TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Mika Rajala

XML OHJELMISTON TOIMINNAN MÄÄRITTELIJÄNÄ

Työn ohjaaja Jari Mikkolainen
Työn teettäjä Wapice OY
Tampere 2008

TAMPEREEN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

Ohjelmistotekniikka

Rajala, Mika XML Ohjelmiston toiminnan määrittelijänä

Tutkintotyö 43 sivua

Työn ohjaaja

Työn teettäjä Wapice OY

Kesäkuu 16.1.2008

Hakusanat XML, Konfigurointi

TIIVISTELMÄ

XML on nykyään hyvin yleisesti käytetty tiedostoformaatti ja sitä tuetaan eri muodoissa monissa eri ohjelmissa. Alun perin hyvin löyhästi määritelty formaatti mahdollistaa hyvinkin monimutkaisien rakenteiden kuvaamista. Ohjelmistojen toiminnan määrittely tai tiedon lataaminen on hyvin usein XML muodossa, usein turhaan. XML tiedosto sisältää verrattain paljon ylimääräistä varsinaisen tiedon lisäksi, jolloin huonosti suunniteltu muoto aiheuttaa helposti tiedoston koon merkittävän kasvun tiedostoon sisällytetyn tiedon kasvaessa. Tiedoston koko vaikuttaa tiedoston viemän tilan lisäksi sen käsittelyyn kuluvaan aikaan. Tiedoston muodon suunnitteluun kannattaakin käyttää aikaa, nopeimmin suunnittelun hoitaa henkilö joka tietää perusteellisesti kuvattavan asian ja ymmärtää eri XML tekniikoita. Tällainen henkilö pystyy luomaan ensin määrittelevän DTD tai Schema tiedoston, jonka pohjalta varsinainen XML tiedosto luodaan.

Tiedon käsittelyssä tulee välttää ylenmääräistä XPath kyselyiden käyttöä. Kysely on helppo tehdä, mutta kovin hidas, varsinkin kun tiedoston sisältämän tiedon määrä kasvaa. XPath kysely on kuten SQL kysely, parhaimmillaan kun haetaan iso joukko tietoa. Yksittäisen tiedon hakeminen XPath kyselyllä tulee suorittaa vain harvoissa tapauksissa.

Tiedoston ja muiden tekniikoiden lisäksi tulee kiinnittää huomiota ohjelman yhteydessä käytettävään XML käsittelyn tekevään kirjastoon. Aluksi tulee kartoittaa ne toiminnot joita XML käsittelyssä vaaditaan. Näitä ovat mm XPath, XML DOM sekä XML Schema/DTD. Vaikka tekniikat ovatkin hyvin yleisiä, eivät kaikki XML kirjastot niitä tue. Toisaalta, kun tehdään ohjelmistoa johonkin erikoisempaan ympäristöön, tulee selvittää mitkä kirjastot voidaan kääntää ko. ympäristöön.

XML kirjastot tarjoavat hyvin erilaisia rajapintoja XML käsittelyyn, jolloin onkin kannattavaa luoda XML kirjaston päällä rajapinta josta tarvittavat käskyt saadaan ajettua. Rajapinta kannattaa pitää eri projekteissa samana, jolloin riittää kun yksi työntekijä tutustuu tai tietää käytettävään XML kirjastoon.

TAMPERE UNIVERSITY OF APPLIED SCIENCES

Computer technology program

Software engineering

Rajala, Mika	XML definition of software behavior
Thesis	43 pages
Thesis supervisor	
Comissioning company	Wapice OY
May	16.1.2008
Keywords	XML, Configuration

ABSTRACT

XML is currently one of the most used file-formats and it is supported in various forms in many different programs. Originally very loosely defined format allows the definition of complex structures. Many programs have their behavior defined or have data in XML form, usually this is futile. An XML file contains lots of overhead for the data stored, which leads to a rapid growth in file size, upon adding new data. Not only does the size of the file matter in the space it takes, but it also matters when handling the data contained. Therefore, it is imperative that the XML file is properly designed. The most suitable person for this task is the one that understands thoroughly the aspects of the thing that is being described and understands the various XML techniques. This kind of person can first create a DTD- or XML Schema file that defines the structure of the XML file. This kind of approach is better, since a definition of the structure is easier to understand and it also makes it clear what values are possible in different locations.

When handling the data of an XML file, the use of XPath queries should be avoided. A query is easy to implement, but it is very slow, problems usually occur later, when data is added. XPath query is as a SQL query is, at best when acquiring a big load of data. Getting a single result from a huge group is something to be done rarely.

Apart from the file and other techniques, attention should be paid on the XML parser to be used. At first, the requirements of the XML handling should be charted. Usually these consist of XPath, XML DOM and XML Schema/DTD. Although all of these are fairly common, not all XML parsers support them, especially when the program is to run on a special environment. On a non-standard environment, the range of available XML parsers can be limited.

XML parsers offer various interfaces to do XML handling with. Therefore, it is better to create an interface through which the XML handling is done. The interface should be kept same thought for different projects, so that it is enough when one employee studies or knows the XML library to be used.

SISÄLLYSLUETTELO

1 Johdanto 8
2 XML - eXtensible Markup Language 9
2.1 XML-formaatti 10
2.2 XML DOM 14
2.3 XML validointi 15
2.3.1 XML DTD 16
2.3.2 XML Schema 19
2.4 XPath 21
2.5 XSLT 23
3 XML käsittely 25
3.1 Yleistä XML käsittelystä 26
3.2 XML tiedoston rakenteen suunnittelu 27
3.3 XML rajapintaluokka 28
3.3.1 Yleinen määrittely 29
3.3.2 Sisäisesti käytettävät funktiot 30
3.3.3 Ulkoapäin käytettävät funktiot 31
3.3.4 XPath kyselyiden käyttö 34
3.3.5 Tiedon tallentaminen luokkarakenteeseen 35
3.3.6 XML kirjastojen vertailua 37
3.3.7 Yhteenveto XML kirjastoista 39
4 CANopen konfiguraatio 40
4.1 CAN 40
4.2 CANopen 40
4.3 XML tiedostoformaatin muodostaminen 41
4.3.1 Objektikirjasto 42
4.3.2 PDO määrittely 42
4.3.3 SDO määrittely 43
4.3.4 Yhdyskäytävän määrittely 44
4.4 Tiedoston toteuttaminen 44
4.5 Tietojen hakeminen 47

4.5.1 Yhdyskäytävän lataaminen 47
4.5.2 Objektikirjaston lataaminen 48
4.5.3 PDO tietojen lataaminen 49
4.5.4 SDO tietojen lataaminen 50
4.6 Yhteenveto 50
5 Lähteet 51

SYMBOLILUETTELO

XML	eXtensible Markup Language, erityinen tiedostoformaatti.
XML DOM	Document Object Model, XML tiedoston pohjalta tehty objektimalli.
XPath	Komento jolla haetaan XML DOM rakenteesta tietoa.
Elementti	XML tiedostossa oleva rakenne.
Muuttuja	XML tiedostossa olevaan rakenteeseen tallennettu arvo-nimi pari.
Solmu	XML DOM mallin objekti.
XSLT	XML tiedoston muunnos muuhun muotoon käyttäen erityistä skriptiä.
XML DTD	Document Type Definition, XML tiedoston rakenteen määrittelytiedosto.
XML Schema	Kuten XML DTD, mutta kehittyneemmillä komennoilla.
SQL	Server Query Language, tietokannoista tietojen hakuun tehty kieli.
CAN	Controlled Area Network. Väyläpohjainen ratkaisu tiedonsiirtoon.
CANopen	CAN ratkaisun päälle rakennettu, OSI-mallin verkkokerroksen protokolla.
OSI-malli	Tiedonsiirtoprotokollien kuvausmalli.
Objektikirjasto	CANopen järjestelmän tiedonsiirtoa varten tarkoitettu muistiavaruus.
PDO	Process Data Object, yhdeltä kaikille tyyppinen datansiirto CANopen järjestelmässä
SDO	Service Data Object. kahden osapuolen välinen tiedonsiirto CANopen järjestelmässä.
Parseri	Ohjelman toiminnallinen osa joka tutkii tekstimuotoista tietoa, etsii siitä joitain tiettyjä rakenteita ja muodostaa niiden mukaan käsiteltävää tietoa.
Enkoodaus	Tiedon muokkaus binääriseen muotoon jostain toisesta muodosta, esimerkiksi tekstistä.
W3C	The World Wide Web Consortium on tehnyt mm XML määrittelyt, mutta myös muita sovellusten välillä olevia yhteisiä tiedonjakomenetelmiä ja niihin liittyen työkaluja.
HTML	HyperText Markup Language on XML formaattia muistuttava formaatti, joka nettiselaimessa muodostetaan graafiseksi kokonaisuudeksi.
Qt	Trolltechin valmistama kirjastopaketti, joka sisältää mm käyttöliittymäkirjastoja sekä XML kirjastoja.
WPF	Windows Presentation Foundation on Microsoftin käyttöliittymiä varten tehty kirjastopaketti.

1 Johdanto

Tässä kappaleessa esitellään dokumentin sisältö, sekä esitellään XML tekniikkaa yleisesti.

Tämä työ käsittelee XML dokumenttien käyttöä ohjelmien määrittelyssä ja tiedonlähteenä. Työssä esitellään XML tiedostojen yleisiä käsittelymenetelmiä, kuten XSLT muunnokset, XML Schema, XPath ja XML DOM.

Työssä pyritään tutkimaan kuinka XML rakenteesta saadaan kuvaava sekä nopeasti käsiteltävä. Työssä myös luodaan XML parsijan päälle luokka jonka kautta saadaan XML toiminnot toimimaan samoin, käytetystä XML kirjastosta riippumatta. Lisäksi kokeillaan tehdä XML rajapinta käyttäen eri XML kirjastoja, sekä tutkitaan eroavuuksia libxml2 ja Qt XML kirjastojen välillä.

Viimeisessä kappaleessa esitellään CANopen viestinnän konfigurointi XML tiedostolla. Suunnitellaan varsinainen XML tietorakenne CANopen järjestelmän pohjalta, sekä luodaan tavat lukea tietoa XML rakenteesta.

XML ja siitä johdetut tiedostomuodot ovat nykyisin hyvin yleisiä, nimi XML on lyhenne sanoista eXtensible Markup Language Uusimpia XML formaatista kehitettyjä muotoja edustaa XAML, eXtensible Application Markup Language, jolla voidaan tehdä käyttöliittymiä WPF, Windows Presentation Foundation, kanssa.

XML-formaattia käytetään hyvin paljon tiedon välittämiseen erinäisten ohjelmien välillä Internetin välityksellä, käyttötarkoitus johon se oli alun perin suunniteltu. XML-formaatin mukaisia tiedostoja käytetään mm. myös teollisuudessa erilaisten tietokoneohjattujen laitteiden asetusten tallentamiseen. Qt Designer ohjelmalla tehdyt käyttöliittymät tallennetaan XML muotoon, josta ne muutetaan erillisellä parserilla kääntäjää varten käännettäväksi koodiksi. Jopa tämä insinöörityö on tallennettu XML formaatissa.

XML ei siis ole pelkkään tiedonjakoon webbisovellusten välillä. XML ei myöskään ole tiedon varastointiin tarkoitettu tietokantaformaatti. Yleisin virhekäsitys on se,

että XML formaatilla ohjelmoidaan jotenkin, osa tästä johtuu siitä, että XML tiedostojen käsittelyyn liitetään usein toimintoja jotka ovat ohjelmointia, esim. XML Schemat /6 s.18-21/.

Paljon käytetty XML tarvitsee tarkempaa ymmärtämistä sen sisäisestä toiminasta hyödyn saamiseksi käyttämällä pienin tarvittava määrä resursseja. Väärin tai huonosti käytettynä XML voi olla merkittävä tekijä eri ohjelmiston ominaisuuksissa, kuten suorituskyvyssä. Ymmärrystä formaattiin tulee olla sekä määrittely-, suunnittelu- että toteutusvaiheessa. XML:n paikka ei ole kaikkialla, vaikka sen tuki olisikin hyvä olla.

2 XML - eXtensible Markup Language

Tässä kappaleessa on koottu muutamia tärkeimpiä asioita XML:n ja sen käsittelyyn liittyen. Suurin osa näistä on koottu lähteen 1 eri osioista.

Perusajatuksena XML-formaatissa on tiedon säilyttäminen ja kuljettaminen. XML ei pakota käyttäjää käyttämään jotain tiettyä rakennetta tai nimiä, eikä siten pelkkä XML-tiedosto tee mitään. Tämä mahdollistaa XML muotoisen datan käsittelyn myös ohjelmilla, jotka pystyvät käsittelemään vain pelkkää tekstiä.

Kuten lähteessä 4 sivulla 87 todetaan XML tiedon ymmärrettävyydestä, XML tiedoston eräs hyvä ominaisuus on sen luettavuus, ihmiselle ja koneelle. Tallennettu tieto yleensä kuvaa hyvin itseään ja siitä voi päätellä usein mistä on kyse. Kuvassa 1 esitetystä XML tiedostosta tämä voidaan helposti todeta.

