XML-JÄSENTIMIEN VERTAILU

Kari Kuvaja

29.4.2008
Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu –tutkielma

Tiivistelmä

Tämä pro gradu-tutkielma käsittelee ohjelmoijan näkökulmasta *XML*-tiedon käsittelemistä XML-jäsentimillä. Tutkielmassa selvitetään, onko eri XML-jäsentimien välillä tehokkuudessa eroja käsiteltäessä erikokoisia XML-tiedostoja. Tutkittavat jäsentimet perustavat toimintansa *SAX (*Simple API for XML), *DOM* (Document Object Model) tai *JAXP* (Java API for XML Processing)-rajapintoihin. Mittausvälineenä on Javalla toteutettu ohjelma, joka laskee XML-tiedoston jäsentämiseen kulutetun ajan, sekä mittaa käytettyä muistin määrää. Lähdemateriaali on generoitu ohjelmallisesti käyttäen Python-kielistä skripitiä. Tutkimuksessa käytettävä materiaali koostuu erikokoisista XML-dokumenteista, jotka kaikki sisältävät sisäisen DTD:n. Tutkittavalle materiaalille suoritetaan testeissä ajot ilman validointia, sekä validoinnin kanssa niillä jäsentimillä, jotka kykenevät validointiin.

Tutkimuksessa saatujen tulosten perusteella ominaisuuksiltaan monipuolisin ja tasaisen hyvä XML-jäsennin on Xerces. Piccolo on nopein SAX-jäsennin kaiken kokoisilla tiedostoilla. Ominaisuuksiltaan suppea Crimson osoittautui nopeaksi jäsentimeksi riippumatta käytettävästä rajapinnasta ja riippumatta siitä, validoidaanko dokumentti. SAX-rajapinnan toteuttava Aelfred-jäsennin osoittautui todelliseksi muistinkuluttajaksi vieden huomattavan määrän enemmän muistia kuin muut jäsentimet isoja dokumentteja käsitellessä. GNU Jaxp-pakettiin kuuluva Aelfred2-jäsennin oli tasaisen hyvä jokaisessa testissä.

ACM-luokat (ACM Computing Classification System, 1998 version): C.4, D.3.4, D.1.5
Avainsanat: SAX, DOM, XML-jäsentimet, suorituskyky

SISÄLLYSLUETTELO

1. Johdanto

XML (Extensible Markup Language) on nykypäivänä hyvin yleisesti käytetty teknologia, joka soveltuu monenlaisiin tarkoituksiin. Käyttöjärjestelmäriippumattomuus sekä laitteistoriippumattomuus tekevät siitä erittäin toimivan tavan siirtää tietoa.

XML-tiedosto kertoo jo aika paljon sisällöstään. XML-tiedostot ovat yleensä tekstimuodossa, joten sen sisältöä voimme silmäillä ja hakea editorilla, ja mahdollisesti tämä jo riittää joihinkin tarkoituksiin. Jos rupeamme käyttämään XML:ää monimutkaisempiin tarkoituksiin, tulemme todennäköisesti tarvitsemaan tavan, jolla saamme haettua tietoa nopeammin.

Tähän tarkoitukseen tarvitsemme XML-jäsentimiä. Jäsentimet toimivat työkaluna dokumentin käsittelyssä ilman, että meidän tarvitsee huolehtia dokumentin sisäisistä merkkauksista. XML-jäsennin löytyy esimerkiksi useista Internet-selaimista, kuten Firefox ja Internet Explorer. Yleisesti käytössä olevat XML-jäsentimet tukevat yhtä tai useampaa XML-rajapintaa. Rajapinta määrittelee toimintoja, joita jäsentimet toteuttavat. Se, miten paljon ominaisuuksia jäsennin tukee, vaihtelee eri jäsentimien välillä. XML-jäsentimillä on eroja paitsi rajapinnan suhteen, myös toteutuksessa. Jotkut jäsentimet käyttävät enemmän muistia XML-olion rakenteen tallentamiseen muistiin ja samoin myös dokumentin jäsentämisnopeudessa löytyy eroja eri jäsentimien välillä. Muita eroja ovat muun muassa käyttömukavuus, virheettömyys sekä lisenssi.

Välttääkseen vaikeuksia XML-dokumentin läpikäynnissä ja käsittelyssä, lähes kaikki ohjelmoijat käyttävät tehtävän suorittamiseen XML-jäsentimiä. Jäsennin itsessään on joko kirjasto tai luokka, kuten esimerkiksi Javassa. Jäsennin lukee XML-dokumenttia ja tarkistaa, että se on oikein muodostettu. Asiakasohjelmistot käyttävät metodikutsuja, jotka on määritelty jäsentimen rajapinnassa, hakemaan tietoa, jonka jäsennin palauttaa XML-dokumentista [1].

