-765-3</ISBN>
</Lähde>
<Lähde Julkaisuvuosi="2000">
   <Tekijä>Phillips L. A.</Tekijä>
   <Nimeke>Using XML</Nimeke>
   <Julkaisija>QUE</Julkaisija>
   <ISBN>0-7897-1996-7</ISBN>
</Lähde>
</Lähdeluettelo>
```

Esimerkki 8 esittelee XSLT-muunnoksen, minkä tarkoituksena on tehdä XML-dokumentista HTML-tiedosto. Tyylisivu etsii XML-dokumentista lähteiden nimen ja tekijän ja muodostaa niistä taulukon. Yksinkertaisuuden vuoksi kaikki otsikkotiedot on jätetty HTML-koodista pois. Tyylisivu voidaan liittää XML-dokumenttiin lisäämällä XML-julistuksen alle rivi `<?xml-stylesheet type="text/xsl" href="Tyylisivun_nimi.xsl"?>`. Tällöin web-selain muuntaa XML-tiedoston automaattisesti HTML-muotoon ja XML-dokumentin alku näyttää seuraavalta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="html.xsl"?>
<Lähdeluettelo>
.
</Lähdeluettelo>
```

Esimerkki 8. XSLT-tyylisivu HTML:n luontiin html.xsl.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" />
<xsl:template match="/">
<html>
<body>
<table border="1">
<tr>
<th>Nimeke</th>
<th>Tekijä</th>
</tr>
<xsl:for-each select="Lähdeluettelo/Lähde">
<tr>
<td><xsl:value-of select="Nimeke"/></td>
<td><xsl:value-of select="Tekijä"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Tyylisivun rakenne alkaa XML-dokumentti-julistuksella ja XSLT-tyylisivu -julistuksella. Tämän jälkeen `<xsl:output method="html" />` määrittää ulostulevan tulosteen muodon HTML-muotoiseksi. `<xsl:template match="/">` valitsee

kontekstisolmuksi juurisolmun, mikä on `Lähdeluettelo`. `<xsl:for-each select="...">`-elementin XPath-lausekkeena on `Lähdeluettelo/Lähde` mikä valitsee kaikki `Lähdeluettelo`-elementin `Lähde`-nimiset jälkeläiset. `<xsl:value-of select="..." />`-lauseke valitsee tulosteeseen kaikki solmut, joiden nimi on `...` paikalla. Tällainen XPath-lauseke etsii XML-dokumentista kaikki `Lähde`-solmun lapset, joiden nimi on `Nimeke` tai `Tekijä` ja siirtää ne tulosteeseen, kuten esimerkistä 9 huomaa.

Esimerkki 9. html.xsl tyylisivun tulos.

```
<html>
<body>
<table border="1">
<tr>
<th>Nimeke</th>
<th>Tekijä</th>
</tr>
<tr>
<td>XML Bible</td>
<td>Harold E. R.</td>
</tr>
<tr>
<td>Using XML</td>
<td>Phillips L.A.</td>
</tr>
</table>
</body>
</html>
```

Nimeke	Tekijä
XML Bible	Harold E. R.
Using XML	Phillips L. A.

Edellisessä esimerkissä käytetty tapa on yleisin, kun käytetään XPath lausekkeita. Suurin osa XSLT-tyylisivuista tehdään yhtä dokumenttia varten, joten XPath lausekkeena voidaan käyttää selväsanaisia ilmaisuja, jolloin tuloksena saadaan aivan samanlainen tulos kuin käytettynä akseli-muotoista ilmaisua.

Akseli-muotoinen ilmaisu tuottaa aivan saman lopputuloksen kuin selväsanainen XPath-lauseke, kuten esimerkki 10 osoittaa. Siinä esitellään tyylisivu, mikä tekee uuden XML-muotoisen lähdeluettelon kaikista lähteistä, mitkä on tehty 2000-luvulla. Tarkoituksena on ainoastaan kopioida XML-dokumentti toiseksi dokumentiksi eikä tehdä siihen mitään muutoksia. Tällaisella tyylisivulla voidaan suodattaa tuloksesta pois kaikki solmut, mitä ei haluta ehdon tai ehtojen mukaan tulosteeseen.

Esimerkki 10. XSLT-tyylisivu XML-tiedoston luomiseksi xml.xsl.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="/">
<xsl:element name="Lähdeluettelo" >
<xsl:for-each select="descendant-or-self::*">
<xsl:if test="@Julkaisuvuosi &gt;=  2000">
<xsl:copy-of select="."/>
</xsl:if>
</xsl:for-each>
</xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Koska dokumentti muutetaan XML-muotoon, niin `output method`-elementti saa attribuutikseen `xml`. Alussa kontekstisolmuksi valitaan juurisolmu, minkä jälkeen koko puu käydään läpi `<xsl:for-each select="descendant-or-self::*">`-lausekkeella, mikä valitsee kaikki kontekstisolmun jälkeläiset ja itse kontekstisolmun.

`Lähde`-elementit valitaan siten, että attribuutti `Julkaisuvuosi` on 2000 tai suurempi. Attribuuttiin päästään kiinni lisäämällä attribuutin nimen eteen merkki `@`. Koska XPath-lausekkeissa ei voi käyttää `<` ja `>` merkkejä, sillä ne kuuluvat XSLT:n muotoiluun, niin ne joudutaan korvaamaan `<` ja `>` merkeillä. Näin ollen lausekkeeksi muodostuu `<xsl:if test="@Julkaisuvuosi >= 2000">`. Tämä komento etsii kaikki `Lähde`-elementit, joiden `Julkaisuvuosi`–attribuutti on 2000 tai suurempi.

`<xsl:copy-of select="."/>` lauseke kopioi tulostuspuuhun valitut solmut kokonaisuudessaan sekä kaikki niiden lapsisolmut, sillä lyhenne `.` vastaa komentoa `self::node()` ja `copy-of select` valitsee mukaan myös lapsisolmut. Mikäli tulospuuhun tulisi enemmän kuin yksi lähde, ei tulosdokumentti olisi XML suosituksen mukaista, sillä siihen tulisi useampia juurisolmuja. Näin ollen pitää tulostuspuuhun tehdä juurisolmu, ja se tapahtuu määrittelemällä uusi elementti `Lähdeluettelo`. Tämä tapahtuu `<xsl:element name="Lähdeluettelo" >`-komennolla joka suljetaan vasta juuri ennen mallineen sulkemista. Näin kaikki valitut `Lähde` solmut kopioituvat juurielementin sisään kuten tulos esimerkissä 11 esittää.

`Esimerkki 11. xml.xsl tyylisivun tulos.`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Lähdeluettelo>
<Lähde Julkaisuvuosi="2000">
<Tekijä>Phillips L. A.</Tekijä>
<Nimeke>Using XML</Nimeke>
<Julkaisija>QUE</Julkaisija>
</Lähde>
```

```
</Lähdeluettelo>
```

Tällaisten XPath-rakenteiden, kuten edellisessä esimerkissä näytettiin, käyttäminen mahdollistaa myös yleiskäyttöisempien XSLT-tyylisivujen rakentamisen. Käyttämällä akselimuotoisia ilmaisuja, kuten `descendant-or-self::*` saadaan XML-puu käytyä kokonaan läpi. Tällöin myös kaikki solmut, joilla on attribuutti `Julkaisuvuosi` minkä arvona on 2000 tai suurempi, olisi kopioitu tulosteeseen.

Esimerkissä 12 osoitetaan, kuinka lähdeluettelosta saadaan RDF-dokumentti, mikä käyttää Dublin Core metadata määrityksiä. Dublin Core on yhteisö, jonka tarkoituksena on luoda metadatastandardi erityisesti digitaalisten julkaisuiden kuvailuun. Dublin Core sisältää tällä hetkellä yleisimmät kentät useiden erilaisten aineistojen kuvailuun ja niitä on 15. Monessa tapauksessa nämä 15 kenttää eivät riitä kuvailutarpeisiin, mutta Dublin Coren etuna on se, että se on helposti laajennettavissa eri organisaatioiden tarpeisiin.

Suomessa kansalliskirjasto pitää yllä suomenkielistä versiota Dublin Core:sta (Kansalliskirjasto 2008). Lisäksi kansalliskirjasto on tehnyt oman metadata-formaatin opinnäytteiden verkkojulkaisemiseen, mikä on Dublin Core -pohjainen. Lisäksi Suomen valtion- ja kunnallishallintoa koskevat JHS-suositukset, mitkä kuvailevat asiakirja-aineistoille tarkoitettua metadataformaattia, on tehty Dublin Core -suositusten pohjalta. JHS-suositusten taustalla on yhtenäistää tietojen yhteen toimivuutta ja tehostaa olemassa olevan tiedon hyödyntämistä. Nämä suositukset hyväksyy julkisen tietohallinnon neuvottelukunta JUHTA. (JUHTA 2008)

Esimerkki 12. XSL-tyylisivu RDF-dokumentin luomiseksi.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

```
<xsl:output method="xml" />
<xsl:template match="/">
<rdf:RDF>
<xsl:for-each select="Lähdeluettelo/Lähde">

