Valtteri Ahlberg

XML-tiedostojen vertailija

Opinnäytetyö
Kevät 2020
SeAMK Tekniikka
Automaatiotekniikka

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Koneautomaatio

Tekijä: Valtteri Ahlberg

Työn nimi: XML-tiedostojen vertailija

Ohjaaja: Marko Hietamäki

Vuosi: 2020 Sivumäärä: 34

Opinnäytetyö suoritettiin Finn-Power Oy:lle. Finn-Power Oy on Seinäjoella sijaitseva metalliteollisuuden yritys, joka luo asiakkaille tilaustöinä ohutlevyjen työstöön suunniteltuja järjestelmä- ja laiteratkaisuja. Yritys myös kehittää omat ohjelmistot laitteiden ohjaamiseen. Ohjelmistojen ylläpito ja jatkokehitys on monimutkaista ja hienovaraista. Muutokset ohjelmoinnissa saattavat johtaa ennustamattomiin virheisiin. Nämä ennustamattomat virheet pyritään huomaamaan ohjelmistotestauksen avulla.

Yrityksen ohjelmistot tuottavat erilaisia tiedostoja, joiden oikeellisuutta halutaan varjella. Oikeellisuuden varmistamiseen päätettiin rakentaa tiedostojen vertailija, joka oli opinnäytetyön aihe. XML-tiedostojen vertailija rakennettiin C++-ohjelmointikieltä käyttäen. Työssä käytettiin hyväksi pugixml-kirjastoa, joka mahdollisti XML-formaatin helpon lukemisen ohjelmassa.

Opinnäytetyön lopputuloksena saatiin toimiva ja yksinkertainen XML-tiedostojen vertailija, joka rakennettiin suoraan yrityksen NCX-ohjelman testaushaaraan. XML-tiedostojen vertailija saavutti määrätyt tavoitteet, mutta jättää kuitenkin mahdollisuuden jatkokehittämiseen.

Avainsanat: C++, XML, NCX, pugixml, ohjelmistotestaus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Technology

Specialisation: Machine Automation

Author: Valtteri Ahlberg

Title of thesis: XML Document Comparator

Supervisor: Marko Hietamäki

Year: 2020 Number of pages: 34

This thesis was carried out for Finn-Power Oy. Finn-Power Oy is a company in metal industry. It is located in Seinäjoki, Finland. Finn-Power Oy creates to its customers customised systems and machines designed for sheet metal tooling. The company also builds and upkeeps its own software to control the said machinery. The upkeeping and advancement of software is complex and sensitive. Changes in the programming can lead to unexpected faults. These unexpected faults can be discovered by utilising software testing.

The company's software creates documents that must be valid. To ensure the validity, the company decided to create a document comparator. The topic of this thesis was the development of that XML document comparator. The comparator was programmed utilising C++ programming language. The work also took advantage of the pugixml library, which enabled convenient reading of the XML format within the program.

As the result of the thesis a functional and straightforward XML document comparator was built directly into the testing branch of the NCX program. The XML document comparator achieved the specifications allotted to it, whilst permitting future improvements to it.

Keywords: C++, XML, NCX, pugixml, software testing

Sisältö

Kuvaluettelo

Käytetyt termit ja lyhenteet

C++	C++ on Bjarne Stroustrupin 1980-luvulla kehittämä korkean tason ohjelmointikieli.
CAD	CAD eli tietokoneavusteinen suunnittelu (Computer-aided design) on tietokoneiden hyödyntämistä työn suunnitteluvaiheessa.
CAM	CAM eli tietokoneavusteinen valmistus (Computer-aided manufacturing) on tietokonesovelluksien käyttöä työstökoneiden ohjaamiseen.
NC Express (NCX)	PrimaPowerin rakentama ja ylläpitämä CAD/CAM-sovellus. Voidaan käyttää automaattisena työstöohjelmistona, sekä yksittäisten kappaleiden työstöön.
Ohjelmistotestaus	Ohjelmiston virheettömyyttä ja laatua selvittävä prosessi.
pugixml	Kevyt XML:n prosessointiin käytettävä C++-kirjasto.
XML	Extensible Markup Language on tiedon rakennetta kuvaava metakieli.

1 Johdanto

1.1 Työn tausta

Sovellusta kehitettäessä tiedostojen luonti saattaa muuttua tavalla, jota ei osata odottaa. Tällöin on tärkeää, että muutokset pystytään huomaamaan, tiedostamaan ja niihin pystytään reagoimaan tarvittavalla tavalla. Tähän tarkoitukseen tarvittiin tiedostojen vertailija, joka vertailee vanhaa tiedostoa ja uutta tiedostoa keskenään, ja huomioi erot niiden välillä. Sopivaa vertailijaa halutuilla toiminnoilla ei löytynyt markkinoilta, joten firmassa päätettiin rakentaa oma tiedostojen vertailija.

1.2 Työn tavoite

Tämän opinnäytetyön tavoitteena on rakentaa toimiva XML-tiedostojen vertailija, joka lopulta asetetaan NCX-sovelluksen kehittäjäympäristöön. Tarkoituksena on, että XML-tiedostojen vertailija vertailee hyväksi todettua referenssitiedostoa uuteen, juuri käännettyyn verrokkitiedostoon, ja etsii mahdolliset eroavaisuudet tiedostojen välillä. Mahdollisista eroavaisuuksista välittyy tieto ohjelmistokehittäjälle, joka voi perehtyä eroavaisuuksiin tarkemmin. XML-tiedostojen vertailijasta haluttiin alkuperäisformaattia taltioiva yksinkertainen vertailija.

1.3 Työn rakenne

Luvussa 2 tutustutaan XML-metakieleen ja sen rakenteeseen. Työn luonteen vuoksi on tärkeä ymmärtää XML-kielen perussanat.

Luvussa 3 on XML-tiedostojen vertailijalle luotu vaatimusmäärittely. Vaatimuksena vertailijalle olivat muun muassa rivinumeroiden ylläpitäminen ja alkuperäisformaatin taltioiminen.

Luvussa 4 käydään läpi vertailijan toteuttaminen.

Luvussa 5 on opinnäytetyön yhteenveto sekä pohdintaa.

1.4 Finn-Power Oy

Finn-Power Oy on levytyökoneisiin ja valmistusjärjestelmiin erikoistunut teollisuusyritys, jonka päätoimipiste on Seinäjoella. Yritys on osa Milanon pörssissä noteerattua Prima Industrie -konsernia. Yrityksen liikevaihto vuonna 2017 oli noin 450 miljoonaa euroa. Finn-Power Oy:n palveluiden ja tuotteiden tunnus on Prima Power. Tunnus Prima Power tunnetaan yhtenä maailmanmarkkinoiden johtavana levyntyöstökoneisiin ja -järjestelmiin erikoistuneena yrityksenä. (Työpaikat [Viitattu 23.3.2020].)

Prima Power toimittaa koneita ja järjestelmiä kaikkialle maailmaan sen tuotantoyksiköistä. Tuotantoyksiköitä Prima Powerilta löytyy Suomesta, Italiasta, Yhdysvalloista sekä Kiinasta. Prima Powerin erikoisvahvuuksia ovat koneiden ja solujen automatisointi halutulle tasolle sekä pitkälle kehitetyt palvelut. Prima Power on toimittanut yli 10 000 konetta ja järjestelmää. (Yritys [Viitattu 23.3.2020].)

Finn-Power perustettiin vuonna 1969. Finn-Power tuli aluksi tunnetuksi hydraulisilla letkuliitinpuristimillaan. Vuonna 1983 Finn-Power toi markkinoille kehittämänsä ja valmistamansa hydraulisen levytyökeskuksen. Finn-Power perusti ensimmäisen tytäryhtiönsä Yhdysvaltoihin vuonna 1985. Finn-Power jatkoi kehittymistään, ja lopulta se tuli osaksi Prima Industrie -konsernia vuonna 2008. (Historia [Viitattu 23.3.2020].)