XML-dokumenttien käsittelyyn kaksi käytetyintä rajapintaa ovat SAX sekä DOM. Molemmista niistä on olemassa useita versioita. Lisäksi on olemassa useita muita rajapintoja, kuten JAXP, JDOM, domj4, ElectricXML, XMLpull ja Data Binding [1].

Tutkielman luvussa 2 käydään läpi johdanto XML:n perusteisiin. Luvun alussa on lyhyt johdatus XML:ään ja myöhemmin käsitellään oikein muodostetut ja validit XML-dokumentit. Luvussa käydään läpi myös XML:n syntaksi. Luvussa 3 perehdytään kahteen suurimpaan XML-rajapintaan, SAX ja DOM. Luvussa 4 tarkastellaan jäsentimen valintaan vaikuttavia tekijöitä. Luvussa 5 suoritetaan jäsentimen tehokkuuden mittaus ja luku 6 on yhteenveto tutkimuksesta.

2. XML:n perusteet

Jo painokoneen keksimisestä lähtien kirjoittajat ovat tehneet merkintöjä dokumentteihinsa, jotka ohjeistivat painajia muotoilussa. Noita muistiinpanoja kutsuttiin merkkauksiksi. Oikolukijat käyttivät käsin kirjoitettua symbolista merkkauskieltä, joilla ilmaistaan virheet editoijille ja painajille. Jopa erotinmerkkien kuten piste, pilkku jne. käyttö on itse asiassa tapa ilmaista lukijalle, miten tekstiä pitäisi tulkita [5].

2.1. XML vs. HTML

XML on suhteellisen uusi merkkauskieli. Se juontaa juurensa aikaisemmasta merkkauskielestä, *SGML*:stä (Standard Generalization Markup Language). Myös *HTML* (Hypertext Markup Language) on sovellettu SGML:stä. On myös olemassa uusi versio HTML 4:stä, jota kutsutaan *XHTML*:ksi (Extensible Hypertext Markup Language), joka on sovellus XML:stä. Kaikki nämä merkkauskielet soveltuvat metatiedon kirjoittamiseen, mutta SGML:ää ja XML:ää voidaan käyttää myös tällaisten merkkauskielien määrittelemiseen [5]. HTML- ja XML-esimerkkilistaukset samasta sivusta selventävät kielten välistä eroa.

```
<HTML>
<HEAD>
   <TITLE>Product Catalog (Toysco-only)</TITLE>
</HEAD>
<BODY>
   <H1>Product Catalog (Internal-use only!)</H1>
   <H2>Product Descriptions</H2>
   <H3>Mega Wonder Widget</H3>
   <P>The <EM>Mega Wonder Widget</EM> is a popular toy with a 20 oz.
capacity. It costs only $12.95 to make, whilst selling for $33.99 (plus
$3.95 S&H).<BR>
   <H3>Giga Wonder Widget</H3>
   <P>The <EM>Giga Wonder Widget</EM>is even more popular, because of its
larger 55 oz. capacity. It has a similar profit margin (costs $19.95,
sells for $49.99).
   <P><I>Updated:</I> 2001-04-01 <I>by Webmaster Will</I>
</BODY>
</HTML>
```

Kuva 1. HTML-dokumentti [5].

Kuvan 1 esimerkkidokumentti käyttää muutamaa rakenteellista merkkausta, kuten <TITLE>, <H1>, <H2>, <H3> otsikoihin ja <P> kappaleenvaihtoihin. Tämä rakenne rajoittuu esitysmalliin dokumentin ulkonäöstä [5].

XML sallii luoda rakenteellisen mallin DTD:n avulla tiedosta kuvan 2 mukaisesti.

```
<?xml version="1.0" ?>
<!DOCTYPE ProductCatalog
[
<!ELEMENT ProductCatalog  (HEAD?, BODY?) >
<!ELEMENT HEAD  (TITLE, Updated, Author+, Security*) >
<!ELEMENT BODY  (H1, H2, (H3, Products)+ ) >
<!ELEMENT Products  (Product+) >
<!ELEMENT Product  (#PCDATA|Prodname|Capacity|Cost|Price|Shipfee)* >

<!ELEMENT H1    (#PCDATA) >
<!ELEMENT H2    (#PCDATA) >
<!ELEMENT H3    (#PCDATA) >
<!ELEMENT TITLE (#PCDATA) >
<!ELEMENT Updated  (#PCDATA) >
<!ELEMENT Author   (#PCDATA) >
<!ELEMENT Security (#PCDATA) >
<!ELEMENT Prodname (#PCDATA) >
<!ELEMENT Capacity (#PCDATA) >
<!ELEMENT Cost     (#PCDATA) >
<!ELEMENT Price    (#PCDATA) >
<!ELEMENT Shipfee  (#PCDATA) >
<!ENTITY MWW "Mega Wonder Widget" >
<!ENTITY GWW "Giga Wonder Widget" >
]>
```

Kuva 2. DTD – määrittely [5].