<xsl:variable name="title">
<xsl:value-of select="Nimeke"/>
</xsl:variable>

<rdf:Description rdf:about="http://www.esimerkki.fi/{$title}">
<dc:author><xsl:value-of select="Tekijä" /></dc:author>
<dc:publisher><xsl:value-of select="Julkaisija" /></dc:publisher>
<dc:identifier><xsl:value-of select="ISBN" /></dc:identifier>
<dc:date><xsl:value-of select="@Julkaisuvuosi" /></dc:date>
</rdf:Description>
</xsl:for-each>
</rdf:RDF>
</xsl:template>
</xsl:stylesheet>
```

Esimerkissä 12 olevan tyylisivun tarvitsee käyttää myös muita nimiavaruuksia kuin pelkkää `xsl`-nimiavaruutta. Koska tehdään RDF:ää ja käytetään Dublin Core -metadataa, pitää niiden nimiavaruudet myös esitellä. `xsl`-etuliitteen esittelyn jälkeen. Tämän esittelyn jälkeen oleva esittely osoittaa etuliitteen `dc` käyttävän Dublin Core nimiavaruutta, mikä löytyy osoitteesta `http://purl.org/dc/elements/1.1/`. Lisäksi esittely kertoo `rdf` etuliitteen käyttämän RDF-syntaksin nimiavaruuden osoitteen `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

Tyylisivun tarkoitus on tehdä RDF-kuvaukset lähteen teoksen mukaan ja lisätä kuvaukseen tieto tekijästä, julkaisija, päiväys ja ISBN-numero. Nimiavaruuksien esittelyn jälkeen valitaan juurielementti kontekstisolmuksi ja ulostulo metodi valitaan XML:ksi, sillä RDF kirjoitetaan XML:llä. RDF-dokumentti alkaa aina `<rdf:RDF>` elementillä mikä sisältää tarvittavat nimiavaruusesittelyt, mitkä ovat tässä tapauksessa `rdf` ja `dc` -etuliitteiden esittely. Tyylisivu siirtää automaattisesti etuliitteiden nimiavaruuksien esittelyn tulospuun juurielementtiin. Lisäksi tyylisivu/malline siirtää kaiken kirjoitetun tekstin tulosdokumenttiin, joten juurielementin luomiseksi riittää kirjoittaa `<rdf:RDF>` mikä luo RDF-dokumentin juurielementin.

Kun kaikki `Lähde` solmut on valittu, tehdään paikallinen muuttuja `title` `<xsl:variable name="title">` lausekkeella ja muuttujan arvoksi valitaan `Nimeke`-elementin sisältö. Tämä siitä syystä, että näin saadaan RDF-kuvaus tehtyä nimekkeen perusteella.

Tämän jälkeen alkaa itse RDF-kuvauksen rakentaminen. RDF-kuvaus alkaa resurssin esittelyllä mikä tapahtuu lausekkeella `<rdf:Description rdf:about="...">`, missä `...` paikalle tulee resurssin URI. Tässä tapauksessa URI on `http://www.esimerkki.fi/Nimeke`, ja että saataisiin oikea teos, niin käytetään edellä esiteltyä muuttujaa. Tyylisivulla esitettyyn muuttujaan pääsee kiinni lisäämällä `$`-merkki muuttujan nimen eteen ja laittamalla se aaltosulkeisiin. Näin URI:ksi muodostuu `http://www.esimerkki.fi/{$title}`.

Resurssin kuvauksen jälkeen tulee resurssin ominaisuuksien kuvaukset sekä niiden arvot. Kuvauksissa käytetään Dublin Core -metadataa hyväksi, jolloin ominaisuudet pitää kirjoittaa tyylisivulle. Tämä tapahtuu esim. tekijän osalta laittamalla `<dc:author>...</dc:author>`, mikä tarkoittaa että `...` paikalla on ominaisuuden `author` arvo. Arvo saadaan XML-dokumentista yksinkertaisesti valitsemalla solmun `Tekijä` sisältö, mikä tapahtuu lausekkeella `<xsl:value-of select="Tekijä" />`. Näin tehdään myös muidenkin ominaisuuksien kohdalla.

Lopuksi suljetaan RDF-kuvaus </rdf:Description>-elementillä sekä RDF-dokumentti </rdf:RDF>-elementillä. Kun tiedostot laitetaan XSLT-parserin läpi, niin tulokseksi saadaan RDF-dokumentti, mikä esitellään esimerkissä 13.

Esimerkki 13. RDF-kuvaus lähdeluettelo.xml dokumentista.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF               xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description
    rdf:about="http://www.esimerkki.fi/951-762-765-3">
<dc:Author>Harold E. R.</dc:Author>
<dc:title>XML Bible</dc:title>
<dc:Publisher>IDG Books Worldwide</dc:Publisher>
<dc:date>1999</dc:date>
</rdf:Description>
<rdf:Description
    rdf:about="http://www.esimerkki.fi/0-7897-1996-7">
<dc:Author>Phillips L. A.</dc:Author>
<dc:title>Using XML</dc:title>
<dc:Publisher>QUE</dc:Publisher>
<dc:date>2000</dc:date>
</rdf:Description>
</rdf:RDF>
```

6 XML:n muuntaminen RscDF:ksi

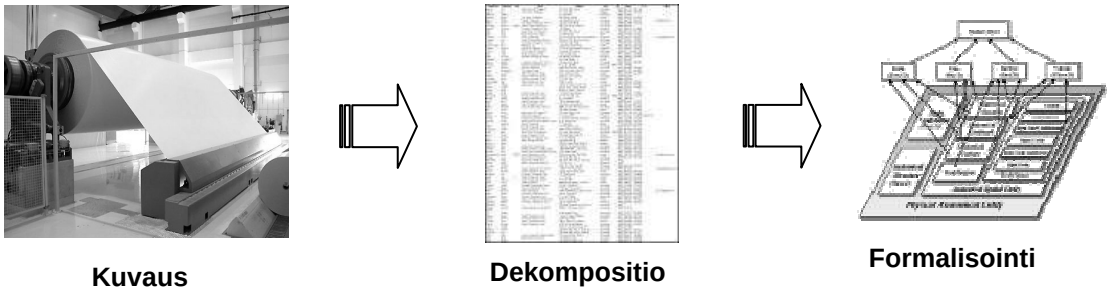
Tässä luvussa esitellään kuinka XML muunnetaan RscDF-muotoon. Aluksi kerrotaan XML^0-rakenteen suunnittelusta ja tämän jälkeen itse muunnoksen toteuttamisesta. Luvussa esitetyt tiedot ovat tuloksia SmartResource-projektista, minkä tarkemmat tiedot löytyvät projektin teknisistä raporteista. (Kaykova ym. 2004a; Kaykova ym. 2004b)

6.1 XML perusmuoto

Syntaktisen muunnoksen suurimpana haasteena on XML^0-rakenteen suunnittelu ja toteutus. Tällaisen laajan XML-skeeman suunnittelu ja toteutus vaatii paljon ammattitaitoa ja kokemusta kyseessä olevasta ongelma-alueesta.

Ongelma-alue, mikä kiinnosti erityisesti SmartResource-projektissa, liittyi paperiteollisuuteen, paperikoneisiin ja paperinvalmistusprosesseihin. Peruslähtökohtana kanonisen mallin suunnittelussa lähdetään rakentamaan käsitteellistä mallia ongelma-alueesta kuten kuvio 17 esittää. Ensinnäkin ongelma-alue pitää pystyä kuvailemaan normaalilla kielellä siten, että kuvailu sisältää kaikki tärkeimmät näkökohdat alueesta. Eli kuten tässä tapauksessa kuvailut olisivat esim. ”paperikone tekee paperia”, ”paperin valmistuksessa käytetään selluloosaa” jne.

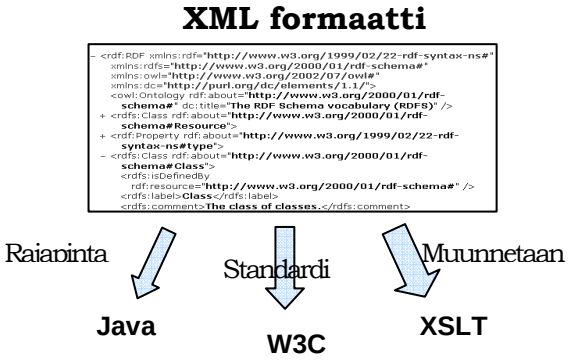
Toisessa vaiheessa suoritetaan ongelma-alueen dekompositio kuvailujen pohjalta. Tällä tarkoitetaan olioiden, luokkien ja ominaisuuksien relaatioiden ja käyttäytymissuhteiden erottamista toisistaan. Tämän jälkeen ongelma-alueen kuvailu suoritetaan käyttämällä vapaasti valittavaa formalisointi tekniikkaa. Formalisointi tehdään käyttämällä esim. ER-kaavioita, UML:ää, ontologiaa jne.



Kuvio 17. Ongelma-alueen adaptaatio.

Edellä mainittujen vaiheitten jälkeen alkaa datan esitysstandardin analysointi, kuten kuviossa 18 esitetään, mitä käytetään tehtäessä kanoninen muoto. Tämä sisältää kulloinkin käytettävän datan, tässä tapauksessa XML:n, esitysstandardin analysoinnin, käytettävissä olevien ohjelmointirajapintojen analysoinnin jne.

Näin saadaan kaikki tarvittava tieto ongelma-alueesta, käytettävästä standardista ja ohjelmointirajapinnoista.



Kuvio 18. Datan esitysformaatin analyysi.

Tällaisesta käyttökelpoisesta XML^0-rakenteen suunnittelusta voi tässä yhteydessä mainita PaperIXI-projektin (PaperIXI), minkä tarkoituksena on suunnitella XML-pohjainen standardi paperitehtaasta koko sen elinkaaren ajalle. Tästä johtuen ei SmartResource-

projektissa lähdetty toteuttamaan erillistä laajaa XML^0-rakennetta, vaan katsottiin rakenteen saatavan PaperIXI-projektin tuloksena.

Kuitenkin testaustarpeita varten rakennettiin erään XML-muotoisen vikailmoituksen pohjalta XML^0-rakenne, minkä muoto esitellään esimerkissä 14. Rakenne tehtiin vastaamaan RscDF:n tarpeita, joten kaikki mitä itse RscDF ei tarvitse, jätettiin rakenteesta pois. Vikailmoitus saatiin projektin yhteistyökumppanilta Metso Automationilta. XML-dokumenttiin on jätetty myös elementtien ja attribuuttien arvot lukemisen helpottamiseksi ja sen skeema esitellään liitteessä 1.