2 Extensible Markup Language

Tässä luvussa tutustutaan XML-kieleen ja sen rakenteeseen. Työn luonteen vuoksi on tärkeä ymmärtää XML-kielen perusasiat. On liki mahdotonta rakentaa spesifin tiedostotyypin vertailijaa, jos ei ymmärrä mistä vertailtavien kohteiden eri osat koostuvat.

XML on tekstimuotoinen dataformaatti, joka on vahvasti tuettu Unicoden avulla eri ihmiskielille. Vaikka XML:n muotoilu keskittyykin dokumenteille, käytetään XML-kieltä laajasti esittämään mielivaltaisia datarakenteita. (Fennel 2013, 80.)

2.1 XML-kielen alkuperä ja tavoitteet

XML-kielen kehitys aloitettiin W3C:n (World Wide Web Consortium) alaisuudessa ja sen toimesta XML Working Groupissa vuonna 1996 (Extensible Markup Language (XML) 26.11.2008). Extensible Markup Languagen perusolemus löytyy sen nimestä (Myer 2006, 2).

XML on laajennettava (Extensible). XML antaa käyttäjälle mahdollisuuden määrittää omat taginsa, niiden järjestyksen sekä sen, miten ne käydään läpi (Myer 2006, 2).

Yksi XML:n tunnuspiirteistä on sen merkinnät (Markup) eli elementit. Tässä suhteessa XML on osittain samankaltainen HTML-kielen kanssa. XML kuitenkin mahdollistaa omien tagien määrittelyä toisin kuin HTML. (Myer 2006, 2.)

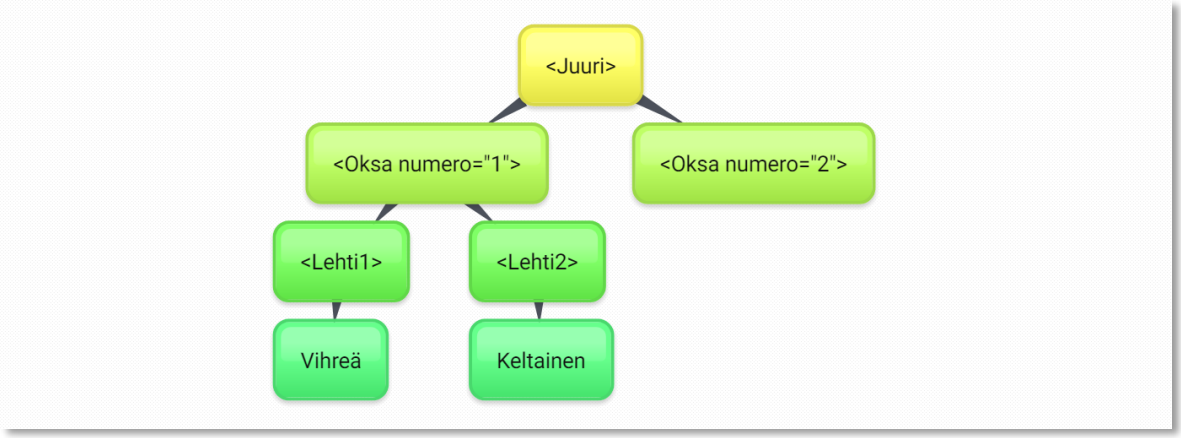
XML on kielenä (Language) jokseenkin samanlainen HTML:n kanssa. XML on kuitenkin paljon HTML-kieltä joustavampi tietyissä asioissa. XML ei kuitenkaan ole pelkkä kieli, vaan se on metakieli. Tämä tarkoittaa, että XML on kieli, joka mahdollistaa toisien kielien luomisen tai määrittelemisen. Yksi merkintäkieli, jonka luomiseen on käytetty XML-kieltä, on RSS. (Myer 2006, 2.)

2.2 XML-tiedoston dokumenttipuu

XML:n rakennetta voidaan verrata puuhun. XML-tiedosto alkaa juuresta ja johtaa oksien lehtiin. (Myer 2006, 11.)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Juuri>
3      <Oksa numero="1">
4          <Lehti1>Vihreä</Lehti1>
5          <Lehti2>Keltainen</Lehti2>
6      </Oksa>
7      <Oksa numero="2"></Oksa>
8  </Juuri>
```

Kuva 1. Esimerkki XML:n puumaisesta rakenteesta.



Kuva 2. XML-tiedosto havainnollistettuna.

Kuvassa 1 ja 2 havainnollistetaan XML-tiedoston rakennetta. Kyseessä oleva XML-tiedosto kuvailee puuta, jolla on yksi juuri, josta kaikki alkaa, kaksi oksaa, joista toisessa on kaksi lehteä. Nämä lehdet taas puolestaan sisältävät tietoa eli dataa.

Tässä tiedostosta löytyy elementeille sopivat sisennykset. Tagi *Oksa* on yhden askeleen oikeammalla verrattuna tagiin *Juuri*, sillä tagi *Oksa* kuuluu tagiin *Juuri*. Sisennys tai sen puuttuminen ei vaikuta XML-tiedoston toimivuuteen, mutta sen olemassaolo helpottaa hahmottamaan tiedoston rakenteen. XML-tiedostojen vertailija hyödyntää sisennyksiä vertailussa, joten takeita vertailijan toimivuudesta sisennyksen puuttuviin XML-tiedostoihin ei ole.

2.3 XML-terminologia

XML-kielellä on hierarkkinen struktuuri, ja sen eri osien ymmärtäminen on välttämätöntä hyvän kokonaiskuvan saamiseksi. XML-tiedostot koostuvat prologista, elementeistä, tageista ja attribuuteista. Näiden lisäksi XML-dokumentista saattaa löytyä kommentteja. (Extensible Markup Language (XML) 26.11.2008.)

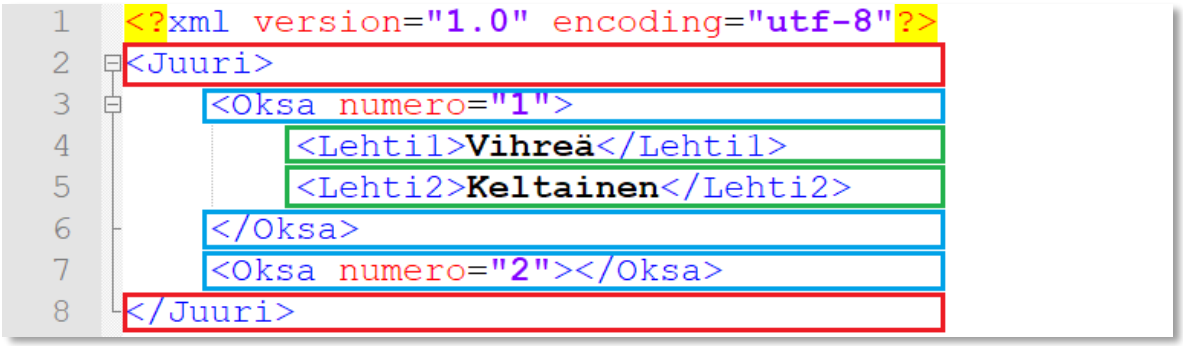
2.3.1 Prologi

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Juuri>
3      <Oksa numero="1">
4          <Lehti1>Vihreä</Lehti1>
5          <Lehti2>Keltainen</Lehti2>
6      </Oksa>
7      <Oksa numero="2"></Oksa>
8  </Juuri>
```

Kuva 3. XML-tiedoston prologi.

XML-tiedoston prologi, eli eräänlainen esipuhe, on korostettuna kuvassa 3. Prologissa voidaan esitellä tietoa käytettävästä XML-versiosta sekä siitä, mitä merkistöä tuetaan. Näiden lisäksi prologissa voidaan viitata rakennemäärittelytiedostoon tai XML-skeemaan. (XML [Viitattu 13.4.2020].)

2.3.2 Elementit



Kuva 4. XML-tiedoston elementit.