Kuvassa 3 esitely kuvan XML-dokumentti näyttää samalle, kuin vastaava HTML-versio kuvassa 2, mutta antaa vapauden käyttää itse määriteltyjä merkkauksia, kuten <Prodname> ja <Weight>. Tätä ei voi tehdä HTML-kielellä, koska sen merkkaus on ennalta määritelty, muuttuen hitaasti sitä mukaa, kun selaimet tukevat uusia ominaisuuksia [5].

```
<ProductCatalog>
   <HEAD>
      <TITLE>Product Catalog</TITLE>
      <Updated>2001-04-01</Updated>
      <Author>Webmaster Will</Author>
      <Security>Toysco-only (TRADE SECRET)</Security>
   </HEAD>
   <BODY>
      <H1>Product Catalog</H1>
      <H2>Product Descriptions</H2>
      <Products>
         <H3>&MWW;</H3>
         <Product>
            The <Prodname>&MWW;</Prodname> is a popular toy with a
            <Capacity unit="oz.">20</Capacity> capacity. It costs only
            <Cost currency="USD">12.95</Cost> to make, whilst selling for
            <Price currency="USD">33.99</Price> (plus
           <Shipfee currency="USD">3.95</Shipfee> S&amp;H).<BR/>
         </Product>
         <H3>&GWW;</H3>
         <Product>
            The <Prodname>&GWW;</Prodname> is popular, because of its
            larger <Capacity unit="oz.">55</Capacity> capacity. It has a
            similar profit margin (costs <Cost currency="USD">19.95</Cost>,
            sells for <Price currency="USD">33.99</Price>).<BR/>
         </Product>
      </Products>
   </BODY>
</ProductCatalog>
```

Kuva 3. XML-dokumentti [5].

2.2. XML:n tavoitteet ja historia

Vuonna 1996 *W3C* (World Wide Web Consortium) alkoi suunnitella merkkauskieltä, joka yhdistäisi SGML:n joustavuuden, sekä laajalle levinneen HTML:n suosion. Kielen nimeksi tuli XML [5]. W3C kehitti 10 suunnittelutavoitetta XML:lle, jotka olivat:

1. XML:n tulee olla helposti käytettävissä internetissä.
2. XML tulee tukemaan laajasti erilaisia sovelluksia.
3. XML tulee olemaan yhteensopiva SGML:n kanssa
4. Tulee olemaan helppoa kirjoittaa ohjelmia, jotka käsittelevät XML-dokumentteja.
5. Vaihtoehtoisten ominaisuuksien määrä XML:ssä tulee pitää minimissään, ideaalisesti nollassa.
6. XML-dokumenttien pitäisi olla ihmisten ymmärrettävissä ja suhteellisen selviä.
7. XML-määrittely pitäisi valmistella nopeasti.

8. XML:n määrittely tulee olemaan formaalia ja tehokkaasti ilmaisevaa.
9. XML-dokumentteja tulee olemaan helppo luoda
10. XML-merkkauksen pituudella on vain minimaalinen merkitys.

XML:n 1.0 versio määriteltiin helmikuussa 1998. Lukuisat pienet virheet dokumentoinnissa ja muutokset standardeissa johtivat lokakuussa 2000 XML 1.0:n toisen painoksen (Second Edition) tulemiseen, joka korjasi ja päivitti dokumentaatiota muuttamatta itse XML:ää [5].

Helmikuussa 2004 ilmestyi kolmas painos XML:stä. Se sisältää korjauksia XML 1.0:n määrittelystä löytyneisiin virheisiin, ja nämä virhekorjaukset sisällytettiin kolmanteen painokseen. Lisäksi on selvennetty, milloin avainsanoja kuten MUST, SHOULD ja MAY käytetään muodollisessa merkityksessä [4].