XML-formaatin tekstimuotoisena talletettu tieto voidaan lukea helposti monissa eri järjestelmissä, näin yksinkertaisessa muodossa tallennettu tieto voidaan helposti lukea eri järjestelmissä, mitkä muuten tallentaisivat tietoa yhteensopimattomissa muodoissa. Tekstimuotoinen tieto voidaan siis tarjota suuremmalle määrälle käyttäjiä samassa muodossa.

```
1  <Tehtävä>
2      <Nimi>
3          Tee insinöörityö
4      </Nimi>
5  </Tehtävä>
6
```

Kuva 1. XML formaatin mukaista tietoa

XML tiedosto vie kuitenkin melko ison määrän tilaa, koska säilytettävä tieto sisältää melko paljon muuta tekstiä sen ympärillä. Nämä tekstit ovat hyvin itseään toistavia, jolloin tiedoston pakkaamisella on tavanomaista nähdä tiedostokoon tippuminen 90 %:lla /4 s.88/ tai suuremmalla määrällä.

2.1 XML-formaatti

XML Formaatin määrittelyn lähteenä on käytetty lähdettä 1, joka on tehnyt alkuperäisen XML dokumenttien määrittelyn.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <Tehtävä>
3      <Nimi>
4          Tee insinöörityö.
5      </Nimi>
6  </Tehtävä>
7
```

kuva 2. xml tiedosto

Tekstimuotoisen datan yhteensopivuus eri järjestelmien välillä ei aina toimi heti. Tämä johtuu erilaisista tekstien enkoodauksista. XML tiedoston alussa määritellään mitä tekstienkoodausta käytetään erityisessä tietokentässä. Tämä kenttä ei tosin ole välttämätön, mutta auttaa XML-muotoisen tiedon yhteensopivuudessa eri järjestelmien välillä. Tietokenttä ilmenee kuvassa 2 ensimmäisellä rivillä. Tietokentässä määritellään myös tiedostossa käytetty XML-versio.

XML tiedostot muodostuvat pääasiassa elementeistä. Jokaisessa XML tiedostossa pitää olla yksi juurielementti, joka kuvassa 2 esitetyssä tiedostossa on elementti nimeltä *Tehtävä*. Elementin alku ilmoitetaan erityisellä aloitusmerkinnällä, juurielementin aloitusmerkintä on kuvassa 2 <*Tehtävä*>. Lopetusmerkintä alkaa aina merkeillä </ ja loppuu merkkiin >. On olemassa myös erityinen merkintä, jossa määritellään aloitus ja lopettaminen samassa. tällainen merkintä alkaa merkillä < ja loppuu merkkeihin />. XML rakenteessa pitää suljettavan merkinnän olla se mikä on viimeksi avattu ja jota ei vielä ole suljettu. Lisäksi, kaikki merkinnät pitää sulkea, huomioimatta alussa olevaa tietokenttää, joka ei oikeastaan edes ole XML formaatin mukaista.

Elementti voi sisältää toisia elementtejä ja tietoa. Elementin sisällä aloitettuja elementtejä kutsutaan ko. elementin lapsielementeiksi, kaikki elementit paitsi juurielementti ovat jonkin elementin lapsielementtejä. Elementtiä, jonka sisällä lapsielementti on, kutsutaan lapsielementin näkökulmasta isäntäelementiksi. Elementeillä voi olla myös sisaruselementtejä, näillä elementeillä on yhteinen isäntäelementti. Jokaisella elementillä, lukuun ottamatta juurielementtiä, on vain ja ainoastaan yksi isäntäelementti. XML dokumentti muodostaa eräänlaisen puurakenteen, jossa juurielementti on aloituspiste. Kuvassa 3 esitellään jo hieman monipuolisempi XML muodon mukainen tiedosto.

Elementin ja sen merkintöjen välissä olevassa tiedon muodossa on tiettyjä rajoituksia. Elementin nimen tulee alkaa joko kirjaimella, alaviivalla tai kaksoispisteellä, mutta se ei saa olla millään kirjainkokoyhdistelmällä kirjoitettu kirjainsarja *xml*. Loput merkit saavat olla aloittavien merkkien lisäksi numeroita, pisteitä tai väliviivoja. Merkintöjen välissä oleva tieto saa sisältää kaikkia merkkejä, lukuun ottamatta pienempikuin, suurempikuin-, et-, lainaus- tai heittomerkkiä. Näiden merkkien tilalle on määritelty erikseen tietyt koodit joilla merkit saadaan ilmestymään lopulliseen tietoon. Nämä merkit ovat &lt pienempikuin-, &gt suurempikuin-, &amp et-, &quot lainaus-, &apos heittomerkki. Toisin kuin HTML muodossa, XML säilyttää useat perättäiset välilyöntimerkit, toisena erityishuomiona, XML muotoon rivinvaihto on tallennettuna pelkästään LF

merkkinä, tämä koskee vain tallennettua tietoa. Kuvassa 4 on käytettynä &lt merkintää rivillä 25.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <Tehtävä>
3      <Nimi>
4          Tee insinöörityö.
5      </Nimi>
6      <Aihe>
7          XML:n käyttö ohjelmiston tietolähteenä.
8      </Aihe>
9      <Aikataulu>
10         <Aloitus>
11             <Päivä>
12                 9
13             </Päivä>
14             <Kuukausi>
15                 4
16             </Kuukausi>
17             <Vuosi>
18                 2008
19             </Vuosi>
20         </Aloitus>
21         <Valmis>
22             <Päivä/>
23             <Kuukausi/>
24             <Vuosi/>
25         </Valmis>
26     </Aikataulu>
27 </Tehtävä>
28
```

kuva 3 Laajennettu XML tiedosto

XML syntaksi tukee myös kommentteja, kommentit asetetaan erityiseen kommentti-elementiin, joka alkaa merkkisarjalla <!-- ja loppuu merkkisarjaan -->. Näiden merkkisarjojen sisälle kirjoitettua tekstiä ei varsinaisesti käsitellä XML parserissa, jolloin ne saavat sisältää mitä tahansa. Jotkin XML parserit, kuten libxml2, mahdollistavat kommenttien käsittelyn /3/.

XML formaatissa tietoa voidaan tallentaa sekä elementin sisältöön, nimeen, erilliseen muuttujaan että elementtien isäntä-lapsi tai sisaruusjärjestykseen järjestykseen, mutta ei elementtien sisältämien muuttujien järjestykseen. XML muuttuja asetetaan elementtiin aloitusmerkinnän sisälle. Yksittäinen elementti voi sisältää minkä tahansa määrän muuttujia, mutta samannimistä muuttujaa ei saa olla toiseen kertaan yhdessä elementissä. Kuvassa 4 on esiteltynä xml muuttuja rivillä

9. Kuten taas voimme huomata, XML näyttää hyvin paljon siltä mitä se oikeasti tarkoittaa.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Tehtävä>
    <Nimi>
        Tee insinöörityö
    </Nimi>
    <Aihe>
        XML:n käyttö ohjelmiston tietolähteenä
    </Aihe>
    <Aikataulu Tila="Kesken">
        <Aloitus>
            <Päivä>
                9
            </Päivä>
            <Kuukausi>
                4
            </Kuukausi>
            <Vuosi>
                2008
            </Vuosi>
        </Aloitus>
        <Valmis>
            <Päivä/>
            <Kuukausi/>
            <Vuosi>
                &lt 2009
            </Vuosi>
        </Valmis>
    </Aikataulu>
</Tehtävä>
```

Kuva 4 XML tiedosto.

XML muuttujan nimeämisessä täytyy ottaa huomioon samat säännöt kuin elementin nimeämisessä. Muuttujan arvo sijoitetaan yhtäläisyysmerkillä ja varsinainen arvo on joko lainaus- tai heittomerkeissä. Mikäli varsinainen tallennettava arvo sisältää jonkin tyyppisiä em. merkkejä, voidaan muuttujan arvo ympäröidä toisella tyypillä, jolloin niiden sisällä voidaan käyttää vapaaksi jäänyttä tyyppiä. Turvallisempaa on kuitenkin käyttää aikaisemmin määriteltyjä erityisiä koodeja, jotka esiteltiin tässä dokumentissa sivulla 11.

Tekstimuotoisen tiedon tallentaminen onnistuu elementtien sisäisellä tekstillä tai muuttujilla, mutta niissä on kuitenkin oleellisia eroja. Muuttuja ei voi sisältää

lapsielementtejä, jolloin laajennusten tekeminen vaikeutuu. Muuttujia tulisi käyttää kuvaamaan sitä mitä varsinainen tieto koskee.

XML rakenteessa on myös mahdollista käyttää nimiavaruuksia. Termi on käytännössä sama kuin ohjelmoinnin termi nimiavaruus. Jossain tietyssä nimiavaruudessa olevat nimet eivät mene päällekkäin toisessa nimiavaruudessa olevien samojen nimien kanssa. Nimiavaruus määritellään elementin nimessä kaksoispisteen etupuolelle, esim. *koti:puhelin*. Näin luodaan nimiavaruuteen *koti* elementti jonka nimi on *puhelin*.

2.2 XML DOM

Alkuperäinen XML DOM määrittelyn on tehnyt lähteen 1 opasteet tehnyt W3C, opasteista on kerätty tähän oleellisimpia kohtia XML DOM:sta.

Kuten aikaisemmin todettiin, XML muodostaa eräänlaisen puurakenteen, jossa on yksi juuri josta kaikki muut jollain tapaa periytyvät. Tämä puurakenne voidaan lukea parsimalla XML DOM(Document Object Model) rakenteeksi. Tällä tavalla luettu XML rakenne voidaan tallentaa muistiin ja sitä voidaan käsitellä helposti.

XML DOM rakenteen käsittelyä käytetään hyvin paljon eri XML tekniikoissa, joissa on tarkoitus tutkia rakenteen eri paikkoja. DOM puun varsinainen rakenne määrittyy kuitenkin käytetyn kirjaston toiminnan perusteella /3/ /4/. Tämä saattaa luoda hämmennystä toteutusvaiheessa.

Luettu rakenne koostuu solmuista, jotka tavallaan näkyvät XML tiedoston rakenteessa. Solmuja ovat kaikki elementit, mutta myös niiden sisäiset tekstit ja muuttujat. Eri parsereilla luetut solmut saattavat sisältää erilaista tietoa, mutta perusajatuksena jokaisella solmulla on tieto nimestään, arvostaan, isäntäsolmustaan, seuraavasta ja edellisestä sisarsolusta, ensimmäisestä lapsisolusta sekä viimeisestä lapsisolusta. Yleisesti on käytössä myös tieto solmun nimiavaruudesta.

Näiden tietojen pohjalta voidaan mistä tahansa kohtaa solmurakennetta päätyä johonkin toiseen paikkaan. Hyvin muodostettu puurakenne mahdollistaa myös jonkin tietyn osion kopioinnin rekursiivisesti, toiseen XML dokumenttiin.

2.3 XML validointi

W3C on määritellyt XML-validoinnin, tässä kappaleessa esitellyt asiat ovat peräisin W3C:n ohjesivustolta /1/.

Termillä “Well Formed XML” /1/ tarkoitetaan XML rakennetta joka on muodostettu siten että se kattaa tietyt säännöt. Sääntöjen mukaan:

-XML dokumentissa pitää olla yksi juurielementti.

-Kaikki elementit pitää sulkea.

-Elementtien nimien tulee olla kirjainkooltaan samat.

-Elementit sulku ei saa olla ennen sen isäntäelementin sulkua.

-Muuttujien arvot tulee olla lainausmerkeissä.

Nämä säännöt läpäissyt XML-dokumentti voidaan lukea XML-parserilla ilman virheitä. Koska XML parserin käyttö on hyvin yleistä XML dokumenttien lukemisessa, on selvää että XML dokumentti kannattaa tehdä well formed XML:ksi. Tällaisen hyväksyttämisen voi tehdä erityisellä validaatioparserilla, joka löytyy useimmista XML muokkaamiseen tarkoitetuista ohjelmista.

“Hyvin muodostettu” XML ei kuitenkaan aina riitä, joskus on tarpeen määritellä XML tiedoston vaatimukset tarkemmin. “Vahvistettu” XML tulee tässä kohtaa vaihtoehdoksi. Tällaisessa XML dokumentissa on alussa talletettuna erityiseen solmuun tieto erityisestä vahvistamistiedostosta. Vahvistamistiedostossa on määriteltynä asiat joiden mukaan XML dokumentti tarkastetaan. Vahvistustiedostoja on useita tyyppejä, yleisimmät kaksi ovat DTD(Document Type Definition) sekä XML Schema.

Prosessia, jossa tutkitaan XML dokumentin vastaavuus määrittelevään tiedostoon, kutsutaan vahvistamiseksi. Tässä prosessissa XML parseri käy läpi XML dokumentin joka on todettu hyvin muodostetuksi. Parseri käy läpi vahvistustiedostossa määritellyt asiat ja tutkii XML dokumenttia niiden pohjalta. Kaikki parserit eivät tue kaikkia vahvistustiedostojen muotoja /3/ /4/, yleisimmin tuetut muodot ovat XML DTD ja XML Schema.

2.3.1 XML DTD

Kuten useat muutkin XML tekniikat, myös XML DTD:n on määritellyt W3C, jonka ohjesivustosta /1/ on kerätty tähän materiaalia.

XML DTD(Document Type Definition) tiedosto määrittelee tietyt rajat eri XML tiedoston sisällöille ja sen mukaan, joko hyväksyy tai hylkää tiedoston. XML DTD ei kuitenkaan ole itse ole XML tiedosto, vaikka se voisi hieman siltä näyttää. Se sisältää tiettyjä ennalta määrättyjä nimiä, joiden mukaan sen tekemä XML tiedoston hyväksymistarkastelu suoritetaan. Näiden lisäksi DTD tiedostossa voidaan määritellä erityisiä “olioita”, jotka korvataan lopulliseen parsittavaan XML tiedostoon.

Hyväksymisperusteiksi voidaan määrittää tietynnimisten elementtien sisältö, muuttujien arvoja ja esiintymistä sekä monenlaista muuta.

Elementin määrittely tapahtuu formaatin <!*ELEMENT (element-name) (category)* | *((element-content))*> mukaan. *element-name* kohtaan tulee käsiteltävän elementin nimi. Elementti voi sijaita missä tahansa kohtaa XML dokumenttia. Seuraavista kentistä *category* ja *element-content* tulee vain toinen. *category* kenttä voi sisältää käskyt *ANY* tai *EMPTY*. Käsky *ANY* hyväksyy minkä tahansa rakenteen tämän elementin alle, kun taasen *EMPTY* hyväksyy vain tyhjän elementin. *element-content* arvo tulee laittaa tavallisten sulkumerkkien sisälle. Tässä kentässä voidaan määrittää tarkemmin elementin sisällön rakenne. Kuvassa 5 rivillä 2 kerrotaan, että elementti jonka nimi on *Valmis* sisältää lapsielementit *Päivä*, *Kuukausi* ja *Vuosi*. Näiden lapsielementtien määrittely on esiteltynä seuraavilla riveillä. Lapsielementeille on määritelty että ne sisältävät parsittavaa tietoa, jolloin niiden

sisällöstä tutkitaan erityisiä olioita jotka on määritelty DTD tiedostossa tai jotka ovat yleisiä. Voidaan myös määrittää että näitä olioita ei parsita elementin sisältämästä tiedosta, tämä tehdään *element-content* parametrillä *CDATA*.