XML-elementti on kokonaisuus, joka sisältää kaiken alkaen elementin aloitustagista ja loppuen elementin lopetustagiin (Myer 2006, 6). Elementit ovat XML-tiedoston loogisia rakenneosia. Elementit voivat sisältää dataa tai toisia elementtejä. Kuvan 4 tapauksessa elementti *Juuri* sisältää kaksi lapsielementtiä *Oksa (rivi 3)*, ja *Oksa (rivi 7)*. Rivin 3 *Oksa* sisältää myös itsessään kaksi lapsielementtiä *Lehti1* ja *Lehti2*. *Lehti* -elementit taas sisältävät dataa eli sisältöä; *Lehti1* sisältää datan Vihreä, kun taas *Lehti2* sisältää datan Keltainen.

Kuvasta 4 löytyy yhteensä viisi eri elementtiä: *Juuri, Oksa, Lehti1, Lehti2* ja *Oksa*. Punaisella merkitty *Juuri*-elementti on nimeltään juurielementti (englanniksi root element). Se on ensimmäinen elementti, joka merkitään prologin jälkeen. Juurielementtejä on vain yksi XML-tiedostossa, minkä seurauksena kaikki muut elementit liittyvät juurielementtiin (Myer 2006, 14).

2.3.3 Tagit

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Juuri>
3      <Oksa numero="1">
4          <Lehti1>Vihreä</Lehti1>
5          <Lehti2>Keltainen</Lehti2>
6      </Oksa>
7      <Oksa numero="2"></Oksa>
8  </Juuri>
```

Kuva 5. XML-tiedoston tagit ja elementin sisällöt.

Tagit aloittavat ja lopettavat elementit (Myer 2006, 7). Kuvasta 5 löytyy yhteensä viisi tagiparia, eli viisi aloittavaa tagia ja viisi sulkevaa tagia. Sulkevan tagin erottaa siinä olevasta vinoviivasta (/), joka löytyy pienempi kuin -merkin (<) jälkeen ennen elementin nimeä. Aloittavan tagin ja sulkevan tagin täytyy vastata toisiaan, eli niiden nimien täytyy olla täysin samanlaiset. Tagit ovat myös merkkikokoriippuvaisia, joten tagi </oksa> ei sulje elementtiä, joka aloitettiin tagilla <Oksa>. Tyhjän elementin voi avata ja sulkea yhdellä tagilla kahden tagin sijaan. Rivin 7 voisi vaihtoehtoisesti siis kirjoittaa kuvassa 6 esitetyllä tavalla.

```
6      </Oksa>
7      <Oksa numero="2"/>
8  </Juuri>
```

Kuva 6. Elementin aloitus ja lopetus yhdellä tagimerkinnällä.

2.3.4 Attribuutit

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Juuri>
3      <Oksa numero="1">
4          <Lehti1>Vihreä</Lehti1>
5          <Lehti2>Keltainen</Lehti2>
6      </Oksa>
7      <Oksa numero="2"></Oksa>
8  </Juuri>
```

Kuva 7. XML-tiedoston attribuutit ja niiden sisällöt.

Attribuutit ovat elementteihin liitettäviä informaation palasia. Attribuutteja voidaan ajatella adjektiiveina – ne antavat lisää informaatiota elementistä, mitä ei olisi välttämättä järkevä antaa itse elementin sisällä. (Myer 2006, 8.) Attribuutit kertovat elementin ominaisuuksista. Kuvan 7 esimerkissä elementeille *Oksa* (rivi 3) ja *Oksa* (rivi 7) on annettu niitä erottavat numeroarvot 1 ja 2 attribuutteina. Attribuutit annetaan elementille sen aloitustagissa ennen suurempi kuin -merkkiä (>). Attribuutteja voi olla useita yhdellä elementillä, ja ne erotetaan välimerkein. Kyseessä on siis vaihtoehtoinen tapa asettaa tietoa elementille. Mikäli kuvan tapauksessa olisi haluttu antaa *Oksa*-elementin numeroarvo lapsielementtinä elementtien *Lehti1* ja *Lehti2* tavoin, olisi sekin ollut mahdollista.

3 Vaatimusmäärittely

Yrityksellä oli satunnaisesti ollut ongelmia nestauksesta syntyneiden XML-tiedostojen kanssa. Luotettavuutta ja vankkarakenteisuutta haluttiin nostaa tällä saralla, minkä vuoksi yrityksessä alettiin miettiä tapaa varmistua näiden tiedostojen virheettömyydestä. Harkinnan jälkeen yrityksessä päätyttiin ratkaisuun, jossa aikaisemmin muodostettuja referenssitiedostoja verrataan äskettäin muodostettuihin verrokkitiedostoihin. Käytettävät referenssitiedostot ovat täsmällisiä, joten verrokkitiedostojen mahdollinen erilaisuus kertoo todennäköisesti virheestä uuden tiedoston muodostavassa ohjelmointikoodissa.

Näin ollen tarvittiin tiedostojen vertailija. Markkinoilla on olemassa monia erilaisia tiedostojen vertailijoita, mutta sieltä löytyvät vertailijat eivät toteuttaneet kaikkia asetettuja vaatimuksia. Tämän vuoksi yritys päätti itse rakentaa omaan testiympäristöön asetettavan vertailijan.

Ennen työhön ryhtymistä palaverissa mietittiin tiedostojen vertailijaan liittyviä vaatimuksia. Alkuperäisformaatin säilyttäminen oli tärkeää, jotta kehittäjä voi helposti lukea vertailussa löydetyt eroavaisuudet. Luettavuuden vuoksi oli myös tärkeää, että XML-tiedostojen rivinumerot säilyisivät. Vertailijaan piti myös sisällyttää tiettyjen elementtien ja attribuuttien sivuuttaminen, eli niiden huomiotta jättäminen. Mikäli jokin näistä sivuutetuista elementeistä tai attribuuteista eroaisi alkuperäisestä, sillä ei saisi olla vaikutusta lopputulokseen.

Vertailijan jatkokehittävyyttä pidettiin myös tärkeänä, sillä osa tarpeista saatetaan huomata vasta vertailijan käyttöönoton jälkeen. Jatkokehittävyyden kannalta tiedostojen vertailijasta pyrittiin tekemään myös yksinkertainen. Yksinkertaisuus helpottaa myös muiden ohjelmistokehittäjien osallistumista jatkokehitykseen.

Nestaustiedostot ovat XML-tiedostoformaatissa. Tiedettiin, että vertailija tulee lukemaan juuri tämänlaisia XML-tiedostoja, joten vertailijasta tehtiin juuri XML-tiedostoihin perustuva vertailija. XML-tiedostojen rakenteellisuutta päätettiin käyttää hyödyksi vertailun eri vaiheissa. Tämän vuoksi päätettiin ottaa käyttöön myös pugixml C++ -kirjasto.

Yksi asia, jota pidettiin myös tärkeänä, oli jonkinlainen tapa, jolla ohjelmistokehittäjä voi perehtyä tarkemmin testeistä löytyviin eroavaisuuksiin. Tavoitteena oli saada tekstitiedosto tai vastaava, josta löytyisi tarkasti eroavaisuuksien määrä sekä paikka tiedostoissa.

Lopputuloksena haluttiin yölliseen koontiversioon liitettävä XML-tiedostojen vertailija, joka vertailee hyväksi todettua referenssitiedostoa uuteen, juuri käännettyyn verrokkitiedostoon, ja etsii sieltä mahdolliset eroavaisuudet. Mikäli eroavaisuuksia löytyy, välittyy tieto ohjelmistokehittäjille. Tiedon saatuaan ohjelmistokehittäjä voi halutessaan tarkistaa epäkohdat tiedostosta, joka sisältää ne. XML-tiedostojen vertailijan kehittyessä pidettiin monia kokouksia, joissa katsottiin vertailijan tilannetta, ja säädettiin tavoitteita tarpeiden mukaan.

4 Toteutus

XML-tiedostojen vertailijaa rakennettiin proseduraalisesti. Sen perustana toimivat luvussa **3** annetut vaatimusmäärittelyt. Perusideaksi muodostui nopeasti ajatus yksittäisien rivien vertailusta, jossa molempien XML-tiedostojen jokainen rivi käydään läpi yksi kerrallaan. Yksittäisien rivien vertailu mahdollistaa alkuperäisformaatin säilyttämisen. Se myös mahdollistaa rivinumeroiden ylläpidon. Kun vertailijaa rakennettiin tämän idean ympärille, helpotti se vaatimusten täyttämistä heti alusta lähtien.