Helmikuussa 2004 valmistui myös XML 1.1. Unicode-standardi, mihin XML 1.0 pohjautuu merkkien määrittelemisessä, ei ole säilynyt muuttumattomana, kehittyen versiosta 2.0 versioon 4.0 jne. Merkkejä, joita ei esitelty Unicode 2.0:ssa voidaan käyttää XML 1.0:ssa. Kuitenkaan niitä ei sallita elementin tyypin nimessä, attribuuttien nimissä, prosessointiohjeissa jne. Lisäksi joitakin merkkejä, joita olisi pitänyt sallia XML-nimissä ei ollut Unicode 2.0:ssa [4].

Yleinen filosofia nimille XML:ssä on muuttunut XML 1.0:n ajoista. XML 1.0 antoi kiinteän määrittelyn nimille, kaikki joka ei ollut sallittua, oli kiellettyä. XML 1.1:n nimet on suunniteltu niin, että kaikki, mikä ei ole kiellettyä, on sallittua. Koska Unicode tulee jatkamaan kehittymistään version 4.0 jälkeen, myöhemmät muutokset XML:ään voidaan välttää sallimalla mitkä tahansa merkit nimissä [4].

XML 1.1 määrittelee lisäksi joukon rajoitteita, joita kutsutaan ”täydeksi normalisaatioksi” XML-dokumenteissa, joista dokumenttien tekijöiden pitäisi huolehtia, ja dokumentin jäsentimien pitäisi varmentaa. Käyttämällä täysin normalisoituja dokumentteja varmistetaan, että attribuuttien arvojen, nimien ja merkkisisällön tunnistaminen voidaan tehdä oikein vertaamalla Unicode-merkkijonoja binäärisesti [4].

Elokuussa 2006 ilmestyi XML 1.1:stä toinen painos, jossa dokumentaatiota korjattiin [4].

2.3. Oikein muodostettu XML-dokumentti

Dokumentin oikea muoto on XML-dokumentin minimivaatimus. Jos dokumentti ei ole oikein muodostettu, niin jäsentimet eivät pysty käsittelemään sitä. Jäsentimillä ei myöskään ole valtuuksia korjata väärin muodostettua dokumenttia. Kun jäsennin törmää väärin muodostettuun dokumenttiin, niin se raportoi virheestä ja lopettaa jäsentämisen [1]. Oikein muodostettu XML-dokumentti sisältää yhden tai useamman elementin, joka on ympäröity alku- ja loppumerkinnöillä. On oltava olemassa ainakin yksi elementti, dokumenttielementti, joka sisältää kaikki muut elementit dokumentin sisällä. Kaikki elementit muodostavat yksinkertaisen, hierarkkisen puun siten, että ainut elementistä elementtiin -suhde on vanhemmasta lapselle -suhde [3].

Oikein muodostettujen XML-dokumenttien olemassaolo sallii XML-datan käytön ilman, että tarvitsee rakentaa ulkoista kuvausta datasta, tai viitata siihen.

2.4.Validi XML-dokumentti

Mikä tahansa XML-dataolio on validi XML-dokumentti, jos se on oikein muodostettu, noudattaa validisuusrajoituksia ja vastaa kielioppia, joka kuvaa dokumentin sisältöä. Kuten SGML, XML voi tarjota kuvauksen dokumentin rakenteesta ja sen kieliopin dokumentin tyypin määrittelyn (Document Type Definition, DTD) tai skeeman avulla [3].

2.4.1. Dokumentin tyypin määrittely

DTD määrittelee XML-dokumentin laillisen rakenteen. Se määrittelee dokumentin rakenteen listalla laillisia elementtejä ja attribuutteja. DTD voidaan määritellä dokumentin sisällä tai ulkoisena viittauksena [6].

Jos DTD-määritys on dokumentin sisällä, dokumentin pitää sisältää DOCTYPE-määrittely:

```
<!DOCTYPE root-element [element-declarations]>
```

Kuvassa 4 on esitetty dokumentti, jolla on sisäinen DTD.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Kuva 4. Dokumentti, jolla on sisäinen DTD [6].

Kuvan 4 DTD tulkitaan seuraavasti [6]:

- !DOCTYPE note määrittelee, että dokumentin juurielementti on note.
- !ELEMENT note määrittelee, että note-elementti sisältää neljä elementtiä: to, from, heading, ja body.
- !ELEMENT to määrittelee to-elementin tyypiksi #PCDATA
- !ELEMENT from määrittelee from-elementin tyypiksi #PDCATA
- !ELEMENT heading määrittelee heading-elementin tyypiksi #PCDATA
- !ELEMENT body määrittelee body-elementin tyypiksi #PCDATA

Jos DTD on määritelty ulkoiseen tiedostoon, dokumentin pitää sisältää DOCTYPE-määritys seuraavan kieliopin mukaisesti [6]:

```
<!DOCTYPE root-element SYSTEM "filename">
```

Kuvan 4 dokumentti voidaan esittää myös ulkoisella DTD:llä tai skeemalla kuvien 6 ja 7 mukaisesti.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Kuva 5. XML-dokumentti, jossa määritellään ulkoinen DTD [6].