```
1  <!ELEMENT Vuosi (#PCDATA)>
2  <!ELEMENT Valmis ((Päivä, Kuukausi, Vuosi))>
3  <!ELEMENT Tehtävä ((Nimi, Aihe, Aikataulu, Lisätiedot))>
4  <!ELEMENT Päivä (#PCDATA)>
5  <!ELEMENT Nimi (#PCDATA)>
6  <!ELEMENT Kuukausi (#PCDATA)>
7  <!ELEMENT Aloitus ((Päivä, Kuukausi, Vuosi))>
8  <!ELEMENT Aikataulu ((Aloitus, Valmis))>
9  <!ATTLIST Aikataulu
10     Tila CDATA #FIXED "Kesken"
11 >
12 <!ELEMENT Lisätiedot (((Tekijät | Tekijä), Teettäjä?))>
13 <!ELEMENT Henkilötiedot (#PCDATA)>
14 <!ELEMENT Tekijä ((Henkilötiedot))>
15 <!ELEMENT Tekijät ((Tekijä+))>
16 <!ELEMENT Teettäjä ((Henkilötiedot))>
17 <!ELEMENT Aihe (#PCDATA)>
```

Kuva 5 DTD tiedosto.

Edellisessä lapsielementtien määrittelyssä Kuvassa 5 rivillä 2, määritellään jokaista lapsielementtiä olevan aina vain ja ainoastaan yksi kappale. Lapsielementtien määrän määrittäminen voidaan myös tehdä erilaiseksi. Voidaan määrittää että lapsia on yhdestä eteenpäin, asettamalla nimen perään + merkki. Lapsien määrä nollasta eteenpäin määritellään * merkillä. Mikäli halutaan että lapsia on yksi tai ei ollenkaan, lisätään perään *?* merkki.

Taulukko 1 attribute-value:n kelvot komennot

Tyyppi	**Seloste**
CDATA	Muuttuja saa sisältää mitä tahansa tekstidataa
(arvo1|arvo2|..)	Muuttujan arvon pitää olla jokin määritellyistä arvoista
ID	Muuttujan arvon pitää olla uniikki ID:llä merkityissä muuttujien
IDREF	Muuttujan arvon pitää olla jokin ID:ksi määritellyn muuttujan
IDREFS	Muuttujan arvo on lista IDREF arvoista, ks ed
NMTOKEN	Muuttujan arvo on validi XML nimi
NMTOKENS	Muuttujan arvo on lista NMTOKEN arvoista, ks ed
ENTITY	Muuttujan arvo on erityinen olioarvo
ENTITIES	Muuttujan arvo on lista ENTITY arvoista
NOTATION	Muuttujan arvo on XML notaatio

Määrän lisäksi voidaan määritellä lapsien keskinäistä suhdetta paremmin. Voidaan määrittää että joko ”Tekijät” tai ”Tekijä” ilmenee lapsielementtien joukossa erottamalla ne | merkillä, kuvassa 5 Rivillä 12 havainnollistettuna kyseinen komento. Lisäksi kaikkia edeltäneitä voidaan yhdistellä ja muodostaa kuhunkin tarpeeseen sopiva lapsijoukko, kuvassa 5 rivillä 12 esiteltynä tällainen monipuolinen joukko.

Taulukko 2 default-value:n kelvot arvot.

Arvo	Seloste
”*arvo1*”	Muuttujan arvon oletusarvon asetus tekstiksi: *arvo1*
#REQUIRED	Muuttujan pitää olla olemassa
#IMPLIED	Muuttujan ei ole pakko olla olemassa, ei oletusarvoa
#FIXED ”*arvo1*”	Muuttujan arvon pitää olla teksti: *arvo1*

Muuttujien määrittelyssä voidaan määrittää arvon tyyppiä ja olemassaoloa koskevia asioita. Muuttujan arvo määritellään rakenteella <*!ATTLIST element-name attribute-name attribute-type default-value*>. Tässä rakenteessa *element-name* määrittää sen elementin nimen, jonka alta etsitään muuttujaa, jonka nimi on määritelty *attribute-name* kentässä. Kentässä *attribute-type* määritetään muuttujan arvon tyyppiä koskevat rajoitukset, tähän kenttään käyvät arvot esiteltynä taulukossa 1. Kentässä *default-value* määritellään oletusarvon lisäksi myös arvon pakollisuus. Arvo voidaan määrittää myös joksikin tietyksi jolloin muut arvot aiheuttavat virheen. Kelvot komennot *default-value* kenttään ovat määriteltynä taulukossa 2.

XML DTD tiedostossa voidaan myös määritellä erityisiä olioita. Näihin olioihin on viitattu aikaisemmin ja niistä on todettu, että niiden arvo korvataan jollain toisella arvolla. Mitään muuta olioihin liittyvää ei ole. Oliot määritellään rakenteella <*!ENTITY entity-name ”entity-value”*>. Tässä rakenteessa *entity-name* määrittää olion nimen, *entity-value* arvot jolla olion ilmenemät XML tiedostossa korvataan. Olion merkitsemiseen XML tiedostossa käytetään syntaksia *&entity-name;*, jossa *entity-name* on edellisessä määrittelyrakenteessa määritelty samanniminen kenttä.

2.3.2 XML Schema

XML Schema on, kuten XML DTD, vahvistamismenetelmä XML dokumenteille. Ja kuten XML DTD, senkin on määritellyt W3C, jonka ohjesivustolta /1/ on kerätty tähän oleellisimpia osia. XML Schema on XML dokumentti itsessään, toisin kuin XML DTD. XML Schema on monipuolisempi ja laajennettavampi kuin DTD ja se tukee XML nimiavaruuksia. Kuvassa 6 esiteltynä pieni osa XML Schemaa, kohdassa määritellään samanlainen rakenne kuin edeltäneessä XML DTD määrittelyssä, kuvassa 5 riveillä 12 - 15.

XML formaatissa oleva Schema on helposti laajennettavissa ja sitä voidaan muokata helposti millä tahansa XML editorilla. Näin voidaan helposti käyttää aikaisemmin tehtyjä Sceman osia uudelleen. Lisäksi voidaan käyttää useaa määrittelytiedostoa kerralla, jolloin jokaisen määrittelemät asiat käydään läpi. XML formaatissa oleva XML Schema voidaan myös vahvistaa oikeaksi toisella XML Schemalla tai sille voidaan tehdä XSLT muutos, josta kerrotaan myöhemmin tässä dokumentissa. XML Schema kuitenkin poikkeaa perinteisestä XML dokumentista, koska se sisältää paljon ennalta määrättyjä elementtien ja muuttujien nimiä ja arvoja, joilla sen toiminta määritetään, ja tästä seuraa myös se, että Schema itsessään tekee jotain, toisin kuin pelkkä XML tiedosto, näistä ennalta määrätyistä muutamia esiteltynä taulukoissa 3 ja 4.