Esimerkki 14. Rakennettu XML^0-rakenne.

```
<?xml version="1.0" encoding="UTF-8"?>
<alrm:State                xmlns:alrm="http://www.metso.com/Alarm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=http://www.metso.comAlarm
file:/.../XML0Skeema.xsd
Time="2004-09-28T23:50:12.578">
    <Measurement>
       <ParamType>Temperature</ParamType>
       <Units>degree</Units>
       <Value>70</Value>
       <Sensor sensorID="KX2834-S">Surface_Temp</Sensor>
    </Measurement>

    <Measurement>
    .
    </Measurement>
</alarm:State>
```

Edellä esitelty XML^0-rakenne kuvaa laitteen tilaa tietyllä ajan hetkellä. Se sisältää `<alarm:State>`-juurielementin mihin sisältyy aika-attribuutti `Time`. Lapsielementtinä on `<Measurement>`, mikä pitää sisällään tietyn sensorin mittaustuloksen. Näitä mittaustuloksia voi olla peräkkäin eri antureilta ko. ajan hetkeltä vaikka kuinka paljon.

Näin ollen tiedot vastaavat hyvin RscDF:n vaatimuksia, sillä sen tarkoituksenahan on keskittyä resurssien ylläpitoon liittyvien tilojen kuvailemiseen.

6.2 Muunnoksen toteutus

Kun XML-perusmuoto on olemassa, niin voidaan toteuttaa muunnos kahden XML-rakenteen välillä. Muunnos jakautuu karkeasti kolmeen osaan:

- Solmujen yhteensovittamiseen,
- skeemojen tarkasteluun ja
- muunnossääntöjen rakentamiseen.

Solmujen yhteensovittamisella tarkoitetaan sitä, että mitä elementtiä tai attribuuttia XML^0-rakenteessa vastaa alkuperäisen XML-dokumentin tietty elementti tai attribuutti. Esimerkiksi XML^0-dokumentissa oleva `KX2834-S` vastaa alkuperäisen XML-dokumentin sensorin nimeä, sillä silloin se saadaan vastaamaan `sensorID`-attribuuttia XML^0-rakenteessa mikä kertoo sensorin id-arvon.

Skeemojen tarkastelulla tarkoitetaan, että alkuperäisen XML-dokumentin skeema on tarkistettava. Tämä siitä johtuen, että menemättä sen enempää itse skeeman standardiin, niin se mahdollistaa antaa elementeille ja attribuuteille myös ominaisarvoja tai arvojoukkoja. Toisekseen skeema voi myös mahdollistaa joidenkin elementtien tai attribuuttien olevan vapaaehtoisia. Tämä tarkoittaa sitä, että skeemaan pohjautuva XML-dokumentti voi sisältää tiettyjä osia, mutta se ei ole välttämättömyys. Ilman skeeman läpikäymistä voisi tulla eteen tilanteita, että jotain tietoja ei aina muunnetakaan XML^0-dokumenttiin. Tällöin ainoastaan tietyissä tilanteissa esille tulevien solmujen tiedot jäisivät muuntamatta.

Kun solmujen vastaavuudet on saatu selville, alkaa itse muunnossääntöjen rakentaminen XSLT-muunnoksen avulla. Alkuperäisestä XML-dokumentista pitää saada kaikki tarpeellinen tieto siirrettyä XML^0-rakenteeseen ja myös tarvittavat muunnokset on tehtävä. Esimerkissä 15 esitetään pieni osa XSLT-muunnoksesta, missä `ParamType`-elementti tehdään.

Esimerkki 15. Esimerkki XSLT-muunnoksesta.

```
<Measurement>
 <xsl:for-each select="ParamType">
  <ParamType>
   <xsl:value-of select="."/>
  </ParamType>
 </xsl:for-each>
</Measurement>
```

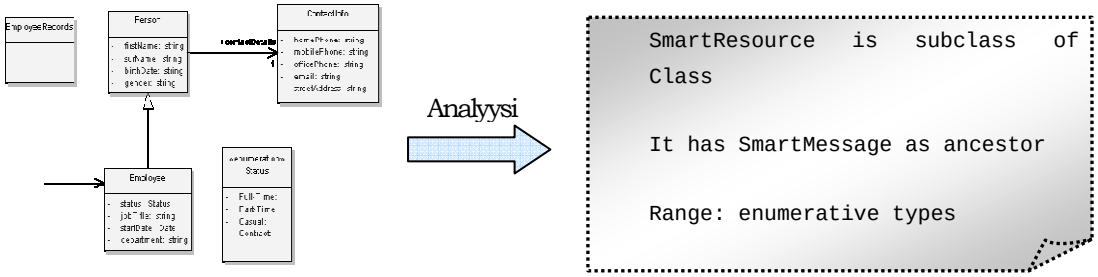
Edellä esitetty XSLT-muunnoksen osa käy läpi alkuperäistä XML-dokumenttia ja sen solmuja. Alkuun tehdään `<Measurement>`-elementti. Kun muunnos löytää `ParamType`-elementin niin se muodostaa uuden elementin nimeltään `ParamType` ja valitsee elementin arvon siihen lausekkeella `<xsl:value-of select="."/>`.

Tällä tavalla kun käytäisiin kaikki vikailmoituksen solmut läpi ja XSLT-muunnos rakennettaisiin täydelliseksi, saataisiin rakennettua XSLT-muunnosta käyttämällä kokonaan XML^0-rakennetta vastaava XML-dokumentti. Tämän pohjalta pystytään tekemään semanttinen muunnos RscDF:ksi.

6.3 Kanoninen semanttinen muunnos

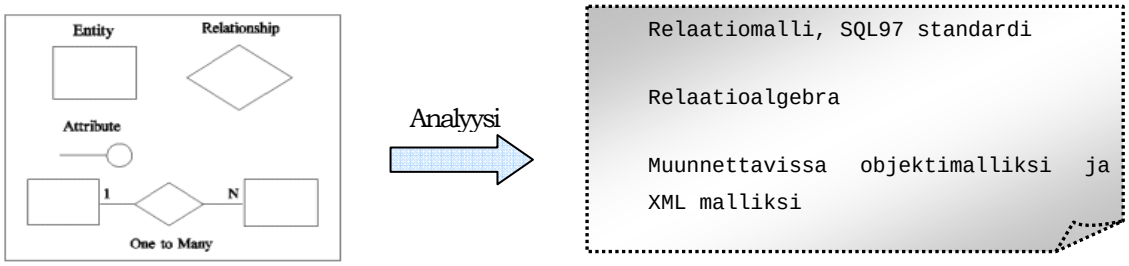
Kanonisen semanttisen muunnoksen toteuttaminen alkaa yleisesti metadatan analysoinnilla. Tämä sisältää analyysin käytettävän datamallin skeemasta (elementit,

relaatiosuhteet, jne.), niiden mahdollisista rajoituksista ja elementtien välisestä hierarkiasta ja rajoituksista kuten kuvio 19 esittää.



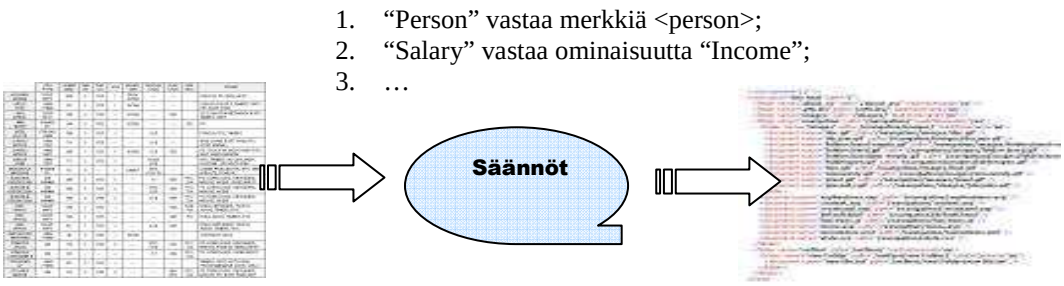
Kuvio 19. Metadatan analyysi.

Seuraavaksi täytyy analysoida käytettävän datamallin standardi, kuten kuviossa 20 esitetään. Tämä sisältää analyysin käytettävän datamallin standardin spesifikaatiosta, olemassa olevasta formaalista teoriasta, olemassa olevien muunnosmetodeista sekä elementtien hierarkiasta ja rajoituksista.



Kuvio 20. Standardin analyysi.

Kuvio 21 esittää viimeisen vaiheen muunnoksen suunnittelussa. Viimeisessä vaiheessa rakennetaan datan muunnossäännöt, eli miksi merkki tai ominaisuus muunnetaan muunnoksen aikana.



Kuvio 21. Muunnossääntöjen rakentaminen.