Kolmas vaatimus, jota pyrittiin noudattamaan heti alusta alkaen, oli tiettyjen attribuuttien ja elementtien sivuutusmahdollisuus. Tämä vaatimus silmällä pitäen otettiin käyttöön pugixml C++ -kirjasto. Pugixml mahdollistaa eri XML-rakenteiden erottelemisen toisistaan, mikä helpottaa yksittäisten asioiden vertailua ja sivuutusta.

Tässä luvussa käydään läpi XML-tiedostojen vertailijan eri toiminnot, ja kuinka nämä toiminnot toteutettiin.

4.1 Rivien vertailu string-tyyppisinä muuttujina

Yksinkertaisin ja ilmeisin tapa verrata referenssiriviä ja verrokkiriviä keskenään on niiden vertailu string-muuttujina.

```
if (rRefLine == rNewLine)            // Lines are exactly the same
{
    write_html_row(…);
    continue;
}
```

Kuva 8. Rivien vertailu string-muuttujina

Kuvassa 8 otetaan molempien XML-tiedostojen nykyiset rivit rRefLine ja rNewLine, ja yksinkertaisella ’==’-operaattorilla selvitetään, ovatko kyseiset string-tyyppiset muuttujat täysin samanlaiset. Tällä yksinkertaisella selvityksellä ohitetaan suurin osa osan XML-tiedostojen riveistä. Jos ehtolause käy toteen, eli rivit ovat täysin samanlaiset, kirjoitetaan rivi muistiin visualisointia varten metodilla write_html_row(), ja siirrytään seuraavaan riviin vertailijassa.

4.2 Rivin sisäiset erot

Ensimmäinen kompleksi metodi, jota kehitettiin, pyrkii etsimään rivinsisäiset muunnokset. Tässä metodissa katsotaan, onko tällä hetkellä tarkastelun alla olevissa riveissä eroja elementeissä, elementtien mahdollisissa sisällöissä, tai elementtien attribuutti avain–arvo-pareissa. Jos eroavaisuuksia löydetään, ne merkitään ylös myöhempää katselua varten.

Tämä metodi käynnistyy, jos rivit eivät läpäisseet luvussa **4.1** esiteltyä yksinkertaista vertailijaa.

4.2.1 Ero elementin nimessä

```
if (std::string(pugiRefLine.name()) != std::string(pugiNewLine.name()))
    return;
```

Kuva 9. Elementtien nimien vertailu string-muuttujina.

Metodi lopetetaan kuvan 9 mukaisesti, mikäli elementtien nimet eroavat. Tällöin oletuksena on, että kyseessä on kokonaan eri rivi kuin referenssitiedostossa. Vaikka oletetaan, että rivit ovat vaihtuneet, on olemassa muitakin mahdollisia vaihtoehtoja. On mahdollista, että rivi ei olekaan vaihtanut paikkaa, vaan kyseessä olevan elementin nimi on yksinkertaisesti muuttunut edellisestä. Tämänlaisen eron löytäminen koneellisesti olisi erittäin hankalaa ja vaatisi kirjainmerkkejä loogisesti vertailevan algoritmin. Tämän vuoksi työssä päädyttiin yksinkertaisesti olettamaan, että kirjainerot elementin nimessä viittaavat vaihtuneisiin riveihin.

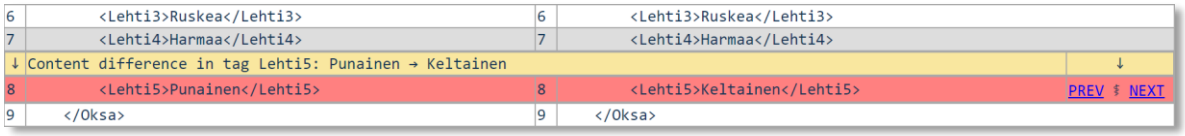
Elementtien nimien erotessa toisistaan palataan yleismetodiin return-lausunnolla. Sisäisten erojen etsiminen lopetetaan, ja ratkaisua etsitään muilla keinoin.

4.2.2 Ero elementin datassa

```
if (std::string(pugiRefLine.text().get()) != std::string(pugiNewLine.text().get()))
{
    const json& json_tag_ignore_list = ignore_list["Tag list"];
    for (const auto& IterTag : json_tag_ignore_list)
    {
       if (IterTag == pugiRefLine.name())
          ignored_element = true;
    }
    if (!ignored_element)
    {
       ssDescription << "Content difference in tag " << std::string(rLine.name())
       << ": " << std::string(rLine.text().get()) << " &rarr; "
       << std::string(nLine.text().get()) << std::endl;
    }
}
```

Kuva 10. Elementtien sisältöjen vertailua.

Kuvassa 10 on toteutettu elementtien sisältöjen vertailu. Elementtien sisällöistä mahdollisesti löytyneet erot listataan ylös ostringstream-muuttujaan, mikä palautetaan myöhemmin string-muuttujana ylempään metodiin. Kyseessä oleva kuvaus tulee esille ainoastaan visualisoinnissa, joka on vapaavalintainen.



Kuva 11. Visualisoinnissa näkyvä elementin datan ero.

Kuvassa 11 visualisoinnin tapa ilmoittaa käyttäjälle poistuneesta rivistä. Eron omaava rivi on merkitty punaisella värillä. Sen yläpuolelta löytyy kommentti virheestä keltaisella taustalla. Mikäli kyseessä oleva elementti löytyy sivuutuslistalta, tämä ero sisällössä sivuutetaan, ja siitä ei merkitä virhettä. Ero kuitenkin näkyy visualisoinnissa hyväksyttävänä virheenä.

4.2.3 Attribuuttien erot

Attribuutteja käytetään laaja-alaisesti eri elementeillä XML-tiedostoissa. Yhdellä elementillä saattaa olla useita eri attribuutteja. Yksikin puuttuva, tai muuttunut tärkeä attribuutti saattaa vaikuttaa lopputulokseen mittavalla tavalla. On siksi erittäin tärkeää pitää huolta myös attribuuttien oikeallisuudesta.

```
if (pugiRefLine.first_attribute() || pugiNewLine.first_attribute())
{
    std::vector<std::pair<std::string, std::string>> RefAttrList;
    std::vector<std::pair<std::string, std::string>> NewAttrList;

    for (pugi::xml_attribute rLineAttr : pugiRefLine.attributes())
         RefAttrList.push_back({ rLineAttr.name(), rLineAttr.value() });

    for (pugi::xml_attribute nLineAttr : pugiNewLine.attributes())
         NewAttrList.push_back({ nLineAttr.name(), nLineAttr.value() });

    check_attributes(RefAttrList, NewAttrList, ignore_list, ssDescription, true, ig-
    nored_element);

    check_attributes(NewAttrList, RefAttrList, ignore_list, ssDescription, false,
    ignored_element);

}
```

Kuva 12. Attribuuttien vertailun aloittaminen.

Kuvassa 12 näkyy, kuinka attribuuttien vertailija ajetaan kahdesti: ensin verrataan verrokki-XML-tiedoston attribuutteja referenssi-XML-tiedoston attribuutteihin, ja sen jälkeen päinvastoin. Näin saadaan täysi selvyys poistuneista sekä uusista attribuuteista. Kun vertailun kohteena on referenssirivin attribuutit, saadaan esille mahdolliset poistuneet attribuutit. Vertailtaessa referenssirivin attribuutteja verrokkirivin attribuutteihin saadaan esille attribuutit, joita ei löytynyt vielä referenssi-XML-tiedostosta.

Attribuuttien paikkojen vaihto huomioidaan myös, vaikka attribuuttien järjestys ei vaikutakaan XML-tiedoston käyttäytymiseen. Attribuuttien paikkojen vaihdon voi huomata visualisoinnin kautta. Eroa vertailun lopputulokseen sillä ei kuitenkaan ole.