```
63      <xs:element name="Lisätiedot">
64          <xs:complexType>
65              <xs:sequence>
66                  <xs:sequence>
67                      <xs:choice>
68                          <xs:element ref="Tekijät"/>
69                          <xs:element ref="Tekijä"/>
70                      </xs:choice>
71                      <xs:element ref="Teettäjä" minOccurs="0"/>
72                  </xs:sequence>
73              </xs:sequence>
74          </xs:complexType>
75      </xs:element>
76      <xs:element name="Henkilötiedot">
77          <xs:complexType mixed="true"/>
78      </xs:element>
79      <xs:element name="Tekijä">
80          <xs:complexType>
81              <xs:sequence>
82                  <xs:choice>
83                      <xs:element ref="Henkilötiedot"/>
84                  </xs:choice>
85              </xs:sequence>
86          </xs:complexType>
87      </xs:element>
```

Kuva 6 XML Schema

Verrattuna XML DTD:hen, XML Schema tarjoaa huomattavasti laajemmat mahdollisuudet XML:n talletettujen arvojen tarkasteluun. Arvoille määritellään arvotyyppi johon ne kuuluvat, XML DTD tukee vastaavaa menettelyä, mutta XML Schemassa se on huomattavasti monipuolisempi. Yleisimpiä XML Schema:ssa käytetyistä arvotyypeistä ovat *xs:string*, *xs:decimal* ja *xs:date*. Näiden lisäksi on olemassa paljon muita arvotyyppejä, joita ei tässä kannata listata, voidaan kuitenkin mainita että XML Schema tukee myös omien arvotyyppien määrittelyä, jos olemassa olevat eivät johonkin erityistapaukseen sopisi. Tässä mainitsemattomat arvotyypit löytyvät lähteestä 1.

Varsinaiset vahvistamisperusteet luodaan Schemaan tekemällä tietyllä nimillä elementtejä nimiavaruuteen *xs*. Kaikkien eri elementtien nimien listaaminen ei tässä ole tarpeen, muutamiin yleisimpiin kuuluvat *element*, *complexType*, *sequence*, *simpleType* ja *attribute*. Loput elementtien nimet löytyvät lähteestä 1. Lisäksi juurielementin nimi kuuluu olla *schema*, nimien merkitykset esiteltynä taulukossa 3.

Taulukko 3 Scheman solmuja

Nimi	**Seloste**
element	Määrittää elementin.
complexType	Määrittää isännälleen mahdollisuuden sisältää lapsielementtejä ja/tai muuttujia
sequence	Määrittää että lapsielementtien tulee ilmentyä sarjassa.
simpleType	Määrittää isäntäelementin yksinkertaiseksi elementiksi ja mahdollistaa arvon rajaamisen.
attribute	Määrittää että isäntäelementillä on attribuutti joka on määriteltynä tässä

Näille eri tavoille määritellä varsinaisen XML dokumentin sisältöä, on myös tarkempia määrittelyjä. Jokaiselle elementin nimelle, joista on osa esitettynä taulukossa 3, on oma ryhmä muuttujia joilla voidaan määrittää siihen liittyviä asioita. Yleisimpiä tällaisista on *maxOccurs* ja *minOccurs*. Näillä voidaan määrittää rajat sille, kuinka usein elementti saa ilmentyä vahvistettavassa XML dokumentissa. Edelliset määrittelevät muuttujat ja muutama muu yleisin esiteltynä taulukossa 4.

Taulukko 4 XML Scheman muuttujia.

Nimi	**Seloste**
name	Määrittään minkä niminen asia on kyseessä, esim elementin nimi.
minOccurs	Määrittää elementtien määrän joka vähintään pitää olla, oletusarvo 1
maxOccurs	Määrittää maksimimäärän, oletusarvona 1, rajattomaksi arvolla *unbouded*
use	Määrittää muuttujalle käytön pakollisuutta
type	Määrittää muuttujalle arvon tyyppiä koskevia rajoituksia

2.4 XPath

XML tiedostosta jonkin tietyn elementin hakemiseen käytetään aivan omanlaista syntaksia, jota kutsutaan XPathiksi. XPath kyselyt pohjautuvat XML DOM rakenteeseen, jolloin XML tieto pitää siis ladata ensin muistiin tällaiseksi rakenteeksi. Syntaksi muistuttaa hyvin paljon hakemistorakenteen vastaavaa syntaksia, mutta se sisältää myös huomattavan määrän omia erityiskomentojaan,

joiden hallitseminen vaatii tutustumista. Muutamia yleisimpiä on esiteltynä taulukossa 5.

XPath määrittelyn on tehnyt W3C, jonka ohjesivustolta /1/ on tähän kerätty muutamia oleellisimpia asioita.

Taulukko 5. XPath komentoja

Komento	**Selite**
elementin_nimi	Valitaan kaikki lapsielementit joiden nimi vastaa esitettyä
/	Ensimmäisenä merkkinä ilmoittaa valinnan alkavan juuresta, muuten valitaan lapsielementtejä edellisestä määritetystä elementistä
//	Valitaan nykyisestä paikasta alaspäin, minne tahansa asti.
.	Valitaan nykyinen elementti.
..	Valitaan tämän elementin isäntäelementti.
@muuttujan_nimi	Valitaan muuttuja jonka nimi on *muuttujan_nimi*

Taulukossa 5 mainitut tavat eivät juuri eroa tavallisesta tiedostopolkujen määrittelystä, ainoa erikoisuus on muuttujan hakeminen. Tavallisen tiedostojärjestelmän polkurakenteesta erotaan kun määritellään kaksi samannimistä elementtiä yhden isäntäelementin alle. Tähän tarvitaan erityisiä lisämääreitä, joilla voidaan tarkemmin määrittää haluttu elementti. Yksinkertaisimmillaan näillä voidaan määritellä halutun elementin järjestysnumero, mutta monimutkaisemmalla määrittelyllä voidaan hakea myös elementtejä joiden alla olevassa muuttujassa on jollain tavalla määritelty arvo. Tällainen määrittely seuraa aina jonkin elementin nimeä, näitä lisämääreitä esiteltynä taulukossa 6.

Taulukko 6 XPath komentoja.

Komento	**Seloste**
[X]	Valitaan se elementti jonka järjestysnumero vastaa tässä annettua. Alkaa ykkösestä.
[last()]	Valitaan viimeinen elementti.
[last()-1]	Valitaan viimeistä edeltänyt elementti.
[position() < 3]	Valitaan kaksi ensimmäistä elementtiä.
[@muuttuja]	Valitaan elementti jonka alla on muuttuja nimeltä muuttuja.
[@muuttuja=’arvo’]	Määritellään edellisen lisäksi arvo joka pitää muuttujalla olla.
[@muuttuja<42]	Määritellään että muuttujan arvon tulee olla alle 42.

XPath kyselyitä voidaan myös yhdistää |-merkillä jolloin saadaan valittua samaan kyselyyn useita eri XPatheja. XPath kyselyn tulos on joukko DOM solmuja, joiden takaa tietoa voidaan kysellä.

Jos kuvitellaan kuvassa 4 esitettyyn tiedostoon juurielementiksi *Root* ja sen alle useampia tehtäviä, joista haluttaisiin etsiä niiden tehtävien nimet, jotka ovat kesken. Voitaisiin käyttää seuraavaa XPath kyselyä tulosten hakemiseen.

/Root/Tehtävä/Nimi[Aikataulu[@Tila='Kesken']]

2.5 XSLT

Mikäli on tarve muuttaa XML tiedosto johonkin muuhun muotoon, käytetään XSLT muunnosta, jolla on mahdollista muuttaa XML tiedosto lähes mihin tahansa muuhun muotoon. Tyypillisimmillään muutetaan XML muotoinen tieto luettavaan muotoon, esimerkiksi HTML sivuksi, jolloin se voidaan näyttää selaimessa. XSLT muutos onkin tuettuna useimmissa selainohjelmissa.

XSLT on peräisin W3C:n määrittelyistä, jonka ohjesivustolta /1/, on kerätty tähän oleellisimpia osia.

XSLT on varsinainen muutosprosessi, jossa käytetään XSL tiedostoa ja XML tiedostoa lopputuloksen tekemiseen. XSLT käyttää XPath kyselyitä tarvittavan tiedon paikantamiseen XML tiedostosta.

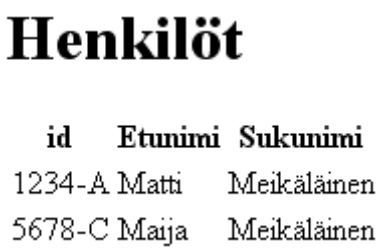
```
11    <xsl:template match="/">
12       <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fi" lang="fi">
13          <head>
14             <meta http-equiv="Content-Type"
15             content="text/html; charset=ISO-8951-1" />
16             <title>Meikäläiset</title>
17          </head>
18          <body>
19             <xsl:apply-templates/>
20          </body>
21       </html>
22    </xsl:template>
23
24    <!-- taulukko -->
25    <xsl:template match="henkilot">
26       <h1>Henkilöt</h1>
27       <table>
28          <tr>
29             <th>id</th>
30             <th>Etunimi</th>
31             <th>Sukunimi</th>
32          </tr>
33          <xsl:apply-templates/>
34       </table>
35    </xsl:template>
36
37    <!-- henkilörivit -->
38    <xsl:template match="henkilo">
39       <tr>
40          <td><xsl:value-of select="@id"/></td>
41          <td><xsl:value-of select="etunimi"/></td>
42          <td><xsl:value-of select="sukunimi"/></td>
43       </tr>
44    </xsl:template>
```

Kuva 7 XSL tiedosto