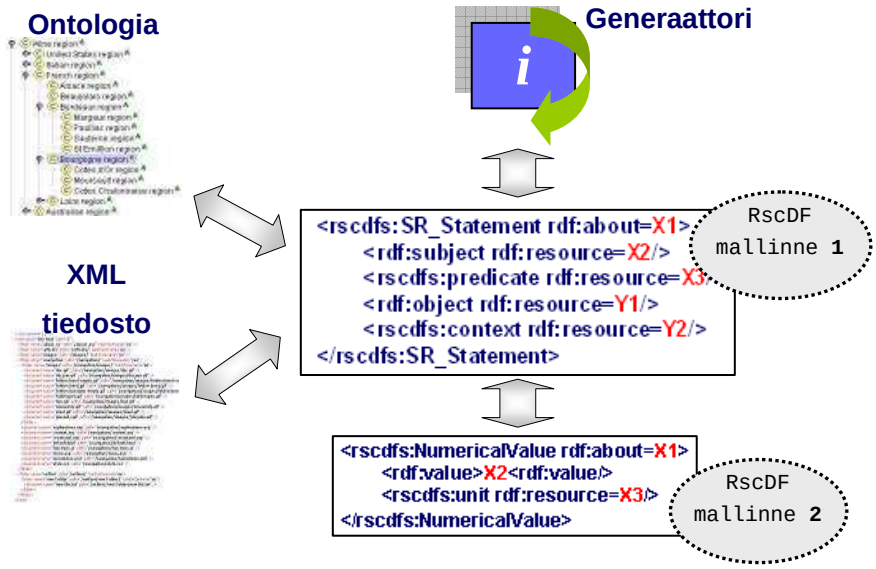
Kanoninen semanttinen muunnos toteutettiin SmartResource-projektissa käyttämällä mallinepohjaista lähestymistapaa. RscDF-skeeman analyysivaiheessa todettiin, että uudestaan käytettäviä mallinteita voitiin hyödyntää muunnoksen aikana. RscDF dokumentista saatiin kahdenlaisia mallinteita: rakenteellisia mallinteita (structural template) ja merkkimallinteita (tag template). Rakenteelliset mallineet vastaavat RscDF-dokumentin graafia. Riippuen kanonisesta XML-dokumentin rakenteesta, mikä voi joiltain osin olla samankaltainen RscDF-dokumentin graafin kanssa, niin se voidaan kloonata prosessin aikana.

Toisaalta merkkimallineet vastaavat RscDF:n luokkia tai tarkalleen ottaen RscDF:n skeeman luokkia. Näitä ovat esim. `SR_Statement, Context_SR_Container, SR_Container, NumericalValue, TempTempMark.` Merkkimallineet ovat ”palikoita”, mitä sovitin käyttää RscDF-dokumentin luomiseen. Näillä mallineilla on runko ja muuttuva osio, mikä voi sisältää neljää erilaista tyyppiä:

- Linkin toiseen mallineeseen,
- linkin XML dataan,
- linkin ontologiaan tai
- generoidun arvon.

Kuvio 22 havainnollistaa merkki-mallineen rakennetta ja esimerkki 16 esittää näytteen RscDF-dokumentista. Muuttuja X_n saadaan ajon aikana joko ontologiasta, XML-

dokumentista tai generaattorin generoimana. Muuttuja Y_n on tunnusnumero, mikä saadaan jostain toisesta mallineesta. Näin ollen RscDF-merkki riippuu muista RscDF-merkeistä, mitkä generoidaan myöhemmin. Tällä tavalla sovitin, rekursiivisesti kutsuu metodia tee-mallinne, kunnes saavutetaan lehti-solmut.



Kuvio 22. Malline-pohjainen muunnos.

`Esimerkki 16. Näyte RscDF-dokumentista.`

`#statement7` on generoitu arvo, #123456XZ24 tulee XML-dokumentista, `#temperature` sisältää linkin ontologiaan ja `#numValue3` sekä `#containen6temp` sisältävät tunnusnumeron linkin toiseen mallineeseen.

```
<rscdfs:SR_Statement rdf:about="http://...#statement7">
 <rdf:subject rdf:resource="http://...#123456XZ24"/>
 <rscdfs:predicate rdf:resource="http://...#temperature"/>
 <rdf:object rdf:resource="http://...#numValue3"/>
 <rscdfs:context rdf:resource="http://...#container6temp"/>
```

```
</rscdfs:SR_Statement>
```

`#numValue3` on linkki yllä olevaan mallineeseen, `70` on XML:stä tuleva arvo ja `#Celsius` on linkki ontologiaan.

```
<rscdfs:NumericalValue rdf:about="http://...#numValue3">
 <rscdfs:value>70</rscdfs:value>
 <rscdfs:unit rdf:resource="http://...#Celsius"/>
</rscdfs:NumericalValue>
```

Käyttämällä rakenteellisia ja merkki mallinteita sovitin suorittaa semanttisen muunnoksen. Tämä lähestymistapa mahdollistaa semanttisen muunnoksen logiikan implementoinnin muunnettaessa XML-muotoista dataa RscDF-muotoon. Se perustuu automaattiseen RscDF-instanssien generoimiseen XML:stä, mitkä määräytyvät ontologiamallineista. Ontologia sisältää luokiteltuja pareja kanonisen XML:n ja RscDF:n välillä, mitkä määrittävät niiden vastaavuuden toisiinsa. Näin ollen semanttinen muunnos kaksiosaista muunnosta käyttämällä XML:n ja RscDF:n välillä vaatii ainoastaan kaksi suhteellisen helppoa toimenpidettä, XML:n muuntamisen XML^0-muotoon sekä ontologiamallineiden rakentamisen. Näiden kahden vaiheen rakentamisen jälkeen muunnos on automaattinen.

7 Muunnoksen pilottiversio

Tässä luvussa esitellään, kuinka semanttinen muunnos toteutettiin ja kuinka sitä testattiin. Aluksi esitellään prototyyppiversio GAF:sta ja sen jälkeen laitesovittimen rakenne. Lopuksi näytetään muunnos oikealla datalla.

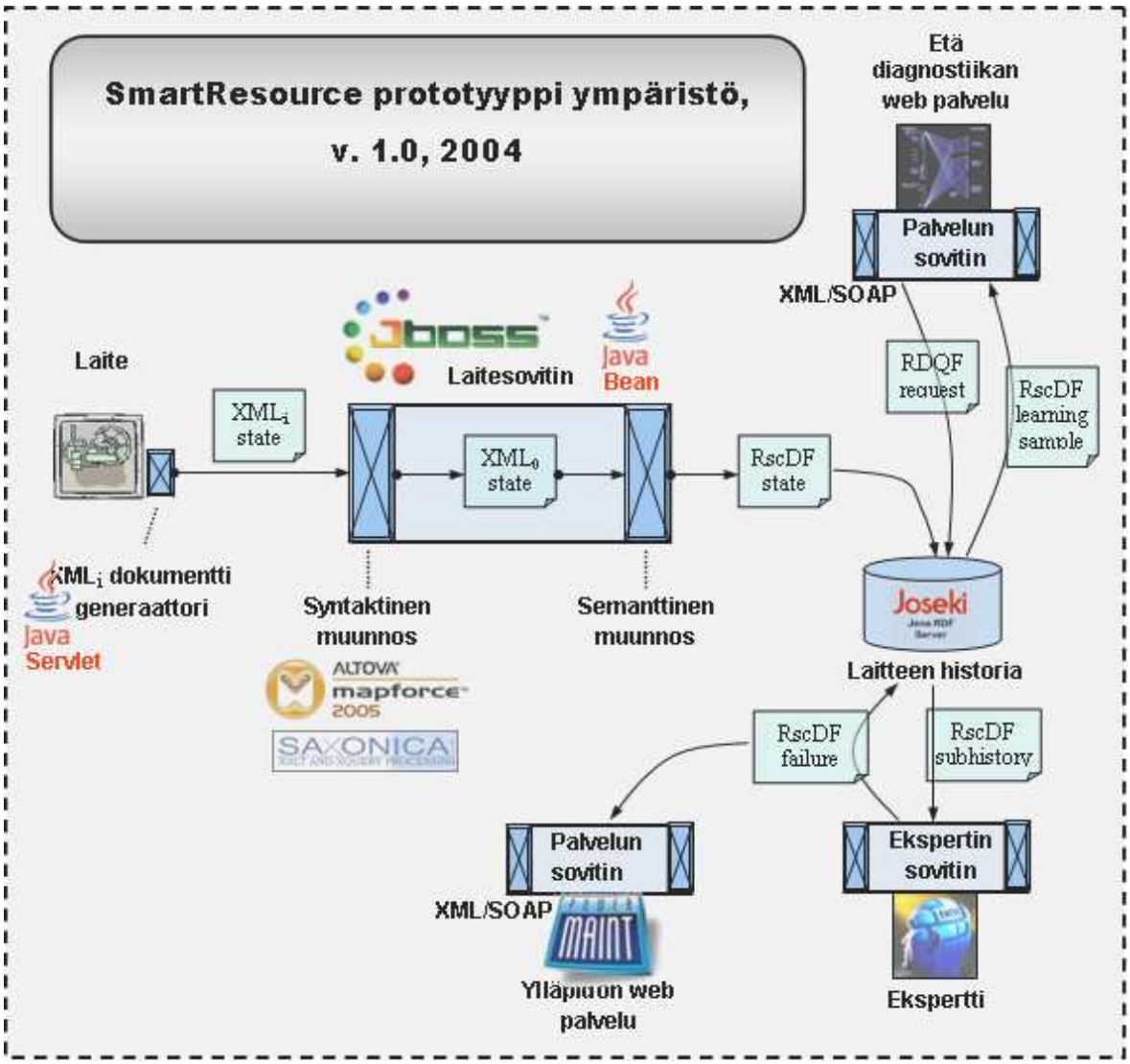
Luvussa esitetyt tiedot ovat tuloksia SmartResource-projektista, minkä tarkemmat tiedot löytyvät projektin teknisistä raporteista. (Kaykova ym. 2004a; Kaykova ym. 2004b)

7.1 Prototyyppi ympäristö

SmartResource-projektissa kaksiosaista muunnosta testattiin prototyyppisovelluksessa, minkä konsepti näkyy kuviossa 23. Pilottiversiota testattiin aidolla datalla, minkä malli tuli Metso Automationilta. Sovelluksessa testattiin paperikoneen tilaa seuraavan sensorin tuottamaa dataa ja sen muuntamista RscDF-ympäristöön. Paperikoneen tilaa kuvaavaa dataa generoitiin ohjelmistosimulaattorilla.