Attribuuttien nimien sopiessa yhteen metodi tarkastaa seuraavaksi attribuuttiarvojen yhtäläisyyden.

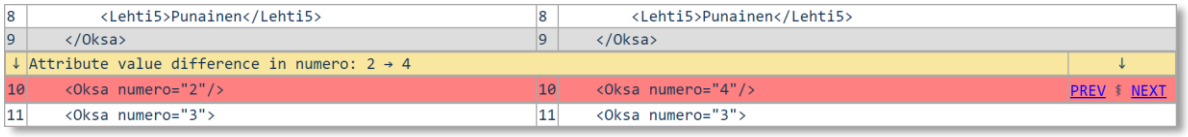
```
const size_t& n_main     = main_list.size();
const size_t& n_comp     = comp_list.size();
for (size_t i_main = 0; i_main < n_main; i_main++)
{
     for (size_t i_comp = 0; i_comp < n_comp; i_comp++)
     {
          ⋮  // Scan for matching attribute. Check attribute value.
          ⋮  // If value differs, mark the difference.
     }
     if (i_comp != n_comp)
          continue;      // Attribute found, move to the next attribute.
     ⋮    // Attribute could not be found. Mark the difference.
}
```

Kuva 13. Attribuuttien vertailuun käytettävä metodi tiivistettynä.

Kuvan 13 koodissa esitellään check_attributes()-metodin for-lauseet, joiden avulla kaikki mahdolliset rivin attribuutit voidaan käsitellä. Ensimmäisessä for-lauseessa iteroidaan pääelementin attribuutteja. Pääelementin seuraaviin attribuutteihin siirrytään vasta sen jälkeen, kun tämänhetkisen attribuutin tilanne on selvitetty. Sen selvitykseen tarvitaan toista for-lausetta, jossa attribuutille etsitään paria toisen elementin attribuuttilistalta. Jos vastaava attribuutti löydetään, attribuuttien arvot vertaillaan keskenään.

Mahdolliset eroavaisuudet listataan, jonka jälkeen siirrytään seuraavaan pääelementin attribuuttiin, ja prosessi toistetaan. Metodi loppuu, kun kaikki attribuutit on selvitetty.

Metodi ajetaan toistamiseen kääntäen, pääattribuuteiksi asetetaan verrokkiattribuutit, ja vertailuattribuuteiksi asetetaan referenssiattribuutit. Rivin sisäisiä eroja etsivä metodi loppu, kun attribuuttierot on selvitetty. Riviltä löytyneet erot merkitään muistiin, jonka jälkeen siirrytään seuraavan rivin prosessoimiseen vertailijassa.



Kuva 14. Attribuuttiero visualisoinnissa.

Kuvassa 14 havainnollistetaan sitä, miten attribuutin arvon muuttuminen näkyy visualisoinnissa. Referenssitiedoston ja verrokkitiedoston riviltä 10 löytyy eroavaisuus elementin attribuutissa ’numero’. Attribuutin arvo on muuttunut luvusta 2 lukuun 4. Rivi näkyy punaisena visualisoinnissa, ja sen ylläolevalla kommenttirivillä kuvaillaan eroavaisuuden syytä.

4.3 Rivien väliset erot

On mahdollista, että eroja XML-tiedostojen välillä tapahtuu laajemmassa mittakaavassa kuin vain rivien sisällä. Tällöin saattaa kyseessä olla rivien paikkojen vaihtuminen, rivien tuhoutuminen, tai uusien rivien syntyminen. Tässä tapauksessa vertailija skannaa vastakkaisen XML-tiedoston elementtiblokin, ja etsii sieltä vastaavaa riviä. Käytännössä kummankaan XML-tiedoston rivit eivät vastaa toisiaan, joten skannaus tapahtuu molemmin päin: referenssitiedosto etsii sen riville oikeaa paria verrokkitiedoston blokista, kun taas verrokkitiedosto etsii oman rivinsä paria referenssitiedoston blokista.

```
size_t i_new_continue = 0; // Searching for rRefLine equivalent in the corresponding
                           // block
for (size_t i_new_scan = 0; i_new + i_new_scan < nRefLines; i_new_scan++)
{
     if (get_indentation(newLines[i_new + i_new_scan]) <= // Stay in range
         get_indentation(newBlocks.back()))
         break;
     if (rRefLine != newLines[i_new + i_new_scan]) // Iterate new file lines until a
         continue;                                 // match was found (same exact
     i_new_continue = i_new_scan;                  // string)
     break;
}
```

Kuva 15. Referenssirivin vastakappaletta etsivä skanneri.

Kuvassa 15 esitetyssä koodissa etsitään paria tämänhetkiselle referenssiriville vastakkaista elementtiblokkia skannaamalla. Jos vastaava rivi löytyy, sen paikka nykyisestä rivistä luetaan i_new_continue-muuttujaan. Vaihtoehtoisesti, jos kyseistä riviä

ei löydetä ollenkaan vastaavalta alueelta, i_new_continue-muuttujan arvoksi jää nolla. Tätä hyödynnetään skannauksien jälkeisessä vaiheessa, jossa rivit merkitään tarvittavalla tavalla.

```
size_t i_ref_continue = 0; // Searching for rNewLine equivalent in the corresponding
                           // block
for (size_t i_ref_scan = 0; i_ref + i_ref_scan < nNewLines; i_ref_scan++)
{
     if (get_indentation(refLines[i_ref + i_ref_scan]) <= // Stay in range
         get_indentation(refBlocks.back()))
         break;
     if (rNewLine != refLines[i_ref + i_ref_scan]) // Iterate ref file lines until a
         continue;                                 // match was found (same exact
     i_ref_continue = i_ref_scan;                  // string)
     break;
}
```

Kuva 16. Verrokkirivin vastakappaletta etsivä skanneri.

Kuvasta 16 löytyy lähes identtinen pätkä koodia edelliseen kuvaan verrattuna. Mainittavana erona on, että tässä kohtaa skannataan referenssitiedoston elementtialuetta, ja sieltä pyritään löytämään samanlaista riviä tämänhetkiselle verrokkiriville.

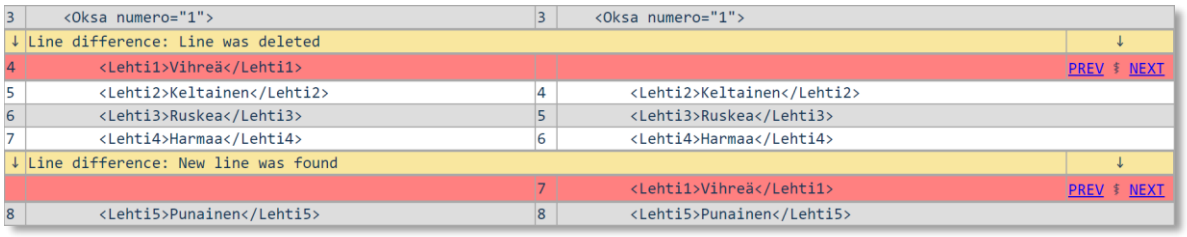
Jos skannausta tehtäisiin vain toiselle riville, ei tiettyjä rivejä huomattaisi. Skannerin etsiessä referenssiriviä vastaavaa riviä verrokkitiedostosta jää ohjelmalta huomaamatta rivit, jotka löytyvät vain verrokkitiedostosta. Skannerin etsiessä verrokkiriviä vastaavaa riviä referenssitiedostosta jää ohjelmalta huomaamatta rivit, jotka löytyvät vain referenssitiedostosta. Skannaamalla molemmin päin saadaan täysi selvyys kummankin XML-tiedoston kaikista riveistä.

```
if (i_ref_continue)
{
    for (size_t i_ref_line = 0; i_ref_line < i_ref_continue; i_ref_line++)
         write_html_row(…);
    i_ref += i_ref_continue - 1;
    i_new--;
}
else if (i_new_continue)
{
    for (size_t i_new_line = 0; i_new_line < i_new_continue; i_new_line++)
         write_html_row(…);
    i_new += i_new_continue - 1;
    i_ref--;
}
else
{
    write_html_row(…);

}
```

Kuva 17. Riviskannauksesta saatujen tuloksien parsimista.