```
3  <henkilot>
4     <henkilo id="1234-A">
5        <etunimi>Matti</etunimi>
6        <sukunimi>Meikäläinen</sukunimi>
7     </henkilo>
8     <henkilo id="5678-C">
9        <etunimi>Maija</etunimi>
10       <sukunimi>Meikäläinen</sukunimi>
11    </henkilo>
12 </henkilot>
```

Kuva 8 XML tiedosto

XSL itsessään täyttää XML tiedostolta vaaditun formaatin. Siinä tietyillä elementeillä ja muuttujilla ohjelmoidaan skripti, joka läpikäytynä tuottaa halutun tuloksen. Tulos ei välttämättä ole XML tiedoston mukainen, mutta se voi toki olla. XSL tiedoston elementtien tulee olla nimiavaruudessa xsl.



Kuva 9 HTML sivu

Tiedoston juurielementiksi tulee asettaa stylesheet niminen elementti, jossa määritetään käytetty XSL tiedoston versionumero. Seuraavana määritellään template-elementti, jossa myös kerrotaan XPathilla kohta XML tiedostossa, johon tämä template osoittaa.

Templateja voi XSL tiedostossa olla useita ja niistä voidaan kutsua toisia templateja, jolloin on mahdollista osioida templateja ja käyttää niitä uudelleen eri kohdissa. Tämä mahdollistaa myös rekursiivisen XSLT muunnoksen.

Kun ajetaan kuvassa 7 esitetty XSL muunnos kuvassa 8 esitetylle XML tiedostolle, saadaan tuloksena HTML sivu, joka näyttää selaimella kuvan 9 näköiseltä.

3 XML käsittely

Tämän kappaleen alla esitellään XML tiedoston käsittelyyn liittyviä asioita ja tutkitaan niiden soveltuvuutta. Lisäksi esitellään XML rajanpintaluokka sekä XML rakenteeseen perustuvien luokkarakenteiden käyttöä. Lopussa vertaillaan kahta XML käsittelyyn tarkoitettua kirjastoa.

3.1 Yleistä XML käsittelystä

XML tiedostoista on tullut tätä nykyä eräänlainen muoti-ilmiö. Moniin ohjelmiin halutaan tuki XML-,tai siitä periytetyn, muotoisen tiedon käsittelyyn.

”Many software vendors are moving to XML for their own data simply because it’s a well-understood, general purpose format for structured data that can be manipulated with easily available, cheap, and free tools. /4 s.46/”

XML tiedoston käyttämiseen kannattaa usein liittää vähintään tyylitiedosto, useimmat XML kirjastot tukevat XML Schema muotoon tehtyä tyylitiedostoa. Usein tällaisen tiedoston luominen sujuu kuin itsestään, kunhan XML tiedostosta on tehty jonkinlainen esimerkki. On olemassa ohjelmia, joilla tällainen tiedosto voidaan tehdä XML tiedoston pohjalta, tällainen toki saattaa jättää joitain asioita turhan avoimeksi tai rajoittaa jotain. Tämän vuoksi esimerkkitiedoston kannattaa esitellä mahdollisimman paljon erilaisia muotoja, joita oikeasti tiedostossa sallitaan. Käsin XML Scheman jatkaminen on usein tarpeen, joten kannattaa perehtyä siihen etukäteen.

Testattaessa automaattista XML Scheman luontia ja sen dokumentaatiota tutkittaessa ilmeni eräs puutteellisesti määritelty asia, jota XML tiedosto tukee mutta XML Schema ei osaa toteuttaa. XML Schemassa ei voida luoda rakennetta, jossa elementillä olisi sekä sisäinen teksti että lapsielementtejä. Altova XMLSpy ohjelmistolla XML Scheman tekeminen tällaisesta rakenteesta antaa tuloksena elementin, jolla on lapsielementtejä, mutta ei sisäistä tekstiä. Tällaisesta rakenteesta ei ole mainintaa lähteessä 1, joka on XML määrittelyt tehnyt taho. Kohtuullisesti suoritettu lisätutkinta ei tuottanut tulosta, mikä johtunee asian teknisyydestä.

Näin voidaankin päätellä että tällaista rakennetta kannattaa välttää, vaikka useissa XML oppaissa kannustetaan tallentamaan varsinainen tieto tällaisiin teksteihin muuttujien sijaan. Ongelma ilmenee kun halutaan määritellä XML tiedostossa tarkemmin jotain asiaa, mikä johtaisi siihen että elementin alle pitäisi luoda lisää elementtejä. Ongelmaa tietysti ei ole, jos ei ole tarkoitus käyttää XML Schemaa, mutta yleensä sen käyttöön kannattaa varautua.

3.2 XML tiedoston rakenteen suunnittelu

Varsinaisen rakenteen suunnittelusta, on useissa eri lähteissä /1/ /4/ /5/ /6/ mainittu, että muuttujia kannattaa välttää tiedoston myöhemmän muokattavuuden helpottamiseksi. Elementtien käyttöä tiedon tallentamiseen perustellaan sillä, että se on helpompaa lukea, lisäksi mainitaan, että elementtien järjestykseen voidaan tallentaa tietoa, kun muuttujilla tämä tieto häviää. Muuttujia kehotetaan käyttämään tietoa kuvaavana tietona, esim. tunnistenumeroina. Toisaalta, lähteessä 4 mainitaan sivulla 109, että käytettäessä tietoa kuvaavaan tietoon elementtejä, mahdollistetaan tiedon kuvausta kuvaavan tiedon lisääminen.

Elementtien käyttö tiedon varastoinnissa, perusteltuna sillä että XML tiedoston laajentaminen helpottuu, ei ole täysin aukotonta. Jos tiedostoa laajennetaan jostain kohtaa, joudutaan joka tapauksessa muokkaamaan tiedostoa käsittelevää ohjelmaa, kuten on selitetty lähteessä 4 sivulla 114. Mikäli tietoa jouduttaisiin lisäämään ratkaisuun, jossa käytettiin aikaisemmin muuttujia, ei ole mitenkään mahdotonta vaihtaa niiden tilalle elementtejä. Muutoksien tekeminen ei eroa tilanteesta, kun laajennetaan elementeillä tehtyä ratkaisua. Laajennus jonkin arvon tarkemmaksi määrittelyksi, johtaa aina yhteensopivuusongelmiin ohjelman versioiden välillä ja ohjelman toiminnan muuttamiseen. Yhteensopivuusongelmia voidaan kiertää tekemällä XSLT muutoksia XML tiedolle eri versioita varten.

Muuttujien käyttö on hyvin perusteltua, kun tehdään tiedostoa jonkin määrittelyn mukaan ja voidaan olla varmoja tietojen pysyvyydestä. Lisäksi, muuttujien käyttö toimii paremmin XML Scheman kanssa, kappaleessa 3.1 esitellyn ongelman vuoksi. Mikäli tiedostoa laajennetaan ja halutaan laajentaa jotain kohtaa missä aikaisemmin kaikki tieto oli tallennettuna yhteen elementtiin, joudutaan poistamaan ko. elementin sisältö. Tämä johtaa siihen, että XML tekniikan eduksi kerrottu ominaisuus, alaspäin yhteensopivuus laajennettaessa /1/, ei täyty. Ratkaisu, jossa arvot on tallennettuna Elementin sisältämiin muuttujiin, jättävät laajennuksen avoimeksi siten että alaspäin yhteensopivuus säilytetään.

Tiedoston rakenne vaikuttaa hyvin oleellisesti moniin eri osiin tiedon käsittelystä, jolloin se kannattaa suunnitella hyvin.

Tietoa kuvaava tieto kannattaa ilmoittaa mahdollisimman aikaisessa vaiheessa. Näin saadaan tieto helpommin lokeroitua ja yksinkertaisempaan muotoon. Esimerkiksi suuressa osoiteluettelossa kannattaisi aikaisella tasolla määritellä maa johon osoite on, tämän jälkeen kaupunki, postiosoite jne. Tällaisessa rakenteessa yksittäisen osoitteen kaikki tiedot on talletettuna hajautetusti. Tiedon hakemisvaiheessa joudutaankin keräämään tiedot eri paikoista jotta saadaan kaikki tiedot esiteltyä. Tämä vaihtoehto sopii tiedolle josta haetaan yksittäisiä tietoja tai johonkin ryhmään sopivia tietoja.

Tiedoston rakenteen suunnittelussa kannattaakin lähteä aluksi tutustumaan varsinaiseen tallennettavaan tietoon. Mikäli kyseessä on jokin monimutkaisempi asia, onkin kannattavaa tehdä yhteistyötä asian tuntevan henkilön kanssa. Paras mahdollinen vaihtoehto toki olisi että henkilö osaa muodostaa sekä XML tiedostojen rakenteita että hän tuntee kuvattavan asian.

Lähteessä 5 sivulla 27 mainitaan asiasta osuvasti, ”When you create an information model for your data, you identify all the pieces that compose the structure of your documents, and any hierarchical relationships between the pieces of content.”

3.3 XML rajapintaluokka

Tiedon kysely XML rakenteesta suoritetaan yleensä aina tietyillä samankaltaisilla kyselyillä, yleisin tapa on XPath kysely. Tällaisten samankaltaisten tiedonhaku tai talletus on kuitenkin XML kirjastoissa /3/ /4/ usein hyvin erilaista. Jokainen XML kirjasto vaatii oman tutustumisensa, jotta sillä kyselyiden tekeminen onnistuu, vaikka kyseessä onkin aina usein samankaltainen asia.

Onkin kannattavaa luoda rajapinta kirjaston ja muun ohjelmiston välille. Tällainen rajapinta kannattaa tehdä yleiseksi eri projektien välillä, jotta aina käytettävät funktiot näyttäisivät ja toimisivat samankaltaisesti. Näin riittää kun yksi henkilö joutuu selvittämään XML kirjaston sisäisen toiminnan, jolloin myös tilanteesta, missä kirjastoa joudutaan vaihtamaan, selvitään kohtuullisen helpolla.

Tarkoitus on tehdä rajanpinta, jonka kautta saadaan kaikki XML käsittelyyn liittyvät perusasiat hoidettua helpon ja ymmärrettävän muodon mukaan. Lisäksi

rajapinnasta saadaan helposti muutettava toimimaan eri rajapinnan ulkopuolella olevien toteutusten kanssa.

Tällaisella rajapinnalla voidaan myös helpottaa toteuttajien työtä, koska muistettavien asioiden määrä vähentyy käytettäessä yksinkertaista rajapintaa.

3.3.1 Yleinen määrittely

Yleisesti määritellään että kaikkien funktioiden paluuarvona pitää olla totuusarvo, joka kertoo operaation onnistumisesta. Epäonnistuneen operaation jälkeen pitää mahdollisimman hyvin palata tilanteeseen joka oli ennen funktion kutsumista.

Suunnittelu jaetaan kahteen osaan, rajapinnan toteutusta helpottaviin sisäisiin funktioihin ja ulkopäin käytettäviin funktioihin jotka rajapinnan käyttäjä näkee. Sisäiset funktiot eivät tee tiedon tarkasteluja tai muutoksia vaan toimivat mahdollisimman nopeasti. Sisäiset funktiot toteutetaan ennen rajapinnasta ulospäin näkyviä funktioita, jotka toteutetaan käyttämällä mahdollisimman paljon hyödyksi sisäisiä funktioita.

XML käsittelyyn liittyvän teksti määritellään kahdella tasolla, se mitä ulkoinen toteutus käyttää ja se mitä XML kirjasto käyttää. Tähän välille luodaan muutosfunktiot joilla voidaan muuttaa tekstimuuttujan tyyppi näiden kahden eri tyypin välillä.

Rajapinnasta ulos annettavat solmut toimivat samankaltaisesti kuin teksti, rajapinnan ulkopuolella käytössä on vain 64 bittinen muuttuja, joka toimii tunnisteena solmuun. XML kirjaston käyttämän solmutyypin ja 64 bittisen muuttujan välille luodaan eri suuntiin toimivat muutosfunktiot.

Funktioiden kutsumisjärjestykselle ei aseteta mitään rajoituksia, mikään kutsumisjärjestys ei saa aiheuttaa määrittelemätöntä toimintaa tai ohjelman kaatumista.

Funktioiden nimeäminen tehdään siten, että tuntemalla muutaman funktion nimet ja toiminta, voidaan helposti päätellä pelkän nimen perusteella jonkin samankaltaisen funktion toiminta. Esim. *GetValueFromPath*- ja

GetNodeFromRelativePath-funktiot tuntemalla voidaan helposti päätellä funktion *GetValueFromRelativePath* toiminta. Lisäksi annettujen parametrien järjestys tulee olla samankaltainen.

3.3.2 Sisäisesti käytettävät funktiot

Tässä osiossa esitellään funktiot joilla muutetaan eri arvot, tekstit ja solmutunnisteet rajapinnan ulkona ja sisäpuolella näkyvien tyyppien välillä. Lisäksi tässä osiossa esitellään muutamia XML rakennetta käsitteleviä funktiota, joita on tarkoitus käyttää rajapinnan käyttäjälle näkyvien funktioiden toteuttamisessa.

static const NodeIDType NodeToNodeID(const NodeType & _dnNode);

static const NodeIDType NodeToNodeID(const NodeType * _dnNode);

Luodaan XML kirjaston käyttämästä solmutyypistä ulkopuolella käytettävä tunniste, jolla voidaan tunnistaa yksittäinen solmu.

static NodeType * NodeIDToNode(const NodeIDType & _lNodeID);

Palautetaan solmutunniste solmuksi jota XML kirjasto käyttää.

static IntStringType MakeIntStringType(const ExtStringType& _sText);

Funktio jonka kautta muutetaan ulkopuolella käytettävä tekstijono-tyyppinen muuttuja XML kirjaston käyttämään tekstijono-tyyppiin.

static ExtStringType MakeExtStringType(const IntStringType& _sText);

static ExtStringType MakeExtStringType(IntStringType& _sText);

Muutetaan XML kirjaston käyttämä tekstijono-tyyppi sellaiseksi mitä ulkopuolinen toteutus käyttää.

bool GetNodeFromPath(const IntStringType & _sPath, NodeType & _lNodeID);