Prototyyppisysteemi perustuu J2EE (Java 2 Enterprise Edition) -alustalle ja käyttää hyväkseen JBoss sovelluspalvelinta. Prototyyppisovelluksessa toteutettiin laitesovitin, minkä tarkoituksena on suorittaa semanttinen muunnos kanonisesta XML^0-muodosta RscDF-muotoon. Laitesovittimen logiikka on koteloituna EJB:n (Enterprise JavaBeans) sisään ja suoritetaan JBoss-palvelimessa. Tämän jälkeen RscDF-muotoon muunnettu data tallennettiin Joseki RDF-palvelimelle.

Testaustarpeita varten rakennettiin kanoninen XML^0-muoto sekä kolme erilaista XML-skeemaa. Syntaktista muunnosta varten tehtiin kolme XSLT-muunnosta, minkä avulla XML-dokumentit muunnettiin kanoniseen XML^0-muotoon. Tämän jälkeen laitesovitin suoritti semanttisen muunnoksen toisen osan.



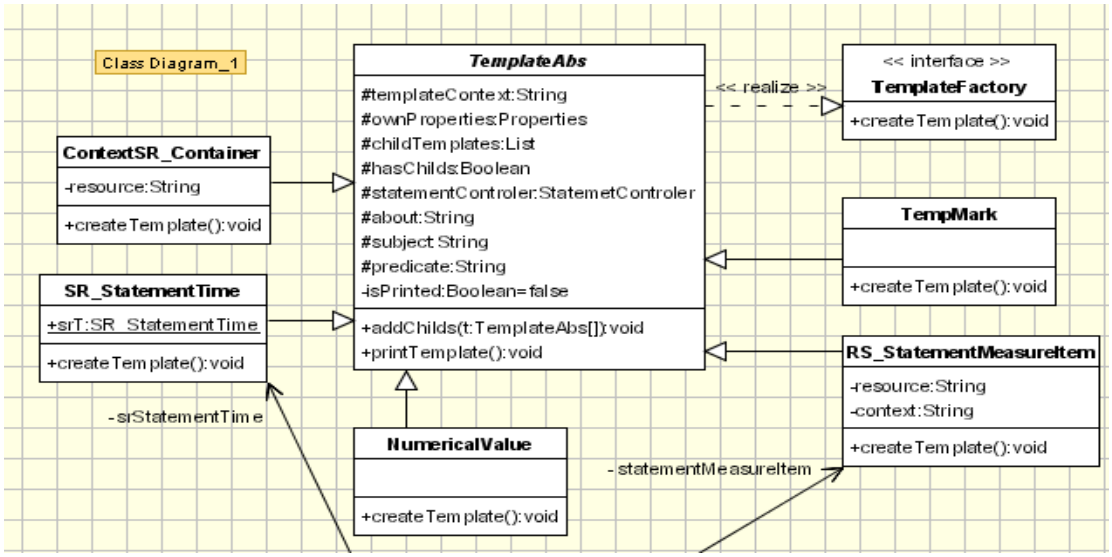
Kuvio 23. Prototyyppiympäristö.

7.2 Laitesovittimen rakenne

Kaikki luokat, mitkä muodostavat laitesovittimen, on pakattu yhteen mallinepakettiin. Loogisesti luokat voidaan jakaa neljään osioon. Ensimmäinen osa luokista vastaa logiikkaa, mikä heijastaa RscDF dokumentin rakennetta. Toinen osa luokista vastaa XML^0-dokumentin rakennetta ja kapseloi dokumentin prosessoinnin rakenteen. Kolmas luokka vastaa moottoria, minkä avulla RscDF-dokumentti rakennetaan ja neljäs sisältää DOM:n (Document Object Model) käyttöön liittyviä luokkia.

Kuvio 24 esittää RscDF:n dokumentin rakennetta kuvaavia luokkia. Ydin-luokkana on `TemplateAbs`, mikä sisältää kaiken logiikan RscDF-luokkien rakenteesta. Muuttuja `templateContext` esittää RscDF-merkin rakennetta. Muuttujat `subject`, `about` ja `predicate` vastaavat RDF-väittämiä. Säiliön `childTemplates` tarkoituksena on tallentaa kontekstisolmun kaikki lapsisolmut. `TemplateAbs`-luokka hyödyntää `TemplateFactory`-rajapintaa, mikä omaa ainoastaan metodin `createTemplate`, mitä kaikki alaluokat hyödyntävät. `TemplateAbs`-luokalla on `printTemplate`-metodi, mikä rekursiivisesti kutsuttuna kirjoittaa mallinemerkin sisällön.

Pääluokalla `TemplateAbs` on viisi ala-luokkaa, mitkä vastaavat RscDF-luokkia. Luokat ovat `ContextSR_Container`, `SR_StatementTime`, `NumericalValue`, `TempMark` ja `SR_StatementMeasureItem`. Jokainen näistä luokista hyödyntää `createTemplate`-metodin logiikkaa, minkä tarkoituksena on saada merkin sisältö ja kutsua samaa metodia myös kaikille esi-isille.



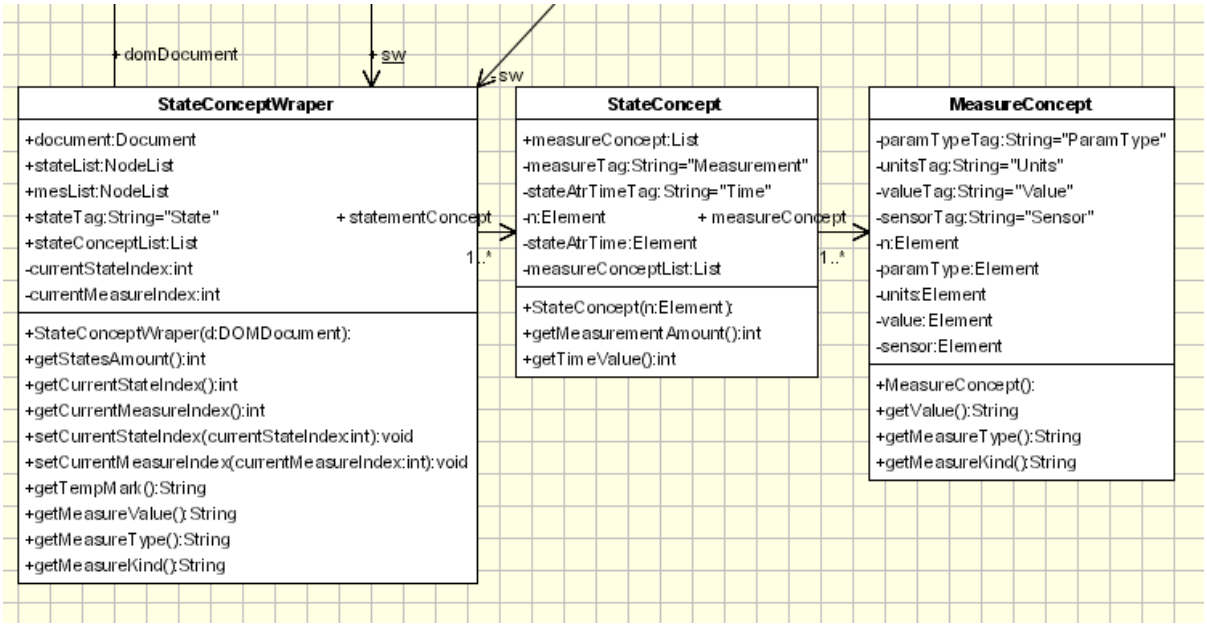
Kuvio 24. RscDF-dokumentin rakennetta kuvaavat luokat.

Kuvio 25 esittää luokkakaavion luokista, mitkä kuvaavat XML^0-dokumentin rakennetta. XML^0-dokumenttihan rakentui laitteen tilasta, mikä pystyi sisältämään mielivaltaisen

määrän mittaustuloksia. Luokka `MeasureConcept` vastaa XML^0-dokumentissa olevaa mittaustulosta. Jotkin osat tästä luokasta vastaavat XML-dokumentin solmuja. Esim. `sensor` ja `value` vastaavat XML^0-dokumentin `<Sensor>` ja `<Value>` solmuja. Metodit `getValue` ja `getMeasureType` hankkivat mittauksen tuloksen ja tyypin dokumentista.

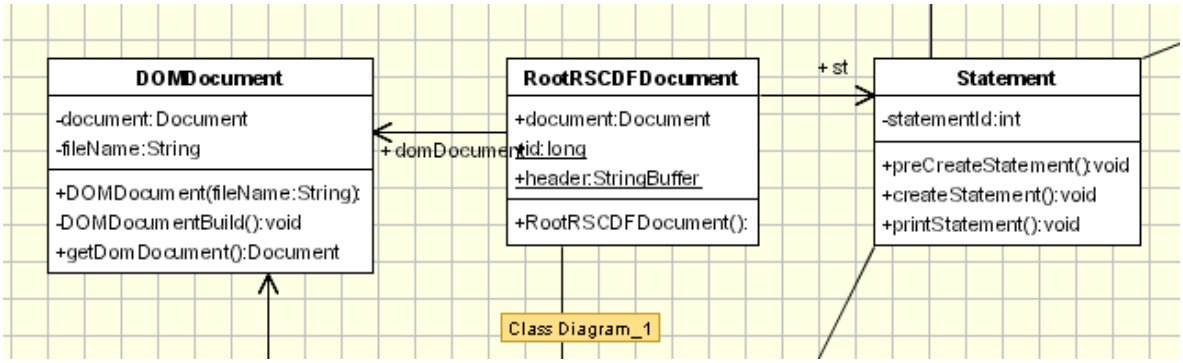
Luokka `StateConcept` vastaa XML^0-dokumentin juurisolmua `State`. Luokka toimii säiliönä mittaustuloksille ja sillä on `ArrayList`-säiliö `measureConsept`, mikä vastaa mittaustuloksia. Luokan päämetodina on `getTimeValue`, mikä hakee `State`-solmun aika-attribuutin.