Kuvan 17 koodissa tehdään toimenpiteet skannauksista saaduille tuloksille. Jos verrokkiriviä vastaava rivi löydetään referenssiblokista, päädytään ensimmäiseen if-lauseeseen. Tällöin jokainen referenssiblokkiin kuuluva rivi, mikä edeltää skannauksesta löydettyä verrokkirivin vastinetta, merkitään poistuneeksi riviksi. Tämän jälkeen rivi-iteraattorit tasataan skannauksessa löydetyn vastaavan rivin mukaan. Näiden toimenpiteiden jälkeen vertailu jatkuu normaalin tapaan.



Kuva 18. Rivien siirtyminen visualisoinnissa.

Kuvassa 18 on esimerkki rivien siirtymisestä XML-tiedostojen välillä. Referenssitiedostossa elementin *Lehti1* sisältävä rivi löytyy riviltä 4, kun taas verrokkitiedostossa samainen rivi löytyy riviltä 7. Tämä eroavaisuus prosessoidaan edellä mainituin keinoin. Merkityt rivit näkyvät halutulla tavalla käyttäjälle tehdyssä visualisointitiedostossa.

Mikäli taas referenssiriviä vastaava rivi löydetään verrokkiblokista, saavutaan else if -lauseeseen. Tässä tapauksessa jokainen rivi ennen skannauksessa löydettyä referenssirivin vastakappaletta merkitään uusiksi riveiksi. Tämän jälkeen rivien iteraattorit tasataan skannauksessa löydetyn vastaavan rivin mukaan. Näiden toimenpiteiden jälkeen vertailu jatkuu normaaliin tapaan.

Jos kumpikaan edeltävistä vaihtoehdoista ei toteudu, saavutaan else-lauseeseen. Käytännössä tämä tulee toteen, jos referenssirivillä oleva rivi on poistunut verrokkitiedostosta, ja verrokkitiedostoon on tullut uusi rivi, jota ei löydy referenssitiedostosta. Tällöin ero merkitään muistiin ja siirrytään seuraavalle riville tiedostoissa.

4.4 Elementtiblokit

Termi blokki, tai elementtiblokki, kertoo tässä asiayhteydessä elementtialueesta, joka alkaa emoelementin aloitustagista ja loppuu kyseisen emoelementin lopetustagiin. Blokkiin kuuluu vähintään yksi lapsielementti, mutta siihen voi sisältyä lukematon määrä lapsielementtejä.

```
3     <Oksa numero="1">
4         <Lehti1>Vihreä</Lehti1>
5         <Lehti2>Keltainen</Lehti2>
6         <Lehti3>Ruskea</Lehti3>
7         <Lehti4>Harmaa</Lehti4>
8         <Lehti5>Punainen</Lehti5>
9     </Oksa>
```

Kuva 19. Elementtiblokki havainnollistettuna.

Kuvassa 19 on havainnollistettuna elementtiblokki. Se alkaa riviltä 3 aloitustagilla *<Oksa>*, ja loppuu rivin 9 lopetustagiin *</Oksa>.* Kyseinen blokki sisältää 5 lapsielementtiä. Elementtiblokkien rakennetta käytetään hyödyksi tässä vertailijassa tilanteissa, joissa koko riville tai jopa koko blokille on tapahtunut jotakin tiedostojen

välillä. Näitä toimenpiteitä varten tarvitaan mukana kulkevan tieto siitä, missä blokissa ollaan. Tämän tarpeen täyttävät vektorimuuttujat refBlocks ja newBlocks, sekä niitä päivittävä metodi update_blocks().

4.4.1 Aktiivinen elementtiblokki

```
while (!blocks.empty() && indentCurrentLine <= get_indentation(blocks.back()))
    blocks.pop_back();   // Removing blocks that have closed

if (indentCurrentLine < indentNextLine)
{
    std::string rCurrentLine = erase_ignored_attributes(curr_line, ignore_list);
    blocks.push_back(rCurrentLine);    // Push the line as a new active block
}
```

Kuva 20. Metodin update_block() toimintaperiaate.

Kuvan 20 metodissa vertaillaan nykyisen ja seuraavan rivin, sekä aktiivisen blokin sisennystä. Vektorimuuttujasta tiputetaan säännöllisesti epäaktiiviset blokit pois. Epäaktiivisia blokkeja ovat blokit, joiden sisennys on suurempi kuin tämänhetkisen rivin.

Vektorimuuttujaan lisätään uusi aktiivinen blokki, mikäli tämänhetkinen rivi on emoelementti, eli sillä on lapsielementtejä. Emoelementin tunnistamiseen käytetään sisennystä. Jos seuraavan rivin sisennys on suurempi kuin nykyisen rivin sisennys, on nykyinen rivi emoelementti. Tätä metodia ajetaan aina kun vertailtava rivi muuttuu. Tämän vuoksi käytössä on aina tieto siitä, minkä blokin alaisuudessa tämänhetkinen rivi on.

4.4.2 Eroavaisuudet elementtiblokeissa

On mahdollista, että jokin XML-tiedoston blokeista muuttuu radikaalisti. Se saattaa vaihtaa paikkaa tiedostossa, se saattaa tuhoutua, tai uusia elementtiblokkeja saattaa löytyä.

```
else if (refBlocks.back() != newBlocks.back())
{
    if (refBlocks.back() == rNewLine)             // Contents inside the block
    {                                             // were deleted
        ⋮                 // Mark deleted contents
        continue;
    }
    if (newBlocks.back() == rRefLine)             // Contents inside the block
    {                                             // were added
        ⋮                 // Mark added contents
        continue;
    }
    if (ComparingToRef)  // Scan for ref block in new document
    {
        compare_blocks(…);
        ComparingToRef = false;
        continue;
    }
    if (!ComparingToRef) // Scan for new block in ref document
    {
        compare_blocks(…);
        ComparingToRef = true;
        continue;
    }
}
```

Kuva 21. Blokkieroavaisuuksien eri mahdollisuuksien parsimista.

Kuvan 21 koodissa käydään läpi kaikki mahdollisuudet, jos XML-tiedostojen aktiiviset blokit eivät enää vastaa toisiaan. Jos blokin sisältö on tuhoutunut emoelementistä, ovat referenssiblokki ja verrokkirivi samanarvoisia. Blokin sisällön tuhoutuminen käydään läpi ensimmäisessä if-lauseessa, jossa kaikki blokin rivit merkitään poistuneiksi.

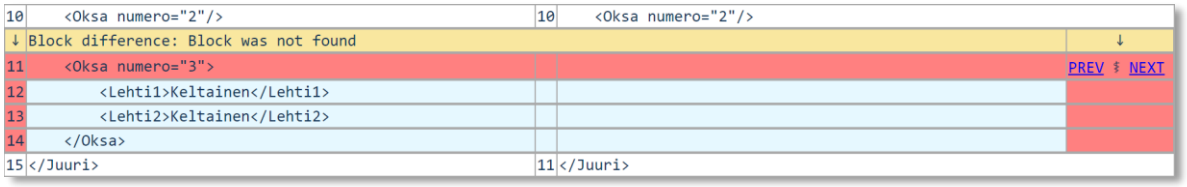
Jos referenssitiedoston elementtiin on syntynyt lapsielementtejä verrokkitiedostossa, ovat verrokkiblokki ja referenssirivi samanarvoisia. Uudet rivit käydään läpi toisessa if-lauseessa, jossa kaikki blokin rivit merkitään uusiksi riveiksi.

Kolmanteen tai neljänteen if-lauseisiin siirrytään, mikäli edelliset if-lauseet eivät toteutuneet. Näissä tapauksissa itse koko blokki on siirtynyt, poistunut, tai uusi blokki on syntynyt. If-lauseiden erona on se, kumman tiedoston blokkiin blokkiskannerissa verrataan.