Funktio jolla voidaan tehdä XPath kysely aloittaen juuresta, mikäli kyselyn tuloksena saadaan vain yksi node, se palautetaan viimeisenä annetussa viittauksessa.

bool GetValueFromNode(const NodeType& lNodeID, IntStringType& _sValue);

Haetaan annetulta solmulta arvo, tavalla joka riippuu annetun solmun tyypistä. Esimerkiksi, jos kyseessä on tavallinen solmu, haetaan sen sisältämä teksti, mutta jos kyseessä on muuttuja, haetaan sen arvo.

3.3.3 Ulkoapäin käytettävät funktiot

Tässä osiossa esitellään muutamia rajapinnan käyttäjälle näkyviä funktioita. Esittelemättä jätetään muutamia toiminnaltaan samankaltaisia funktioita, joiden toiminta voidaan päätellä edellisessä kappaleessa mainitulla tavalla.

bool OpenXMLFileFromPath(const ExtStringType & _sFilePath);

Avaa xml tiedoston mikäli se on ”hyvin muodostettu” XML tiedosto. Mikäli tiedoston lataaminen epäonnistuu, tulee palauttaa false ja kaikki mahdollisesti varattu muisti tulee vapauttaa. Jolloin periaatteessa palataan tilanteeseen ennen funktiokutsua. Tämän funktion kutsumisen jälkeen muiden XML rakennetta käsittelevien funktioiden tulee toimia.

bool GetXMLContentAsText(ExtStringType & _sContentText);

Kirjoitetaan koko XML tiedoston sisältö tekstinä annetun tekstiviittauksen taakse. Voidaan käyttää esimerkiksi epäonnistuneen Schema tarkastelun jälkeen näyttämään tiedostosta virheellinen kohta.

bool GetNodeFromPath(const ExtStringType & _sPath, NodeIDType & _lNodeID);

Funktio jolla voidaan tehdä XPath kysely aloittaen juuresta, mikäli kyselyn tuloksena saadaan vain yksi node, se palautetaan viimeisenä annetussa viittauksessa.

bool GetPathFromNode(const NodeIDType & _lNode, ExtStringType & _sPath);

Hakee polun kyseiseen elementtiin, käänteinen operaatio XPath kyselyyn verrattuna. Palautetaan kuitenkin vain järjestysnumeroilla määritelty XPath, jolloin eri järjestykseen ladattu XML tiedosto tai dataan lisätty uusi node aiheuttaa sen että XPath ei enää välttämättä osoita samaan paikkaan.

bool GetNodeTagName(const NodeIDType & _lNode, ExtStringType & _sName);

Palauttaa kyseisen XML noden nimen.

bool GetNodesFromPath(const ExtStringType & sPath, NodeStack & Nodes);

Haetaan useita solmuja, voidaan palauttaa mikä tahansa nollasta eroava määrä solmuja. Solmujen hakemiseen käytetään XPath kyselyä.

bool GetNodeFromRelativePath(const ExtStringType & _sPath, const NodeIDType & _lNodeID, NodeIDType & lResultNodeID);

Haetaan solmu XPath kyselyllä joka alkaa annetusta solmusta.

bool GetValueFromPath(const ExtStringType & _sPath, ExtStringType& _sValue);

Haetaan arvo XPath kyselyn osoittamasta paikasta. XPath kysely aloitetaan aina juuresta.

bool GetChildNodeFromNode(const ExtStringType & _sChildname, const NodeIDType & _lNodeID, NodeIDType & lResultNodeID);

Haetaan annetulta solmulta lapsi jolle on annettu joku tietty nimi. Ei käytetä XPath hakua vaan haetaan tieto mahdollisimman nopeasti XML DOM rakenteesta.

bool GetValueFromNode(const ExtStringType & _sPropertyName, const NodeIDType & _lNodeID, ExtStringType & sValue);

Haetaan annetunniminen muuttuja annetulta solmulta ja palautetaan muuttujan arvo. Haetaan tieto suoraan XML DOM rakenteesta mahdollisimman nopeasti.

bool GetValueFromNode(const NodeIDType& lNodeID, ExtStringType& _sValue);

Haetaan annetulta solmulta arvo, tavalla joka riippuu annetun solmun tyypistä. Esimerkiksi, jos kyseessä on tavallinen solmu, haetaan sen sisältämä teksti, mutta jos kyseessä on muuttuja, haetaan sen arvo.

bool SaveXMLToFile(const ExtStringType & _sFilePath);

Tallennetaan muistissa oleva XML tieto tiedostoon, jonka polku on annettu.

bool AddValueToNode(const NodeIDType & lNodeID, const ExtStringType & _sValueName, const ExtStringType & _sValue = NULL);

Lisätään annetulle solmulle muuttuja, jolle annetaan arvo, mikäli se on annettu.

bool AddNodeToNode(const NodeIDType _lNode, const ExtStringType & _sNodeName, const ExtStringType & _sNodeContent = NULL);

Luodaan uusi elementti annetun elementin alle. Elementin nimeksi tulee annettu nimi, ja sen tekstiksi asetetaan annettu teksti, mikäli se on annettu.

bool ModifyValueAtPath(const ExtStringType & _sPath, const ExtStringType & _sValue);

Haetaan annetun polun takaan XPath kyselyllä muuttuja ja asetetaan sille annettu uusi arvo.

bool CreateNewXMLDoc();

Luodaan uusi tyhjä pohja, johon voidaan lisätä solmuja ja joka voidaan tallentaa.

bool FreeDoc();

Vapautetaan muistiin ladattu XML DOM rakenne

3.3.4 XPath kyselyiden käyttö

XPath haku suoritetaan tarkastelemalla muistiin ladattua XML DOM rakennetta. Tämä rakenne riippuu hyvin paljon XML kirjaston tavasta parsia tietoa ja tallentaa sitä. Muistiin tallentaminen pääasiassa tapahtuu siten, että kaikki XML data luetaan perustyypiltään samanlaiseen rakenteeseen, joka sitten sisältää osoitteita toisiin samanlaisiin rakenteisiin. Näiden osoitteiden taakse tallennetaan tieto isännästä, sisarista sekä lapsista. Lisäksi tällainen rakenne sisältää tiedon omasta tyypistään, nimestään sekä arvostaan. XML kirjastosta riippuen saattaa tällainen rakenne sisältää myös muita tietoja, kuten nimiavaruuden nimi.

XPath haku pohjautuu edellä mainittuun tapaan tallentaa tietoa. Kun haluaan valita jonkin elementin lapsielementti, käydään läpi kaikki lapsielementit ja tutkitaan niiden nimet. XPath haku on juuri tämän takia kovin hidas. XPath haku joutuu tutkimaan huomattavan määrän tekstiä jotta hakutulos saadaan aikaiseksi. XPath haulla suoritettua hakuprosessia voidaan tietyssä mielessä verrata SQL kyselyyn. XPath haku on parhaimmillaan haettaessa suurta massaa tietoa.

Haun hitauteen vaikuttaa myös tietysti eri tasoilla olevien elementtien määrä. Pelkällä XML tiedoston muodon muuttamisella päästään nopeammin toimivaan ratkaisuun, mutta myös tiedostossa olevien elementtien nimien lyhentäminen vaikuttaa parsinnan nopeuteen.

Eri kyselyitä voidaan helposti nopeuttaa jos tiedetään XML rakenne varmasti jonkin tietynmuotoiseksi. */Root/Node/ChildNode[@param[.='value']]* kyselyä voidaan nopeuttaa huomattavasti jos tiedetään että *ChildNode* tasolla on pelkästään *ChildNode* nimisiä elementtejä. Tällöin elementin nimen tarkastelu on turhaa ja yksinkertaisempaa onkin valita kaikki elementit *Node* elementin alta. Kysely saadaan silloin muotoon */Root/Node/*[@param[.='value']]*. Testaustapauksessa vastaava muokkaaminen aiheutti ajankäytön tippumisen 40 %:n edellisestä. Testaustapauksessa käytössä oli libxml2 kirjasto, ja ajettavassa tiedostossa oli 700 tämän tapauksen *ChildNode* elementtiä. Nopeus siis muuttui tässä testitapauksessa noin 2,4 kertaiseksi. Tästä voidaan todeta, että XPath komentojen muokkaamisella voidaan päästä suorituskykyisempään ratkaisuun. Varsinaiset tulokset ovat

kuitenkin verrannollisia käytettyyn tietorakenteeseen ja tietorakenteeseen tallennetun tiedon määrään.

Tällainen kyselyiden muodon muokkaaminen ei kuitenkaan ole kovin turvallista, sillä kun XML tiedostoon yllättäen halutaankin lisätä tiettyyn kohtaan lisää tietoa, voi olla että tarkasteluun tulee tällöin joitain aivan muita elementtejä. Tästä syystä kannattaakin jättää XPath kyselyt helposti muokattavaksi, eräs vaihtoehto olisi niiden lataaminen vaikka samasta XML tiedostosta missä varsinainen tieto on talletettuna.

XPath kyselyn käyttämistä tulisi välttää usein toistettavissa asioissa, useimmiten onkin kannattavaa kysyä paljon tietoa kerralla ja lopuksi käsitellä saatu tieto. Saadun tiedon käsittelyssä pitäisi välttää XPath kyselyiden käyttämistä tiedon hakuun. Jatkokäsittelyssä kannattaakin käyttää suoria XML DOM rakenteeseen perustuvia tiedonhakumenetelmiä.

3.3.5 Tiedon tallentaminen luokkarakenteeseen

Eräs mahdollisuus tiedon nopeampaan käsittelyyn, on omien tietorakenteiden muodostaminen XML tiedon perusteella. Etuna tällaisessa ratkaisussa on se, että tietoja ei tarvitse hakea XML DOM rakenteesta eikä tallentaa sinne. Koska tieto tallennetaan tekstimuodossa XML DOM rakenteeseen, esimerkiksi numeroina käsiteltävien muuttujien toiminnoista saadaan nopeampia. Ratkaisun nopeuttava vaikutus riippuu tietysti hyvin paljon ympärillä olevista ratkaisuista ja tiedon muodoista. Tämän insinöörityön yhteydessä tehdyssä projektissa havaittiin käsittelyn nopeutuminen arviolta kolminkertaiseksi. Nopeus tuntui eniten tallentaessa ja vertailtaessa numerotyyppisiä muuttujia. Tämän tyyppisen ratkaisun rakenne on kuitenkin hyvin erilainen em. XML DOM malliin verrattuna ja sen käyttöä tulisikin harkita tarkemmin ohjelmistoa suunniteltaessa.

Kun on tiedossa tarkka rakenne johon XML puussa tieto on tallennettu, voidaan tehdä helposti nopeasti toimiva tapa ladata tieto sieltä. XML DOM rakennetta käyttäen voidaan helposti päästä käsiksi tietynlaisessa rakenteessa olevaan tallennettuun tietoon.

Tällaisten tietoa hakevien järjestelmien tekeminen on erittäin helppoa, mutta samalla tylsää ja itseään toistavaa. Operaatio on niin yksinkertainen, että sen pohjalta voitaisiin tehdä ohjelma, joka suorittaa ko. operaation.

Näissä ohjelmissa siis luotaisiin, esimerkiksi, C++ luokka, joka sisältäisi muodostajan joka luo luokan olion XML rakenteen perusteella. Toiminto olisi hyvä tehdä myös toiseen suuntaan, jolloin olioon tallennetusta tiedosta voitaisiin muodostaa XML rakenne.

Tällainen ohjelma tarvitsee kuitenkin tuekseen tietynlaiset funktiot joilla XML tietoa voidaan käsitellä, jolloin kannattaakin käyttää tässä dokumentissa esiteltyä XML rajapintaa tiedon käsittelyyn, jotta saadaan aikaan XML kirjastosta riippumattoman toteutuksen automaattinen luominen.

Eräs vaihtoehto tällaiseksi olisi ohjelma, jolla luodaan XML rakenne. Tällaisella ohjelmalla luotaisiin tietorakenne XML muotoon ja rakennetta luotaessa, tehtäisiin myös toiminnot, jolla tiedot voitaisiin hakea XML rakenteesta. Raaka XML datan tutkiminen on siinä mielessä vaikeaa että välttämättä ei tiedetä mitä missäkin kohtaa on haluttu kuvata.

Raa’asta tiedosta luokkien luominen on lähes mahdotonta, tai mikäli näin tehdään, tulee varmasti luotua useita turhia asioita. Tähän kuitenkin vaihtoehtona olisi tehdä interaktiivinen parsinta, jossa tutkitaan XML rakennetta ja kysytään käyttäjältä eri elementtien kohdilta luokan muodostamiseen liittyviä asioita. Esimerkiksi, parsittaessa kuvan 10 mukaista XML rakennetta, kysytään aluksi, haluaako käyttäjä luoda uuden luokkamallin Data elementistä, johon käyttäjä vastaa ei. Seuraavaksi kysytään haluaako käyttäjä luoda uuden luokkamallin Block elementistä, johon käyttäjä vastaa, kyllä. Ohjelma havaitsee että Block elementillä on lapsielementti jonka nimi on Stuff, lisäksi ohjelma löytää lisää Stuff nimisiä elementtejä. Ohjelma kysyy käyttäjältä, halutaanko luokkaan liittää Stuff tieto. Käyttäjä valitsee, kyllä. Tässä vaiheessa ohjelma ei löydä muita erinimisiä asioita Block elementtiin liittyen, jolloin siirrytään Block elementin alle luotua Stuff tietoa. Havaitaan, että Stuff tieto sisältää muuttujia, jolloin ohjelma kysyy käyttäjältä jokaisen muuttujan nimen kohdalla, josko ne halutaan lisätä luokkaan, johon käyttäjä vastaa, kyllä.

Näin luodaan luokka CBlock joka sisältää taulukon jossa on kaksi CStuff luokan olioita. CStuff luokan olio sisältää kaksi muuttujaa Param1 sekä Param2. Lisäksi ohjelmaan voidaan lisätä käsittely jossa kysytään tarkemmin arvojen tyyppejä, esim että tässä tapauksessa Param1 olisi teksti ja Param2:n arvoksi hyväksyttäisiin vain numeroarvo, jolloin Param1:n tyyppi olisi char* ja Param2:n int, tai joku muu määritelty numeromuuttuja. Muita määrittelyitä voisivat olla esimerkiksi lapsielementtien määrän rajaaminen johonkin tietylle alueelle tai määrän rajattomuus.