`StateConseptWraper` implementoi logiikan, minkä avulla esitetään XML^0-dokumentin rakenne Javan luokkamallissa. Se pitää kirjaa kaikista XML-rakenteen tiloista ja sisältää metodeja, millä hankitaan referenssi nykyisestä tilasta ja mittauksesta muunnoksen aikana.



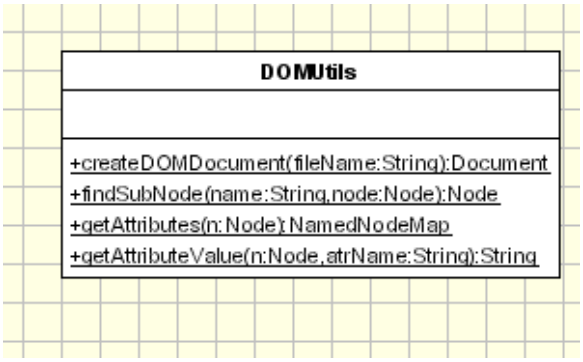
Kuvio 25. XML^0-dokumentin rakennetta kuvaavat luokat.

Kuvio 26 esittää RscDF moottoriluokan rakenteen. Luokka `RootRSCDFDocument` ajaa koko muunnosprosessia eteenpäin. Ensimmäiseksi se hankkii DOM-mallin XML[0]-dokumentista ja siirtää sen parsittavaksi `StateConseptWraper`-luokalle. Tämän jälkeen se saa referenssin parsitusta mallista ja aloittaa muunnoksen. Rakentaakseen RscDF-dokumentin, se kutsuu peräkkäin kolmea metodia: `preCreateStatement`, `createStatement` ja `printStatement`.



Kuvio 26. Moottoriluokan rakenne.

Kuvio 27 esittää joukon yleiskäyttöisiä DOM-mallista peräisin olevia XML-dokumenttien prosessointiin tarkoitettuja metodeja. Näiden metodien avulla rakennetaan Java-malli XML-tiedostoista. Näitä metodeja voidaan käyttää kaikkien XML-tiedostojen kanssa.



Kuvio 27. Yleisestä DOM-mallista perityt metodit.

7.3 Datan muuntaminen

Kaksiosaisen muunnoksen lähtökohta huomioon ottaen SmartResource-projektissa kehitettiin kanoninen XML^0-skeema ja syntaktisen muunnoksen testaamista varten kolme erilaista XML-skeemaa. Yksi kehitetyistä XML-dokumenteista esitetään esimerkissä 17 ja kanoninen XML^0-dokumentti esitetään esimerkissä 18. Kummankin dokumentin skeemat esitetään liitteessä 1.

Tämän jälkeen syntaktista muunnosta varten kehitettiin XSLT-muunnos, mikä muuntaa alkuperäisen XML-dokumentin rakenteen vastaamaan XML^0-dokumentin rakennetta. Seuraavien esimerkkien välinen XSLT-muunnos esitetään esimerkissä 19.

Esimerkki 17. XML-dokumentti 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<alrm:State
 xmlns:alrm="http://www.metso.com/Alarm"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.metso.com/Alarm
 file:/.../XML_1_Skeema.xsd"
 Time="2004-09-28T23:50:12.578">

<Measurement>
 <ParamType>Temperature</ParamType>
 <Units>degree</Units>
 <Value>70</Value>
 <Sensor>Surface_Temp</Sensor>
 <SensorID>KX2834-S</SensorID>
</Measurement>

<Measurement>
 <ParamType>RotationSpeed</ParamType>
```

```
 <Units>rpm</Units>
 <Value>1500</Value>
 <Sensor>Rotation_Speed</Sensor>
 <SensorID>RP2345-M</SensorID>
</Measurement>
</alrm:State>
```

Esimerkki 18. Kanoninen XML-dokumentti.

```
<?xml version="1.0" encoding="UTF-8"?>
<alrm:State
 xmlns:alrm="http://www.metso.com/Alarm"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.metso.com/Alarm
 file:/.../XML0Skeema.xsd"
 Time="2004-09-28T23:50:12.578">

<Measurement>
 <ParamType>Temperature</ParamType>
 <Units>degree</Units>
 <Value>70</Value>
 <Sensor sensorID="KX2834-S">Surface_Temp</Sensor>
</Measurement>

<Measurement>
 <ParamType>RotationSpeed</ParamType>
 <Units>rpm</Units>
 <Value>1500</Value>
 <Sensor sensorID="RP2345-M">Rotation_Speed</Sensor>
```

```
</Measurement>
</alrm:State>
```

Esimerkki 19. XSLT-muunnos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:alrm2="http://www.metso.com/Alarm"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:alrm="http://www.metso.com/Alarm"
exclude-result-prefixes="alrm2 xs">

<xsl:output method="xml" encoding="UTF-8" indent="yes"/>
<xsl:template match="/alrm2:State">
```

Seuraavassa määritetään juurielementti ja sille attribuutti, mikä kertoo skeeman sijainnin. Lisäksi rakennetaan aika-attribuutti `Time` ja otetaan sen arvo.

```
<alrm:State>
 <xsl:attribute name="xsi:schemaLocation">
 http://www.metso.com/Alarm/XML_0_Skeema.xsd</xsl:attribute>
  <xsl:for-each select="@Time">
   <xsl:attribute name="Time">
    <xsl:value-of select="."/>
   </xsl:attribute>
  </xsl:for-each>
```

Valitaan kaikki `Measurement`-elementit ja esitellään muuttuja `Vvar4_Measurement`. Tämän jälkeen tehdään `Measurement`-elementti.

```
<xsl:for-each select="Measurement">
 <xsl:variable name="Vvar4_Measurement" select="."/>
<Measurement>
```

Käydään kaikki `Measurement`-elementin lapset läpi, rakennetaan jokaiselle uusi elementti ja otetaan elementin arvo.

```
<xsl:for-each select="ParamType">
 <ParamType>
  <xsl:value-of select="."/>
 </ParamType>
</xsl:for-each>

<xsl:for-each select="Units">
 <Units>
  <xsl:value-of select="."/>
 </Units>
</xsl:for-each>

<xsl:for-each select="Value">
 <Value>
  <xsl:value-of select="."/>
 </Value>
</xsl:for-each>
```

Valitaan `Sensor`-elementit ja käytetään hyväksi aiemmin esiteltyä muuttujaa `sensorID`:n saamiseksi. Tämä sen takia, että tällöin saadaan `sensorID` oikean `Sensor`-elementin yhteyteen.

```
<xsl:for-each select="Sensor">
 <Sensor>
  <xsl:for-each select="$Vvar4_Measurement/SensorID">
   <xsl:attribute name="sensorID">
    <xsl:value-of select="."/>
   </xsl:attribute>
  </xsl:for-each>
 <xsl:value-of select="."/>
 </Sensor>
</xsl:for-each>
```

Lopuksi suljetaan kaikki avoinna olevat elementit.

```
</Measurement>
</xsl:for-each>
</alrm:State>
</xsl:template>
</xsl:stylesheet>
```

Edellä esitetyn XSLT-muunnoksen avulla saadaan alkuperäinen XML-dokumentti muunnettua kanoniseen XML^0-muotoon. XSLT-muunnoksen jälkeen sovitin suorittaa semanttisen muunnoksen XML^0-muodosta RscDF-muotoon. Muunnoksessa muodostunut RscDF-dokumentti esitetään esimerkissä 20. RscDF-dokumentissa kuvataan kuvitteellisen laitteen 123456XZ24 arvoja. Resurssien URI-viittauksia on lyhennetty lukemisen helpottamiseksi.

Esimerkki 20. Muodostettu RscDF-dokumentti.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
```

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rscdfs="http://.../rscdfs#"
xmlns:ontoContainer="http://.../ontologies/containerOntology#"
xmlns:ontoMeasuremnt="http://.../ontologies/measurementOntology#"
xmlns:ontoTempMark="http://.../ontologies/tempMarkOntology#"
xmlns:ontoMeasuremntUnit="http://.../ontologies/measurementUnitOnt
ology#">
```

Seuraava väittämä sisältää väittämän laitteen 123456XZ24 yhden sensorin mittaustuloksesta sekä linkit `numValue3` ja linkin `container6temp`. `numValue3` sisältää mittaustuloksen sekä yksikön ja `container6temp` sisältää linkin sensorin id-arvoon sekä mittauksen ajankohtaan.

```
<rscdfs:SR_Statement rdf:about="http://...#statement7">
 <rdf:subject rdf:resource="http://...#123456XZ24" />
 <rscdfs:predicate rdf:resource="http://...#temperature" />
 <rdf:object rdf:resource="http://...#numValue3" />
 <rscdfs:context rdf:resource="http://...#container6temp" />
</rscdfs:SR_Statement>

<rscdfs:NumericalValue rdf:about="http://...#numValue3">
 <rscdfs:value>70</rscdfs:value>
 <rscdfs:unit  rdf:resource="http://...#Celsius" />
 </rscdfs:NumericalValue>

<rscdfs:Context_SR_Container
    rdf:about="http://...#container6temp">
```

```
 <rscdfs:member
    rdf:resource="http://...#contextStatement7.sensorID" />
 <rscdfs:member
    rdf:resource="http://...#contextStatement5.time" />
</rscdfs:Context_SR_Container>
```