Blokkiskannerin toimintaperiaate on olemukseltaan samantapainen kuin luvussa **4.3** esitelty riviskanneri. Erona riviskanneriin onkin blokkiskannerin suuruus ja kompleksisuus. Siinä missä riviskannerissa etsitään yhdelle riville vastakappaletta, blokkiskannerissa etsitään koko blokille paria.

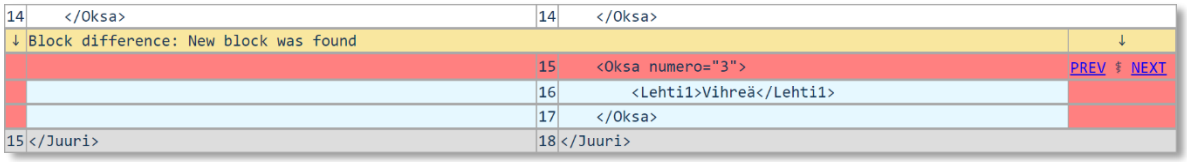
Metodi compare_blocks() etsii puuttuvaa blokin aloittavaa riviä alueelta, jossa se sijaitsee päätiedostossa. Metodista löytyvä blokkiskanneri rullaa koko alueen läpi, ja vertailee jokaista riviä yksi kerrallaan pääriviin. Jos puuttuva rivi löydetään, löydetään myös blokki, jonka rivi aloittaa.

Jos blokki löydetään, sen siirtyminen merkitään muistiin. Tämän jälkeen alkaa blokkiin kuuluvien elementtien vertailu. Rivit vertaillaan yksi kerrallaan hyödyntäen kaikkia käytettävissä olevia vertailumetodeja. Niistä löydettävät eroavaisuudet merkitään muistiin. Blokin loppuessa rivien iteraattorit tasataan skannauksessa löydetyn blokin mukaan. Vertailija palaa takaisin päämetodiin, jossa normaali vertailu jatkuu seuraavasta rivistä lähtien.



Kuva 22. Blokin puuttuminen visualisoinnissa.

Jos referenssiblokin vastinetta ei löydy verrokkitiedostosta blokkiskannauksen seurauksena, merkitään blokki poistuneeksi. Kyseinen tilanne on kuviteltuna kuvassa 22. Myös kaikki blokin rivit merkitään poistuneiksi. Rivien iteraattorit tasataan, ja vertailija palaa takaisin päämetodiin. Vertailu palautuu normaaliksi päämetodissa.



Kuva 23. Uuden blokin löytyminen visualisoinnissa.

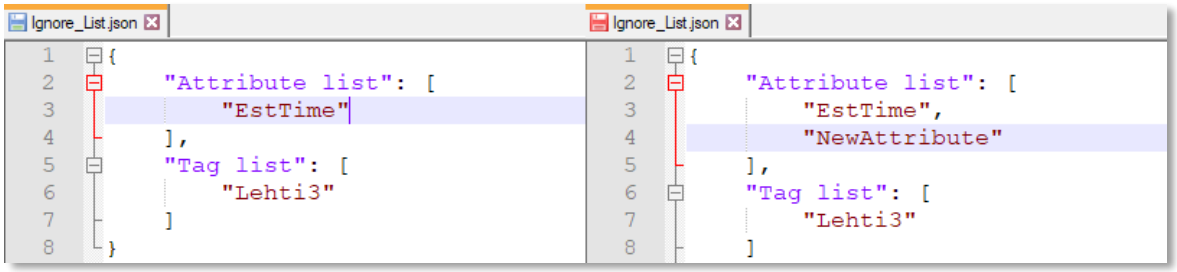
Jos verrokkiblokin vastinetta ei löydy referenssitiedostosta blokkiskannauksen seurauksena, merkitään kyseinen blokki uudeksi, syntyneeksi blokiksi. Tämä tilanne

näkyy kuvan 23 tavalla visualisointitiedostossa. Uuden blokin rivit merkitään uusiksi riveiksi. Toimenpiteiden jälkeen rivien iteraattorit tasataan, ja vertailija palaa takaisin päämetodiin, jossa vertailu palautuu normaaliksi.

4.5 Elementtien ja attribuuttien sivuuttaminen

Läpikäytävät XML-tiedostot sisältävät monia elementtejä ja attribuutteja, joilla ei ole olennaista merkitystä vertailussa. Tämänlaiset elementit ja attribuutit liittyvät muun muassa aikalaskelmiin. Jotta vältytään vääriltä eroavaisuuksilta, jouduttiin eri metodeihin sisällyttämään tapa sivuuttaa nämä tietyt epäolennaiset elementit ja attribuutit.

Työssä päätyttiin luomaan JSON-tyyppinen tiedosto, joka asetetaan sopivaan paikkaan koontiversion tiedostopolkuun. Tiedostoa voidaan muokata helposti tekstieditorilla. JSON-tyyppistä tiedostoa voidaan lukea helposti ohjelman sisällä Niels Lohmannin rakentamalla ja ylläpitämällä JSON for Modern C++ -kirjastolla. Sivuutuslistan tiedostonimi on Ignore_List.json.

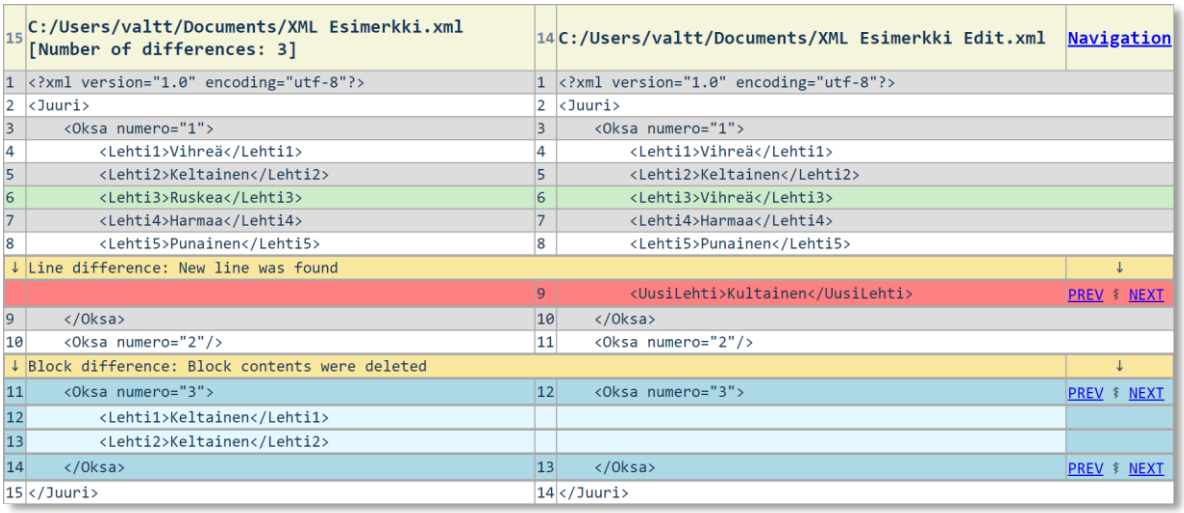


Kuva 24. Sivuutettavan attribuutin lisääminen sivuutuslistalle.

Kuvassa 24 esitellään attribuuttien ja elementtien sivuutuslistaa. Sivuutuslistalla attribuutit ja tagit on eroteltu kategorioiksi. Attribuuttilistaan ja tagilistaan voi helposti lisätä uusia sivuutettavia osia tarpeen mukaan. Kuvassa 17 listalle lisätään uusi sivuutettava attribuutti, jonka nimi on *NewAttribute*. Se lisätään oikeaoppisesti lisäämällä viimeisimpään attribuuttiin pilkku ja rivinvaihto. Tämän jälkeen uusi sivuutettava attribuutti kirjoitetaan lainausmerkkien sisään. Tiedosto tallennetaan, jonka jälkeen lisättyä attribuuttia ei enää huomioida vertailijaa ajettaessa.