```
1  <Data>
2      <Block>
3          <Stuff Param1="1" Param2="2" />
4          <Stuff Param1="1" Param2="2" />
5      </Block>
6      <Block>
7          <Stuff Param1="1" Param2="2" />
8          <Stuff Param1="1" Param2="2" />
9      </Block>
10 </Data>
```

Kuva 10. XML tiedosto

3.3.6 XML kirjastojen vertailua

Sain mahdollisuuden tutustua tarkemmin pariin XML tietoa käsitteleviin kirjastoihin tähän työhön liittyvän projektin myötä. Toteutin näille kirjastoille aikaisemmin määrittelemäni rajanpinnan toteuttavat funktiot.

3.3.6.1 Qt QXML

Qt:n kirjastot ovat yleensä helppokäyttöisiä ja siten myös tehokkaita toteuttamisen kannalta, koska tähän työhön liittyvässä projektissa muutenkin käytettiin Qt:n kirjastoja, tutustuin ensin lyhyesti Qt:n XML käsittelyyn.

Kirjaston dokumentaatio /4/, kuten muutkin Qt kirjastojen dokumentaatiot, oli erittäin helppolukuista

Qt XML tarjoaa hyviä, selkeitä luokkia ja funktioita C++:n ja Qt:n käyttöön tottuneelle toteuttajalle. Ainoa heikko puoli kirjastossa oli, ettei se tukenut ollenkaan XPath kyselyitä eikä XSLT muunnoksia.

Varsinaisen XPath kyselyn puuttumisen vuoksi määritin toteutettavaksi muutamia yksinkertaisia XPath kyselyrakenteita ja toteutin ne. Suunnittelin XPath kyselyn toimimaan rekursiivisesti käyttäen muutamia apufunktioita kyselyn suorittamiseen.

Toimivia XPath kyselyitä oli kuitenkin vain hyvin pieni osa siitä mitä varsinainen XPath kysely tukee. Toimiviksi asti tehtiin, lapsielementin valinta nimen perusteella, isäntäelementin valinta ja lapsielementin valinta tutkimalla sen alta arvoa. Jälkimmäiseen arvojen tarkasteluun toteutettiin vain muuttujan yhtäläisyystarkastelu johonkin tietyn arvon kanssa.

Qt:n uusimmassa versiossa, 4.4, on kuitenkin tämä toteutettuna, toteutuksen tekovaiheessa kyseistä versiota ei ollut vielä käytettävissä.

3.3.6.2 libxml2

Kirjaston ehkä parhain ominaisuus on sen tuki monelle eri alustalle. Kirjasto on käännetty usealle eri alustoille kuten Windows, Linux sekä Mac OS X. Mikäli tarvittavaa käännöstä ei ole vielä olemassa, on se helppoa tehdä sillä libxml2 vaatii vain muutaman, hyvin eri ympäristöissä kääntyvän, kirjaston.

Ominaisuuksia tuntuu löytyvän melko paljon, C++ maailmaan tottuneen onkin hankala löytää haluamansa C kirjaston dokumentaatiosta /4/. Tyypillistä C maailmasta ovat funktiot joille pitää kertoa kaikki mahdollinen jotta homma hoituu, tästä johtuen ei ole tavatonta että joudutaan pallottelemaan erilaisia tietorakenteita suuntaan jos toiseen.

Dokumentaation /3/ sekavuus sekä laajuus luovat ympäristön jossa on hankalaa päästä vauhtiin. Kaikille asioille on tehty jokin oma funktionsa, eivätkä ne tunnu olevan missään selvässä järjestyksessä. Usein funktiomäärittelyissä kerrotaan jokin enumeraatioarvo joka pitäisi antaa, mutta linkki enumeraatioarvon määrittelyyn puuttuu.

Tämän kirjaston päälle on todellakin kannattavaa tehdä rajapinta, jonka kautta hommat hoituvat helpommin. Lisäksi, jos rajapinta kerran tehdään, on se mahdollista siirtää lähes mihin tahansa ympäristöön, kirjaston alustatuen vuoksi.

Loputtomalta tuntuvan dokumentaation seasta ei kuitenkaan tuntunut löytyvän tukea mm XML Schema hyväksyttämiselle, tai XPath hakujen suorittamiselle jostain tietystä solmusta eteenpäin.

XML Scheman tukemisesta löytyi joitain funktioita, mutta lopulta todettiin sen olevan kehitteillä.

XPath hakujen suorittamisesta jostain tietystä pisteestä eteenpäin ei mainittu sanallakaan. On toki mahdollista että tämä funktio on jostain syystä jäänyt huomaamatta, mutta edes esimerkeissä mitään vastaavaa ei käytetty. XPath haussa käytetään erityistä tietorakennetta, jossa on talletettuna erinäistä tietoa haun suorittamiseen, näiden tietojen joukossa ei kuitenkaan näytä olevan tietoa siitä mistä eteenpäin haku suoritetaan.

Tällaisen XPath haun puuttuminen tekee muutamista rajapintaan määritellyistä funktioista erityisen ongelmaisia. Kaikki funktiot joissa haetaan jostain tietystä pisteestä eteenpäin, täytyy tehdä jollain vaihtoehtoisella tavalla. Tosin, koska suunniteltiin funktio jolla saadaan XPath johonkin annettuun solmuun, voidaan tähän XPathin perään liittää haettava XPath. Ongelmaksi tästä muodostuu toiminnon hitaus. Tällä tavalla haettuna kuluu enemmän aikaa kuin että haku suoritettaisiin suoraan juuresta.

3.3.7 Yhteenveto XML kirjastoista

Kirjastoihin tutustumisen pohjalta kävi ilmi, että XML rakennetta käsittelevien kirjastojen välillä on merkittäviä eroja. Voidaan todeta, että on oleellista kartoittaa projektissa tarvittavat XML toiminnot, sekä valita niiden pohjalta sopivin XML kirjasto tarpeiden täyttämiseen. Voi olla, että joudutaan jättämään pois joitain ominaisuuksia, mikä voi vaikuttaa oleellisesti joidenkin osien ratkaisumalleihin, sekä varsinaisen toteutuksen tekemiseen.

Pelkkien XML käsittelyyn liittyvien ominaisuuksien lisäksi tulee huomata kirjaston saatavuus eri alustoilla, sekä tähän liittyen kirjaston vaatimat muut komponentit jne.

4 CANopen konfiguraatio

Tässä kappaleessa otetaan käyttöön aikaisemmin tehtyjä johtopäätöksiä, luomalla CANopen tietoliikenteen konfigurointi XML tietorakenteen kautta. Luodaan pohja johon konfiguraatiota voidaan laajentaa, mutta ei luoda XML Schema tai DTD rakennetta, sillä käytetty XML kirjasto ei näitä tue. Määrittelevän tiedoston puuttumisen vuoksi luodaan dokumentaatio jossa kerrotaan miten eri asiat määritellään XML tiedostossa, sekä luodaan parsinta siten että vääränlainen rakenne ei aiheuta muuta kuin puuttuvia konfiguraatioita. Konfigurointiin liittyvää dokumentaatiota ei voida liittää tähän työhön, sillä se sisältää tietoja asiakkaan muista järjestelmistä.

Lisäksi tehdään lyhyt CANopen esittely, jossa käydään läpi muutamia oleellisimpia piirteitä CANopen tiedonsiirtoon liittyen.

4.1 CAN

CAN-väylä alun perin suunniteltiin käytettäväksi ajoneuvojen laitteiden tietoliikenteen väylänä. Kyseessä on siis väyläpohjainen ratkaisu, jossa useat laitteet kommunikoivat jaetun väylän kautta. Väylä koostuu kahdesta johtimesta.

4.2 CANopen

CANopen toimii CAN-väylän päällä, ts. suuremmalla OSI-kerroksella. Se määrittää erityisiä tapoja tiedonsiirron toteuttamiseksi eri osapuolten välillä ja lisäksi erityisiä alustukseen liittyviä viestejä. Tiedonsiirron lisäksi määritellään tiedonsiirtoon liittyen erityinen muistiavaruus, jota käytetään sekä vastaanotetun tiedon tallentamiseen sekä tiedon lähettämiseen muille. Muistiavaruudesta käytetään nimeä Objekti Kirjasto (eng Object Dictionary). Tiedonsiirrossa käytetään kahta eri tapaa, PDO ja SDO muotoista tiedonsiirtoa. Nämä tiedonsiirron eri muodot on selitetty myöhemmin tässä kappaleessa.

Objektikirjastossa tieto jaotellaan kaksi-ulotteiseen ohjelmistolliseen taulukkoon. Taulukossa ensimmäisenä osoittajana (eng index) on nelinumeroinen etumerkitön heksaluku. Toisena on ns. aliosoittaja (eng subindex) joka on kaksinumeroinen

etumerkitön heksaluku. Taulukon yhdessä kohdassa on vain osoittaja varsinaiseen tietoon ja tieto varsinaisen tiedon tyypistä. Lisäksi tallennetaan tieto objekteja koskevista kirjoitus- ja lukuoikeuksista.

SDO viestintä sitoo aina vain ja ainoastaan kaksi osapuolta. SDO viestin kautta on mahdollista pyytää toiselta laitteelta arvon sen objektikirjastosta tai kirjoittaa jokin arvo toisen laitteen objektikirjastoon. Viestin suorittamiseen tarvitaan kohteen objektikirjaston kohdan parametrien lisäksi tieto kohdelaitteen tunnuksesta, mielekästä on myös tarkistaa paluuviestissä tulevan tiedon koko, vaikka tätä CANopen speksi ei määrittelekään.

PDO viestintä toimii siten, että väylällä kulkee tietyillä tunnuksilla erimittaista tietoa. Tämä tieto kulkee kaikille ja vastaanottaja päättää arvot jotka kiinnostavat. Kiinnostavat arvot siirretään automaattisesti objektikirjastoon. Lisäksi on mahdollista merkitä jotkin tietyt objektikirjaston kohdat lähetettäväksi PDO dataksi, jolloin tämä data lähetetään väylälle kun sen arvo muuttuu.

Näiden lisäksi, projektissa johon tämä työ tehtiin, tarvittiin tiedonsiirtoa kahden eri väylän välillä, jolloin muodostettiin yhdyskäytävä kanavien välille, joka siirtää tietyillä tunnisteilla olevat viestit väylältä toiselle, muuttaen viestin tunnistetta siirrettäessä.

4.3 XML tiedostoformaatin muodostaminen

Tässä kappaleessa muodostetaan edellä esitellylle CANopen järjestelmälle XML-pohjaisen toiminnan määrittelyssä käytetyn tiedoston formaatti. Canopen järjestelmän eri osat on eroteltu eri otsikoiden alle.

Oletetaan että CANopen määrittely pysyy samanlaisena, eikä mahdollisesti joskus tulevaisuudessa tuleville uusille ominaisuuksille ole tarvetta. Näin päädytään ratkaisuun, jossa tiedot tallennetaan muuttujiin. Näin säästetään tiedoston koossa ja siten myös tiedoston lataamisen nopeudessa.

4.3.1 Objektikirjasto

Objektikirjastot tulee määritellä jokaiselle kanavalle erikseen. Tämä määritellään ensimmäiseksi määritteleväksi tekijäksi. Kanavalle tasolla määritellään myös kanavalla käytetty väylänopeus, vaikka se suoraan ei liitykään Objektikirjaston määrittelyyn.

Seuraavalla tasolla tässä vaiheessa määritellään tälle laitteelle erityinen tunniste. Tämän tason alle lisätään kaikki Objektikirjaston pääindeksoijat.

Pääindeksoijien alla määritellään ko. pääindeksille kuuluvat ali-indeksit, joille määritellään myös muuta tietoa. Jokaiselle ali-indeksoijalle määritellään ali-indeksi luvun lisäksi CANopen määrittelyn mukaista tietoa, osoitteessa olevan tiedon tyyppi, käsittelyoikeudet ja lisäksi oletusarvo joka asetetaan ko. paikkaan Objektikirjastoa alustettaessa. CANopen määrittelyn mukaisesti ensimmäiseen ali-indeksiin, arvolla 0, pitäisi tallentaa määriteltyjen ali-indeksien määrä, mutta tässä se on jätetty avoimeksi, sillä Objektikirjaston määrittely tähän työhön liittyvässä projektissa on tehty hieman erilailla. Toisaalta kun määrittely tehdään näin, ei estetä mitenkään CANopen määrittelyn mukaisen Objektikirjaston tekemistä, ainoastaan annetaan käyttäjälle enemmän mahdollisuuksia Objektikirjaston sisällön hallintaan.

4.3.2 PDO määrittely

Koska PDO määrittely on myös kanavakohtainen, se asetetaan Objektikirjastot otsikon alla esitetyn kanavan määrittelyn alapuolelle.

Määrittelyyn tarvitaan erikseen määritelty laitteen tunniste joka määritellään tasolla jonka alle kootaan kaikki PDO määrittelyt. PDO määrittelyt lisätään erikseen omalle tasolleen, jotta saadaan XPath haut kohdistettua paremmin niihin, tai ohi niistä.

Seuraavalla tasolla määritellään ko. PDO määrittelyn suunta, vastaanotettava vai lähetettävä PDO. Elementin nimi määrää, onko kyseessä lähetettävä, ”TX”, vai

vastaanotettava, ”RX”, viesti. Tällä tasolla määritellään myös sen viestin tunniste, jota tämä määrittely koskee, lisäksi määritellään PDO:n lähetystapa.

Tämän tason alapuolella määritellään, mihin kukin tavu PDO määrittelyn mukaisesti siirretään. Määritellään siirrettävän tiedon määrä, sekä Objektikirjaston sijainti. Tämän lisäksi määritellään erityinen määrittelytyyppi.

Olisi tietysti mahdollista määrittää Objektikirjaston kaikilla kohdille jokin tietty tunniste, mikä vain annettaisiin tässä PDO määrittelyssä. Tämä helpottaisi varsinaisen toiminnan määrittelyn tekemistä huomattavasti, mutta määrittelyn tutkiminen ohjelmasta ei olisi enää niin suoraviivaista. Lisäksi törmätään ongelmiin kun käyttäjä ei välttämättä tiedä kaikkien objektikirjaston eri kohdissa olevien kenttien kokoa.

4.3.3 SDO määrittely

Kuten aikaisemmatkin määrittelyt, SDO määrittely koskee aina jotain tiettyä kanavaa. Tämän johdosta SDO määrittely tulee kohdassa Objektikirjaston määrittely, esitellyn kanavajaon alle. Koska jokainen on hyvin erilainen, ei voida niiden suhteen tehdä määrittelyä aikaisemmalla tasolla, joten luodaan ryhmä johon SDO viestit kootaan, XPath hakujen nopeuttamiseksi.

Tämän alle luodaan kaikki XPath kyselyt, niitä ei voida sen tarkemmin ryhmitellä, sillä käyttäjä hakee niitä erityisen nimen perusteella.

SDO viesti koskee aina jotain tiettyä muuta laitetta CAN väylällä, siksi ko. laitteen tunniste kuuluu jokaisen viestin määrittelyyn.

Käsky kohdistetaan myös aina johonkin tiettyyn kohtaan toisen laitteen Objektikirjastossa, siksi on tarpeen esitellä osoittaja ja aliosoittaja, sekä tiedon tyyppi. Tiedon tyyppiä käytetään varmennettaessa tiedon tyypin täsmäävyyttä kohteen tiedon tyypin kanssa, varmennus tapahtuu pelkästään tiedon koon perusteella, joten on mahdollista sekoittaa etumerkillisiä ja etumerkittömiä arvoja. Kirjoittaessa toisen laitteen Objektikirjastoon tarkastelun tekee laite jonka

objektikirjastossa kirjoituksen kohde on. Luettaessa arvoa toisen laitteen objektikirjastosta tarkastelun tekee vastaanottava laite.

Lisäksi on tarpeen esitellä erityinen nimi-kenttä, jonka avulla voidaan tunnistaa kukin SDO viesti.

4.3.4 Yhdyskäytävän määrittely

Tämä määrittely koskee aina kahta jotain tiettyä kanavaa, jolloin kanava jolta viestit poimitaan, määritellään Objektikirjaston määrittelyssä esitellyssä kanavajaossa.

Kanavajaon alle luodaan määrittely, jossa kerrotaan kohdekanava. Lisäksi määritellään suodatuksen tyyppi, estetäänkö ne joita tässä ei ole esitelty, vai estetäänkö muut joita tässä ei ole esitelty.

Varsinaiset siirrot esitellään tämän alla. Mikäli kyseessä on toiminto, jossa vain tässä esitellyt tunnisteet siirretään, ei määritellä erikseen missään mitään tunnisteen muutosta. Näin esitellään vain tunnisteet joiden siirto estetään.

Mikäli halutaan käyttää toimintoa jossa siirretään vain ne viestit joiden tunniste on esitelty, määritetään myös erikseen tunniste joksi viestin tunniste muutetaan kun se lähetetään toisella kanavalla.

4.4 Tiedoston toteuttaminen

Tiedoston toteuttamisessa käytetään tässä dokumentissa esitettyjä keinoja tiedonhaun nopeuttamiseksi jo tiedostoformaattia suunniteltaessa. Suunnittelun kannalta oletetaan että tiedosto ladataan XML DOM rakenteeksi.

Formaatin muodossa kiinnitetään huomiota tiedon jäsentelyyn sekä nimeämisiin mahdollisimman nopean toiminnan takaamiseksi, joskin huomiotta jätetään helppolukuisuus.

Toteutettu XML tiedosto esiteltynä kuvassa 11, tiedoston tarkempaa sisältöä ei käydä tässä rivi-riviltä läpi, mutta jos tutkitaan määrittelyä ja tarkastellaan tiedoston rakennetta samanaikaisesti, nähdään miten eri asiat on toteutettu.

Eri asiat on pyritty ilmaisemaan mahdollisimman lyhyillä nimillä, lyhenteitä on myös käytetty mahdollisimman paljon. Lyhyet nimet nopeuttavat sekä tiedoston varsinaista lataamista muistiin, että XPath hakuja, jolloin kaikki XPath hakuihin pohjautuvat tekniikat, kuten XML Schema-tarkistukset, nopeutuvat. Nimeämisessä on pyritty lyhyyden lisäksi pitämään rinnakkaiset nimet eri alkukirjaimella. Lisäksi tiedon hakeminen suoraan XML DOM rakenteesta nopeutuu.