Seuraava väittämä toimii aivan samoin kuin edellä, mutta ainoastaan käsittelee toisen sensorin arvoja.

```
<rscdfs:SR_Statement rdf:about="http://...#statement11">
 <rdf:subject rdf:resource="http://...#123456XZ24" />
 <rscdfs:predicate rdf:resource="http://...#rotationSpeed" />
 <rdf:object rdf:resource="http://...#numValue8" />
 <rscdfs:context rdf:resource="http://...#container10temp" />
</rscdfs:SR_Statement>

<rscdfs:NumericalValue rdf:about="http://...#numValue8">
 <rscdfs:value>1500</rscdfs:value>
 <rscdfs:unit rdf:resource="http://...#PromptnessPerMinute" />
</rscdfs:NumericalValue>

<rscdfs:Context_SR_Container
    rdf:about="http://...#container10temp">
 <rscdfs:member
    rdf:resource="http://...#contextStatement11.sensorID" />
  <rscdfs:member
    rdf:resource="http://...#contextStatement5.time" />
 </rscdfs:Context_SR_Container>
```

Sisältää väittämän `contextStatement5.time` arvon olevan linkin `WorldEnvironmentTime4` takana, mikä pitää sisällään ajan arvon ja sen muodon.

```
<rscdfs:SR_Statement
    rdf:about="http://...#contextStatement5.time">
 <rdf:subject rdf:resource="http://...#WorldEnvironment" />
 <rscdfs:predicate rdf:resource="http://...#sysTime" />
 <rdf:object rdf:resource="http://...#WorldEnvironmentTime4" />
</rscdfs:SR_Statement>

<ontoTempMark:TempTempMark
    rdf:about="http://...#WorldEnvironmentTime4">
 <rscdfs:value>2004-09-28T23:50:12.578</rscdfs:value>
 <rscdfs:unit
     rdf:resource="http://...#XMLSchemaDateTime" />
</ontoTempMark:TempTempMark>
```

Seuraava sisältää tiedot sensorien id-arvoista.

```
<rscdfs:Context_SR_Container
    rdf:about="http://...#contextStatement11.sensorID">
 <rdf:type rdf:resource="http://...#SR_Statement" />
 <rdf:object rdf:resource="http://...#WorldEnvironment" />
 <rdf:subject rdf:resource="http://...#RP2345-M" />
</rscdfs:Context_SR_Container>

<rscdfs:Context_SR_Container
    rdf:about="http://...#contextStatement7.sensorID">
 <rdf:type rdf:resource="http://...#SR_Statement" />
 <rdf:object rdf:resource="http://...#WorldEnvironment" />
```

```
 <rdf:subject rdf:resource="http://...#KX2834-S" />
</rscdfs:Context_SR_Container>

 </rdf:RDF>
```

Edellä esitetty RscDF-dokumentti sisältää kaiken tiedon, mitä myös alkuperäinen XML-dokumentti sisälsi. Näin ollen kaksiosaista muunnosta käyttämällä, on semanttinen muunnos toteutettavissa myös dynaamisille resursseille.

Toisaalta edellä esitetty muunnos oli tehty ainoastaan pienelle ongelma-alueelle. Kuitenkin on huomattava muunnostavan laajennettavuuden tuomat mahdollisuudet. Tällöin kaksiosaisen muunnoksen pohjalle rakennettavien sovittimien on mahdollista muuntaa erilaisia heterogeenisia resursseja semanttisen webin standardien mukaiseksi.

8 Yhteenveto

Heterogeenisten sovellusten ja datalähteiden integrointi toistensa kanssa yhteensopiviksi järjestelmiksi on eräs suurimmista tämän päivän tietoyhteiskunnan haasteista. Tämän haasteen piiriin kuuluu myös teollisuuden eri laitteiden ylläpidon automatisointi. Ottaen huomioon useat erityyppiset informaatioresurssit ja dataformaatit, on tietojärjestelmien yhteensovittaminen erittäin tärkeä tutkimuksellinen haaste. (Khanna R. 2004)

Semanttinen web tarjoaa globaalin standardin tämän haasteen ratkaisemiseksi, mutta on kuitenkin liian myöhäistä siirtää käsin metadataa paikallisesta standardista globaaliin standardiin. GUN-ympäristön, mikä on semanttisen webin kaiken tutkimus ja kehittämisponnistelujen perimmäinen tarkoitus, tulee perustua universaaliin metodologiaan resurssien integraatiosta. Tällöin kaikenlaiset, niin rakenteeltaan kuin luonnoltaan, erilaiset resurssit saataisiin toimivaan yhteensopivasti.

Tutkielman tarkoituksena oli esitellä yleinen sovitinympäristö GAF, mikä mahdollistaa erilaisten resurssien yhteensovittamisen. Lisäksi tarkoituksena oli selvittää, kuinka pystytään muuntamaan sovittimien avulla XML-muotoista dataa semanttisesti rikkaampaan RscDF-muotoon. Tutkielma keskittyi yleisen sovitinympäristön suurimpaan haasteeseen, semanttiseen muunnokseen, minkä parista ei ole aikaisemmin saatu tarpeeksi tutkimustuloksia. (Sperberg-McQueen & Miller 2004)

Tutkielman tuloksena esiteltiin kaksiosainen muunnos, minkä avulla saadaan poistettua useita semanttiseen muunnokseen liittyviä haasteita. Tutkielma esitteli myös prototyyppisovelluksen, minkä avulla testattiin kaksiosaisen muunnoksen soveltuvuutta semanttisen muunnoksen suorittamiseksi. Lisäksi esiteltiin sovitin, minkä tarkoituksena oli tehdä muunnos kanonisesta XML^0-muodosta RscDF-muotoon.

Tutkielmassa ei otettu kantaa, kuinka muunnettu data pystytään muuntamaan takaisin semanttisesti rikkaasta muodosta alkuperäiseen muotoon. Tämä olisikin erittäin hyödyllinen jatkotutkimuksen aihe. Lisäksi myös kanonisten XML^0-rakenteiden rakentaminen eri ongelma-alueille olisi tarpeellinen jatkotutkimuksen aihe. Lisäksi tarve

yhteisen kirjaston perustamisesta näille rakenteille olisi hyödyllistä tässä yhteydessä kartoittaa.

Lähteet

Adler S., Berglund A., Caruso J., Deach S., Graham T., Grosso P., Gutentag E., Milowski A., Parnell S., Richman J. & Zilles S. 2001, Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2001/REC-xsl-20011015/>, 14.3.2006.

Becket D. (edit.) 2004, RDF/XML Syntax Specification (Revised), W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>, 7.2.2005.

Benjamins R., Contreras j., Corcho O. & Gómez-Pérez A. 2002, The Six Challenges for the Semantic Web, KR-2002 Workshop on Semantic Web, Toulouse, Ranska.

Berglund A., Boag S., Chamberlin D., Fernández M. F., Kay M., Robie J. & Siméon J. (edit.) 2007, XML Path Language (XPath) 2.0, W3C Recommendation, Saatavilla www-muodossa, <http://www.w3.org/TR/2007/REC-xpath20-20070123/>, 15.5.2008.

Berners-Lee T., Hendler J., Lassila O. 2001, *The Semantic Web,* Scientific American, 5/2001, 34-43.

Boag S., Chamberlin D., Fernández M. F., Florescu D., Robie J. & Siméon J. (edit.) 2007, XQuery 1.0: An XML Query Language, W3C Recommendation, Saatavilla www-muodossa, <http://www.w3.org/TR/2007/REC-xquery-20070123/>, 15.5.2008.

Brickley D. & Guha R.V. (edit.) 2004, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/rdf-schema/>, 7.2.2005.

Clark J. (edit.) 1999, XSL Transformations (XSLT) Version 1.0, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/1999/REC-xslt-19991116>, 12.2.2008

Clark J. & DeRose S. (edit) 1999, XML Path Language (XPath) Version 1.0, W3C Recommendation, Saatavilla www-muodossa, <http://www.w3.org/TR/1999/REC-xpath-19991116>, 18.2.2008.

DAML-Time, DAML-Time-projektin kotisivut, <http://www.cs.rochester.edu/~ferguson/daml/>, 14.4.2005.

DeRose S, Maler E. & Orchard D. (edit.) 2001, XML Linking Language (XLink) Version 1.0, W3C Recommendation, Saatavilla www-muodossa, < http://www.w3.org/TR/2001/REC-xlink-20010627/>, 13.5.2008.

DeRose S., Daniel Jr. R., Grosso P., Maler E., Marsh J. & Walsh N. (edit.) 2002, XML Pointer Language (XPointer), W3C Working Draft, Saatavilla www-muodossa, <http://www.w3.org/TR/2002/WD-xptr-20020816/>, 13.5.2008.

Dublin Core, Dublin Core Metadata Initiative viralliset kotisivut, <http://dublincore.org>, 14.4.2005.

Farrar S., Lewis W. & Langendoen T. 2002, A Common Ontology for Linguistic Concepts, Proceedings of the Knowledge Technologies Conference, Seattle, Washington, Maaliskuu 10-13.

Harold E. R. 1999, “XML Bible”, IDG Books Worldwide, USA.

Hazaël-Massieux D. & Connolly D. 2005, Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Team Submission, Saatavilla www-muodossa <http://www.w3.org/TeamSubmission/2005/SUBM-grddl-20050516/>, 21.6.2006.

Heflin J. (edit.) 2004, “OWL Web Ontology Language Use Cases and Requirements”, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/webont-req/>, 8.2.2005.

JUHTA 2008, Julkisen hallinnon tietohallinnon neuvottelukunnan sivusto: JHS 143 Asiakirjojen kuvailun ja hallinnan metatiedot, Saatavilla www-muodossa <http://www.jhs-suositukset.fi/suomi/jhs143>, 20.5.2008.