4.6 Verrattavien tiedostojen visualisointi

XML-tiedostojen vertailijaan rakennettiin tiedostoja ja niiden eroavaisuuksia hahmottava visualisointitiedosto. Visualisointitiedosto on HTML-tyyppinen tiedosto, joka esittää käyttäjälle molemmat XML-tiedostot vierekkäin niiden eroavaisuudet riveihin integroituna. Visualisointitiedostosta löytyy rivinumeroinnit, XML-tiedostojen nimet ja kansiopolut sekä kuvaus jokaisesta vertailusta löydetystä eroavaisuudesta. Näiden lisäksi visualisoinnista löytyy myös navigointipalkki, jonka avulla voi helposti siirtyä eroavaisuudesta toiseen.



15	C:/Users/valtt/Documents/XML Esimerkki.xml [Number of differences: 3]	14	C:/Users/valtt/Documents/XML Esimerkki Edit.xml	Navigation
1	<?xml version="1.0" encoding="utf-8"?>	1	<?xml version="1.0" encoding="utf-8"?>	
2	<Juuri>	2	<Juuri>	
3	<Oksa numero="1">	3	<Oksa numero="1">	
4	<Lehti1>Vihreä</Lehti1>	4	<Lehti1>Vihreä</Lehti1>	
5	<Lehti2>Keltainen</Lehti2>	5	<Lehti2>Keltainen</Lehti2>	
6	<Lehti3>Ruskea</Lehti3>	6	<Lehti3>Vihreä</Lehti3>	
7	<Lehti4>Harmaa</Lehti4>	7	<Lehti4>Harmaa</Lehti4>	
8	<Lehti5>Punainen</Lehti5>	8	<Lehti5>Punainen</Lehti5>	
↓	Line difference: New line was found			↓
		9	<UusiLehti>Kultainen</UusiLehti>	PREV ǂ NEXT
9	</Oksa>	10	</Oksa>	
10	<Oksa numero="2"/>	11	<Oksa numero="2"/>	
↓	Block difference: Block contents were deleted			↓
11	<Oksa numero="3">	12	<Oksa numero="3">	PREV ǂ NEXT
12	<Lehti1>Keltainen</Lehti1>			
13	<Lehti2>Keltainen</Lehti2>			
14	</Oksa>	13	</Oksa>	PREV ǂ NEXT
15	</Juuri>	14	</Juuri>	

Kuva 25. Visualisointitiedoston yleiskuva.

Kuvassa 25 on esimerkki visualisoinnista. Eroavaisuuksia on löydetty kolme kappaletta: yksi uusi rivi on löytynyt, ja kaksi riviä on kadonnut blokin sisältä. Näiden lisäksi sivuutetun elementin *Lehti3* sisältö on muuttunut. Sivuutetun elementin eroa ei merkitä eroksi, mutta se näkyy vihreällä värillä visualisoinnissa.

Visualisointitiedosto luotiin samassa ohjelmointitiedostossa vertailijan kanssa. Se luodaan create_html_report()-metodissa. Sen luomiseen käytetään listamuuttujaan lHTML_table_rows tallennettua tietoa riveistä ja niistä löytyvistä erovaisuuksista. Listamuuttujaan lHTML_table_rows on jokaisen rivin jälkeen viety tieto referenssirivistä, verrokkirivistä sekä niiden rivinumeroista. Listaan lHTML_table_rows on myös viety tieto rivin tyypistä, sekä kuvaus, mikäli riveiltä löydettiin eroavaisuuksia.

Tiedoston luominen aloitetaan C++-standardikirjaston muuttujan ofstream luonnilla. Ofstream-muuttujaan voidaan helposti lukea dataa käyttämällä Input funktiota ’<<’. Ofstream-muuttujaan luetaan tarvittavat HTML-merkinnät kuten rivit ja sarakkeet, sekä listamuuttujasta lHTML_table_rows löytyvät XML-tiedostojen vertailijan tiedot. Tietoja hyödyntäen lopputuloksena on haluttuun kansioon muodostunut HTML-tyyppinen visualisointitiedosto, josta käyttäjä voi helposti nähdä erot XML-tiedostojen välillä.

5 Yhteenveto

Opinnäytetyössä käytiin läpi metakieltä Extensible Markup Language, ja selitettiin siihen liittyvää termistöä sekä sen alkuperää. XML-metakielen eri piirteet ja rakenne huomioon ottaen työssä kerrottiin XML-tiedostojen vertailijan kehittämisestä. Työlle tehtiin ensin vaatimusmäärittely, jossa luetellaan halutut ominaisuudet. Vaatimusmäärittelyn jälkeen toteutuksen esittäminen aloitettiin.

Luvun **3** vaatimuksia silmällä pitäen työssä onnistuttiin hyvin. Työssä haluttiin, että työn XML-tiedostojen vertailija säilyttää tiedostojen alkuperäisformaatin ja rivinumerot. Työhön pyydettiin myös mahdollisuutta sivuuttaa tiettyjä elementtejä ja attribuutteja. Jatkokehittävyyttä pidettiin tärkeänä, kuten myös ominaisuutta nähdä testeistä löydetyt eroavaisuudet yksinkertaisessa muodossa.

Kaikki vaatimukset toteutettiin vähintään hyväksyttävästi. XML-tiedostojen vertailija säilyttää tiedostojen alkuperäisformaatin, sekä antaa kullekin riville oman numeron. Elementtien ja attribuuttien sivuutus onnistuu sivuutuslistatiedostoa käyttäen. Jatkokehittävyys on mahdollista, ja suurin osa vertailumetodeista on selkeitä ja helposti ymmärrettäviä. Joitakin vertailijan osia olisi kuitenkin hyvä selkeyttää vielä entisestään monimutkaisuuden vähentämiseksi. Eroavaisuuksien visuaaliseen esittämiseen panostettiin työssä, ja lopputuloksena saatiinkin selkeä ja nopeasti luettava kevyt visualisointitiedosto.

5.1 Pohdinta

XML-tiedostojen vertailijan kehittäminen oli tekijälle yksi ensimmäisistä laajoista ohjelmointiprojekteista. Työn aloittaessa ei yrityksessä ollut täysin selvää, kuinka tiedostojen vertailijaa ylipäätänsä kuuluisi lähteä kehittämään. Vertailijaa rakennettiinkin yksi osa kerrallaan, ja sitä tehdessä huomattiin paremmin mihin suuntaan tiedostojen vertailijaa tulisi kehittää.

Tekijällä oli vain vähän kokemusta projektissa käytetystä ohjelmointikielestä C++. Työssä käytettiinkin aikaa uuden opetteluun ja asioihin perehtymiseen.

LÄHTEET

Historia. Ei päiväystä. Prima Power. [Verkkosivu]. [Viitattu 23.3.2020]. Saatavana: https://www.primapower.com/fi/historia/

Työpaikat. Ei päiväystä. Prima Power. [Verkkosivu]. [Viitattu 23.3.2020]. Saatavana: https://www.primapower.com/fi/avoimet-tyopaikat/

Yritys. Ei päiväystä. Prima Power. [Verkkosivu]. [Viitattu 23.3.2020]. Saatavana: https://www.primapower.com/fi/yritys/

Fennell, P. 2013. Extremes of XML. XML London. [Viitattu 13.4.2020]. Saatavana: https://xmllondon.com/2013/xmllondon-2013-proceedings.pdf

Pugixml. Ei päiväystä. Pugixml. [Verkkosivu]. [Viitattu 23.3.2020]. Saatavana: https://pugixml.org/

Extensible Markup Language (XML). 26.11.2008. W3C. [Verkkosivu]. [Viitattu 13.4.2020]. Saatavana: https://www.w3.org/TR/xml/

XML. Ei päiväystä. 2K mediat. [Verkkosivu]. [Viitattu 13.4.2020]. Saatavana: https://www.2kmediat.com/xml/syntaksi-2.asp

JSON for Modern C++. Ei päiväystä. GitHub. [Verkkosivu]. [Viitattu 13.4.2020]. Saatavana: https://github.com/nlohmann/json

Myer, T. 2006. No Nonsense XML Web Development With PHP. Revised edition. Melbourne: SitePoint.

Stroustrup, B. 1985. The C++ Programming Language. 4th Edition. Boston: Addison-Wesley Professional.