```
<Conf>
 <Canopen>
  <Channel Nr="0" BR="500">
   <GW>
    <Gate Type="block" Dest="1">
     <Ent SrcID="0x000" DstID="0x001" />
     <Ent SrcID="0x18F" DstID="0x18B" />
    </Gate>
   </GW>
   <OD DevID="10" >
    <Ind Ind="0x2001" >
     <Obj Name="ODEntry 1" SubInd="0x0" Type="CO_RW" Data="CO_U16" Value="0" Size="2" />
    </Ind>
    <Ind Ind="0x2002" >
     <Obj Name="ODEntry 2" SubInd="0x0" Type="CO_RW" Data="CO_U16" Value="0" Size="2" />
    </Ind>
   </OD>
   <PDO DevID="10" >
    <RX ID="0x18D" Type="255">
     <Map SubInd="0x0" Type="255" Len="1" Ind="0x2104" />
     <Map SubInd="0x1" Type="255" Len="1" Ind="0x2104" />
     <Map SubInd="0x2" Type="255" Len="1" Ind="0x2104" />
    </RX>
   </PDO>
   <SDO>
    <Msg Name="Msg3" DevID="1" SubInd="0x3" Type="CO_WO" Data="CO_S32" Size="4" Ind="0x2007" />
    <Msg Name="Msg4" DevID="1" SubInd="0x6" Type="CO_RO" Data="CO_S32" Size="4" Ind="0x2007" />
   </SDO>
  </Channel>
  <Channel Nr="1" BR="500">
   <GW>
    <Gate Type="block" Dest="0" >
     <Ent SrcID="0x088" DstID="0x094" />
     <Ent SrcID="0x188" DstID="0x194" />
    </Gate>
   </GW>
   <OD DevID="10" >
    <Ind Ind="0x2108" >
     <Obj SubInd="0x0" Type="CO_RW" Data="CO_U16" Value="0" Size="2" />
     <Obj SubInd="0x1" Type="CO_RW" Data="CO_S16" Value="0" Size="2" />
     <Obj SubInd="0x2" Type="CO_RW" Data="CO_U16" Value="0" Size="2" />
    </Ind>
   </OD>
   <PDO DevID="10" >
    <RX ID="0x288" Type="255">
     <Map SubInd="0x0" Type="255" Len="2" Ind="0x2108" />
     <Map SubInd="0x1" Type="255" Len="2" Ind="0x2108" />
     <Map SubInd="0x2" Type="255" Len="2" Ind="0x2108" />
    </RX>
   </PDO>
   <SDO>
    <Msg SubInd="0x0" DevID="8" Type="CO_RW" Data="CO_S16" Size="2" Ind="0x2038" />
   </SDO>
  </Channel>
 </Canopen>
</Conf>
```

Kuva 11 CANopen viestinnän määrittelevä XML tiedosto

4.5 Tietojen hakeminen

Tässä kappaleessa tehdään edellä esiteltyyn tiedostoformaattiin hakutoimintojen muodostaminen ja siihen tarvittavat luokkarakenteet.

XPath hauista mainittakoon yleisesti, että ne on tehty toimimaan mahdollisimman nopeasti em. XML tiedoston toteutuksen kanssa. Niissä ei määritellä nimen tarkastelua kohdissa joissa se ei ole tarpeen. Esim. haku /Conf/Canopen/Channel/GW on tuloksiltaan täysin identtinen haun /*/*/*/GW kanssa. Erona on se, että jälkimmäisessä haussa tarkastetaan vain viimeisen elementin nimi, kun edeltäneessä tarkastellaan kaikki elementtien nimet juuresta alkaen. Yksittäisen haun kannalta käyttäjä ei havaitse eroa, mutta laajan laitekonfiguraation lataamisessa havaitaan jo merkittävä ero. Tähän projektiin liittyvässä asiakasprojektissa havaittiin alussa olevan latausajan tippuminen 1minuutista ja 7 sekunnista 26 sekuntiin. Edellisen korjauksen lisäksi tehtiin tietojen haun optimointia tapauksissa, jossa haetaan suuri osa tietoja.

4.5.1 Yhdyskäytävän lataaminen

Asetukset haetaan pääosin vain ohjelman toimintaa alustettaessa, jolloin ne haetaan kaikki kerralla. Yksittäiset haut suoritetaan XPath kyselyllä, joka on hieman epätehokas, mutta oletetaan että näitä tietoja ei juurikaan haeta yksittäisinä. Yksittäisessä haussa pyydetään määrittelemään lähtö- ja tulokanavan lisäksi alkuperäisen viestin tunniste. Tällöin XPath haun muoto on.

//*[@Nr[.='Lähtökanava']]/GW/*[@Dest[.='*
Kohdekanava']]/[@SrcID[.='Tunniste']]*

Kaikkien tietojen haku alussa suoritetaan siten, että haetaan ensin XPath haulla kaikki kuvassa 6 esitetyt *Ent* elementit. XPath haun komento on */*/*/GW/*/*. Näin päästään mahdollisimman pienellä määrällä nimien tarkasteluja haettua kaikki tarvittavat elementit. Toisaalta tällainen haku muuttuu hieman hitaammaksi jos XML tiedostoa laajennetaan, silloin täytyy tutkia, josko jotkin nimet pitää lyödä lukkoon haun rajoittamiseksi aikaisemmassa vaiheessa.

Kun tästä polusta on haettu kaikki elementit, käydään kaikki elementit läpi ja haetaan tarvittavat arvot jokaisen elementin kohdalla suoraan DOM rakenteesta. Suoraan *Ent* elementiltä haetaan ainoastaan *SrcID* ja *DstID* tiedot, jotka voidaan eritellä jo ensimmäisen kirjaimen perusteella.

Elementiltä *Gate*, joka siis on *Ent* elementin isäntäelementti, haetaan kohdekanava, sekä portin tyyppi. Viimeinen arvo, lähtökanavan arvo, haetaan elementin *Gate*, isäntäelementin isäntäelementiltä.

Kun arvot palautetaan yksittäisen kyselyn vastauksena, ne tallennetaan muotoon jossa nimi ja arvo liitetään yhteen, tässä työssä käytettiin *map<string,string>*-rakennetta.

4.5.2 Objektikirjaston lataaminen

Objektikirjasto ladataan, kuten yhdyskäytävä-asetukset, ohjelman toimintaa alustettaessa. Toisaalta objektikirjaston tietoja tarvitaan myös ohjelman ajon aikana, luettaessa tai kirjoitettaessa tietoa sinne.

Yksittäisen tiedon käsittelyssä, tähän työhön liittyvässä projektissa, tarvitaan usein tiettyjä objektikirjaston asioita, jolloin ne kannattaa ladata omaan muistirakenteeseensa XML DOM rakenteesta. Tämä tehdään muodostamalla luokka joka sisältää tarvittavat tiedot kirjoitus- ja lukuoperaatioita varten. Varsinainen tiedon hakeminen tapahtuu erityisellä nimellä joka on jokaisella objektikirjaston elementillä erilainen.

Luokalle tehdään myös erityinen staattinen muodostin-funktio, joka yrittää ladata kaikki tarvittavat tiedot ja muodostaa luokan instanssin näiden pohjalta. Mikäli tarvittavia tietoja ei ollut saatavilla, palautetaan ilmoitus virheestä. Tälle funktiolle annetaan arvona XML DOM puun solmu, josta tiedot haetaan suhteellisien polkujen takaa.

Muodostin-funktiolle annettava XML DOM solmu haetaan XPath käskyllä, joka on muotoa */*/*/*/OD/*/*[@Name[.='Nimi'].]*.

Olioon tallennetaan huomattava määrä erilaista tietoa. Kanavanumero on oleellisin tieto, jonka jälkeen tulevat osoittaja ja aliosoittaja. Näiden lisäksi tarvitaan myös tieto tiedon tyypistä ja tietoon liittyvät käsittelyoikeudet, sekä tiedon oletusarvo ja tiedon tyypin koko tavuina. Nämä tiedot haetaan XPath haun antamasta solmusta, sekä tietyiltä sen isäntäsolmuista. Tarkemmat paikat näkyvät kuvassa 6.

Ohjelman alussa tapahtuvaan alustukseen haetaan kaikki XPath polulla */*/*/*/OD/*/** löytyvät elementit, joista haetaan sitten kaikki tarvittavat tiedot. Näin saadaan kerrottua CANopen järjestelmälle kaikki objektikirjastoon luotavat muistikentät.

4.5.3 PDO tietojen lataaminen

PDO asetukset haetaan pääasiassa ohjelman alussa, kaikki kerralla. Tällöin tyydytään yksittäisen yhdyskäytävä-asetuksen haun kanssa samankaltaiseen XPath haku-ratkaisuun. PDO tiedon hakeminen ajatellaan siihen liitetyn objektikirjaston kohdan kanssa. Haku tapahtuukin ensin hakemalla objektikirjastosta jokin tietty kohta nimen perusteella. Tästä objektikirjaston kohdasta sitten tutkitaan sen osoittajat, joiden perusteella tutkitaan PDO viesti joka on määritelty talletettavaksi ko. objektikirjaston kohtaan.

Varsinainen haku tapahtuu edellisessä kappaleessa esitellyllä objektikirjaston tietojen hakemisella nimen perusteella, tähän lisäksi liitetään XPath haku jolla etsitään ko. PDO. XPath kyselyn rakenne on */*/*/*[@Nr[.='kanava']]/PDO/*/*[@Ind[.='osoittaja'] and @SubInd[.='aliosoittaja']]*.

Kun arvot palautetaan yksittäisen kyselyn vastauksena, ne tallennetaan muotoon jossa nimi ja arvo liitetään yhteen, tässä työssä käytettiin *map<string,string>*-rakennetta.

Ohjelman toimintaa alustettaessa kaikki PDO määrittelyt ladataan kerralla hakemalla solmut XPath kyselyllä */*/*/*/PDO/*/**. Kyselyn tuloksena saaduista elementeistä ladataan arvot jotka sitten kerrotaan CANopen järjestelmälle.

4.5.4 SDO tietojen lataaminen

SDO viestit ovat määrittelyn ainoa osa jota ei pidä ladata ohjelman käynnistyessä, niitä tullaan pyytämään vain yksittäisinä kyselyinä.

SDO viestejä varten luodaan oma luokkansa, joka sisältää samankaltaisen muodostin-funktion kuin objektikirjaston määrittelyssä on esitelty.

Kuten objektikirjastoon liittyvässä luokkarakenteessa, tässäkin muodostin-funktio ottaa arvona XML DOM solmun, josta se etsii luokan tarvitsemia tietoja joko suoraan tai suhteellisilla poluilla. Mikäli tarvittavia tietoja ei saada, palautetaan virheilmoitus.

Luokka sisältää useita eri tietoja, mutta vain tieto kanavasta haetaan muualta kuin annetulta XML DOM elementiltä. Kanavan jälkeen tärkeysjärjestyksessä on kohdelaitteen tunnus, sitä seuraavat osoittaja ja aliosoittaja. Lisäksi haetaan eri tiedot käsiteltävästä tiedosta, tiedon koko, tyyppi sekä käsittelyoikeudet.

SDO viestit eritellään niihin merkityn nimen perusteella, jonka mukaan myös XPath-haku tapahtuu. SDO viestit haetaan XPath komennolla */*/*/*/SDO/*[@Name[.='Nimi']]*.

4.6 Yhteenveto

Tässä ratkaisussa ei ole keskitytty tehostamaan suoriutumis- tai toteuttamisaikaa, vaan on keskitytty etsimään ratkaisu, joka miellyttää sekä toteutuksen rakentamisen maksavaa että toteutusta käyttävää tahoa.

Toteuttamisaikaa voitaisiin tehostaa käyttämällä enemmän XPath hakuja tiedon hakemiseen. Suoriutumisaikaa voitaisiin tehostaa mm jäsentelemällä useimmin kysytyt arvot eri paikkaan, josta niiden hakeminen olisi nopeaa. Yksityiskohtaisemmalla tasolla olisi mahdollista tehdä kerrostetun tiedon hakeminen lähtien liikkeelle juurielementtiä lähimmästä tietoa sisältävästä solmusta.

XML tiedoston rakenteen muotoilu on tehty säästämään tilaa, toistuvien nimien lyhentäminen on eräs esimerkki tästä. Tiedosto ja siihen liitetyt tiedon hakemiseen liittyvät polut on kuitenkin tehty siinä mielessä huonosti, että tiedoston laajentaminen on hankalaa. Toisaalta, kyseessä laajentaminen ei juuri ole tarpeen, sillä lähes kaikki CANopen asiat on määritelty tiedostossa.

Tiedoston rakenteeseen liittyen olisi myös mahdollista muuttaa se laitetta varten johonkin eri muotoon, esimerkiksi suoraan binaariksi joka voidaan suoraan ladata CANopen rajapintaan. Tämän tekemiseen voidaan käyttää jo tehtyjä luokkia ja tiedon hakuun liittyviä toteutuksia, jolloin tässä määriteltyjä tekniikoita voidaan käyttää uudelleen.

5 Lähteet

Sähköiset:

1 World Wide Web Consortium (W3C) tutoriaalit
www.w3schools.com

2 Canopen Manuaali
www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf

3 libxml2 API
http://xmlsoft.org/index.html

4 Qt XML API
http://doc.trolltech.com/3.3/xml.html

Kirjallisuus:

4 XML 1.1 Bible 3rd edition Elliotte Rusty Harold. Wiley publishing, Inc. 2004

5 XML Problem-Design-Solution, Mitch Amiano, Conrad D'Cruz, Kay Ethier, Michael D. Thomas. Wiley publishing, Inc. 2006

6 XML For Dummies 4th edition, Lucinda Dykes, Ed Tittel. Wiley publishing. Inc. 2005.