Kansalliskirjasto 2008, Kansalliskirjaston internetsivusto, Saatavilla www-muodossa <http://www.kansalliskirjasto.fi/julkaisuala/dublincore.html>, 20.5.2008.

Kay M. (edit.) 2007, XSL Transformations (XSLT) Version 2.0, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2007/REC-xslt20-20070123/>, 15.4.2008.

Kaykova O., Khriyenko O., Kononenko O., Terziyan V., Zharko A. 2004, “Proactive Self-Maintained Resources in Semantic Web”, Eastern-European Journal of Enterprise Technologies, Vol. 2, Issue 1, Kharkov, Ukraina, 4-16.

Kaykova O., Khriyenko O., Kovalainen M., Zharko A. 2004, “Visual Interface for Adaptation of Data Sources to Semantic Web”, Proceedings of the IASTED International Conference on Software Engineering (SE 2004), Helmikuu 17-19, Innsbruck, Itävalta, ACTA Press, 544-547.

Kaykova O., Khriyenko O., Kovtun D., Marttinen J., Naumenko A., Terziyan V., Tsaruk Y., Zharko A. 2004a, General Adaptation Framework: Framework for Semantic Adaptation of Maintenance Resources, Technical Report (Deliverable D 1.2), SmartResource, Agora Center, Jyväskylän Yliopisto.

Kaykova O., Khriyenko O., Kovtun D., Marttinen J., Naumenko A., Nikitin S., Terziyan V., Tsaruk Y., Zharko A. 2004b, “SmartResource Prototype Environment, v. 1.0, Adaptation Stage”, Technical Report (Deliverable D 1.3), SmartResource, Agora Center, Jyväskylän Yliopisto.

Kaykova O., Khriyenko O., Kovtun D., Naumenko A., Terziyan V., Zharko A. 2005a,“ General Adaption Framework: Enabling Interoperability for Industrial Web Resources”, International Journal on Semantic Web and Information Systems, Idea Group.

Kaykova O., Khriyenko O., Naumenko A., Terziyan V., Zharko A. 2005b, RSCDF: A Dynamic and Context-Sensitive Metadata Description Framework for Industrial Resources, Eastern-European Journal of Enterprise Technologies, Vol. 3, No. 2, 55-78.

Khanna R. 2004, “Top Challenges in Integration Projects”, White Paper, ITtoolbox EAI – Enterprise Application Integration section of ITtoolbox, Availabe online <http://hosteddocs.ittoolbox.com/WT052004.pdf>, 12.4.2005.

Khriyenko O. & Terziyan V. 2005, Context Description Framework for the Semantic Web, Saatavilla www-muodossa <http://www.cs.jyu.fi/ai/papers/Context-2005.pdf>, 25.7.2005.

Klyne G. & Carroll J. 2004, “Resource Description Framework (RDF): Consepts and Abstract Syntax”, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, 14.4.2005.

Koivunen M. & Miller E. 2001, ”W3C Semantic Web Activity”, Proceedings of the Semantic Web Kick-off Seminar in Finland, Saatavilla www-muodossa <http://www.w3.org/2001/12/semweb-fin/w3csw>, 4.2.2005.

Krupnikov K.A., Thompson H. 2001, Data binding using W3C XML Schema Annotations, Talk at XML 2001, Orlando, Joulukuu, Saatavilla www-muodossa <http://www.idealliance.org/papers/xml2001/papers/pdf/06-03-04.pdf>, 21.6.2005.

Malhotra A., Melton J. & Walsh N. (edit) 2007, XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/>, 13.5.2008.

Malucelli A. & Oliveira E. 2003, "Ontology-Services to Facilitate Agents’ Interoperability", PRIMA'03 - 6th Pacific Rim International Workshop On Multi-Agents, LNAI, Springer-Verlag. Marraskuu 7-8, Soul, Korea.

Manola F., Miller E. 2004, “RDF Primer”, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>, 15.4.2005.

McGuinness D., Harmelen F. 2004, ”OWL Web Ontology Language Overview”, W3C Recommendation, Saatavilla www-muodossa <http://www.w3.org/TR/2004/REC-owl-features-20040210/>, 16.4.2005

Nachimovsky A. & Myers T. 1999, Professional Java XML Programming with Servlets and JSP, Wrox Press, USA.

Ogbuji U. 2001, Thinking XML: Basic XML and RDF techniques for knowledge management, Part 1: Generate RDF using XSLT, IBM developerWorks, Saatavilla www-muodossa <http://www-106.ibm.com/developerworks/library/x-think4/>, 31.6.2005.

Paolucci M., Sycara K. 2003, “Autonomous Semantic Web Services”, IEEE Internet Computing, Vol. 7, Issue 5, 34-41.

PaperIXI, PaperIXI-projektin kotisivut, Saatavilla www-muodossa <http://pim.vtt.fi/paperixi/>, 25.5.2008.

Phillips L. A. 2000, “Using XML”, Special Edition, QUE, Indianapolis.

Ryabov V., Terziyan V. 2003, Industrial Diagnostics Using Algebra of Uncertain Temporal Relations, Proceedings of the 21-st IASTED International Multi-Conference on Applied Informatics (AI-2003), Helmikuu 10-13, Innsbruck, Itävalta, ACTA Press, pp.351-356.

Simpson J.E. 2001, XPath and XPointer: Locating Content in XML Documents, O’Reilly, USA.

Sperberg-McQueen C. M., Huitfeldt C. and Renear A. 2001, Meaning and interpretation of markup, Markup Languages: Theory & Practice 2.3, 215-234, Saatavilla www-muodossa <http://www.w3.org/People/cmsmcq/2000/mim.html>, 21.6.2005.

Sperberg-McQueen C.M., Miller E. 2004, On mapping from colloquial XML to RDF using XSLT, Proceedings of W3C Extreme Markup Languages 2004, Elokuu 3, Montreal, CA.

Swick R. R. & Henry S. T. edit. 1999, The Cambridge Communiqué W3C NOTE, Lokakuu 7, Saatavilla www-muodossa <http://www.w3.org/TR/1999/NOTE-schema-arch-19991007>, 21.6.2005.

Terziyan V. 2003, Semantic Web Services for Smart Devices in a Global Understanding Environment, R. Meersman & Z. Tari (Eds.), On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops: Proceedings of the OTM Confederated

International Workshops, HCI-SWWA, IPW, JTRES, WORM, WMS, and WRSM 2003 Lecture Notes in Computer Science (2889), Springer, 279-291.

Terziyan V. 2005, “SmartResource: Utilizing Semantic Web Services to Monitor Industrial Devices”, Proceedings of the XML Finland 2005: XML - the Enabling Technology for Integrating Business Processes, Maaaliskuu 8-9, Pori.

Terziyan V., Khriyenko O., Kaykova E., Zharko A. & Naumenko A. 2005, RSCDF: A Dynamic and Context Sensitive Metadata Description Framework for Industrial Resources, Eastern-European Journal of Enterprise Technologies, 3.

Vorthman S., Buck L. 2000a, Schema Adjunct Framework: Executive Summary, Helmikuu 24, Saatavilla www-muodossa <http://www.tibco.com/software/standards_support/xmlresources/exec_summary.html>, 21.6.2005.

Vorthmann, S and Buck L. 2000b, Schema Adjunct Framework: Draft Specification, Helmikuu 24, Saatavilla www-muodossa <http://www.tibco.com/software/standards_support/xmlresources/spec.html>, 21.6.2005.

XR, XR-projektin kotisivut, XML-to-RDF transformation format, Saatavilla www-muodossa <http://w3future.com/xr/>, 19.4.2006.

Liite 1: XML- ja XML^0 – dokumenttien skeemat

Skeema XML_1_Skeema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema             xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:alrm="http://www.metso.com/Alarm"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.metso.com/Alarm">
<xs:element name="State">
<xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element name="Measurement">
<xs:complexType>
<xs:sequence>
<xs:element name="ParamType" type="xsd:string"/>
<xs:element name="Units" type="xsd:string"/>
<xs:element name="Value" type="xsd:string"/>
<xs:element name="Sensor" type="xsd:string"/>
<xs:element name="SensorID" type="xsd:ID"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="Time" type="xs:dateTime"/>
</xs:complexType>
</xs:element>
<xs:element name="Measurement">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="Value">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="alrm:Value">
<xs:attribute name="ParamType" type="xs:string"/>
<xs:attribute name="Units" type="xs:string"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="Sensor">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="alrm:SensorDescription">
<xs:attribute name="SensorID" type="xs:ID"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:simpleType name="SensorDescription">
<xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="Value">
<xs:restriction base="xs:string"/>
</xs:simpleType>
```

```
</xs:schema>
```

Skeema XML[0]Skeema.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema             xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:alrm="http://www.metso.com/Alarm"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.metso.com/Alarm">
<xs:element name="State">
<xs:complexType>
<xs:sequence maxOccurs="unbounded">
<xs:element name="Measurement">
<xs:complexType>
<xs:sequence>
<xs:element name="ParamType">
<xs:simpleType>
<xs:restriction base="xsd:string">
<xs:maxLength value="30"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Units">
<xs:simpleType>
<xs:restriction base="xsd:string">
<xs:maxLength value="20"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name="Value">
<xs:simpleType>
<xs:restriction base="xsd:string">
<xs:maxLength value="20"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Sensor">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="alrm:sensorDescription">
<xs:attribute name="sensorID"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="Time" type="xs:dateTime"/>
</xs:complexType>
</xs:element>
<xs:simpleType name="sensorDescription">
<xs:restriction base="xsd:string">
<xs:maxLength value="30"/>
</xs:restriction>
</xs:simpleType>
```

```
</xs:schema>
```