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Kuva 6. tiedosto note.dtd joka sisältää DTD:n [6].

2.4.2. XML-skeema

Dokumentti voidaan validoida myös W3C:n XML Schema-kielen avulla. W3C:n XML Schema-kieli on vähemmän rajoittunut, kuin DTD. Skeemoja kirjoitetaan XML-dokumentin ilmentymän syntaksilla käyttämällä attribuutteja, elementtejä ja tunnisteita. Skeemat ymmärtävät myös nimiavaruuksia. Lisäksi skeemat voivat määrittää tietotyyppejä, kuten kokonaisluku tai päivämäärä elementeille, ja validoida dokumentin, ei vain elementtirakenteen perusteella, vaan myös elementtien sisällön perusteella [1]. Kuvassa 7 on esitetty esimerkki XML-skeemasta ja kuvassa 8 sitä vastaava XML-dokumentti. Ennen dokumentin validointia tulee määritellä XML-skeeman sijainti. XML-skeeman sijainti voidaan määritellä xsi:schemaLocation-attribuutin avulla, jos skeema tarvitsee nimiavaruuksia. Skeema, joka ei tarvitse nimiavaruuksia juurielementissä, tai missään muissa dokumentin elementeissä, voidaan määritellä attribuutin xsi:noNamespaceSchemaLocation avulla [7].

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="catalog">
<xs:complexType>
<xs:sequence>
<xs:element ref="magazine" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="title" type="xs:string"/>
<xs:attribute name="publisher"  type="xs:string"/>
</xs:complexType>
</xs:element>

<xs:element name="magazine">
<xs:complexType>
<xs:sequence>
<xs:element ref="article" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="date" type="xs:string"/>
</xs:complexType>
</xs:element>

<xs:element name="article">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="xs:string"/>
<xs:element ref="author" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="author">
<xs:complexType>
<xs:sequence>
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

Kuva 7. OracleCatalog.xsd-esimerkkiskeema [7].

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="OracleCatalog.xsd"
        title="Oracle Magazine" publisher="Oracle Publishing">
  <magazine date="November-December 2003">
  <article>
    <title>Updating XQuery</title>
    <author>
     <firstname>Jason</firstname>
     <lastname>Hunter</lastname>
     </author>
   </article>
  <article>
    <title>Servlets and JSP Step Up</title>
    <author>
     <firstname>Budi</firstname>
     <lastname>Kurniawan</lastname>
     </author>
   </article>
  </magazine>
</catalog>
```

Kuva 8. XML-dokumentti, joka sisältää Oracle Magazine-katalogin [7].

Jos skeeman sijainti määritellään XML-dokumentissa, kuvan 8 <catalog>-elementti määri–teltäisiin seuraavasti [7]:

```
<catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="file://c:/Schemas/OracleCatalog.xsd"
...>
```

XML-skeema, esimerkiksi kuvan 7 OracleCatalog.xsd, määrittelee XML-dokumentin rakenteen. Elementtien määritykset XML-skeemassa voivat määritellä elementtien nimiavaruudet XML-dokumentissa. Nimiavaruus määritellään merkinnällä xmlns:etuliite=<nimiavaruuden url>. Etuliite on nimiavaruuden etuliite. Oletusnimiavaruus määritellään seuraavasti : xmlns = <nimiavaruuden url>. Elementti xs:schema esimerkkiskeemassa OracleCatalog.xsd on määritelty nimiavaruuden määrityksellä seuraavanlaisesti [7]:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Tällöin elementit OracleCatalog.xsd-skeemassa kuuluvat nimiavaruuteen http://www.w3.org/2001/XMLSchema [7].

2.4.3. Poikkeukset ja virheet

XML-määritys määrittelee kolme luokkaa ongelmille, joita voi esiintyä XML-dokumentissa. Ongelmat ovat vakavuusjärjestyksessä Fatal error, Error, ja Warning [1].

Fatal Error on oikeinmuodostuneisuusvirhe. Heti kun jäsennin törmää tämän tyyppiseen ongelmaan, se ilmoittaa poikkeuksesta ja lopettaa jäsentämisen [1].

Error on virhe, joka ei ole oikeinmuodostuneisuusvirhe. Yleisimmin virhe on validisuusvirhe, mutta muitakin virheitä on olemassa. Jotkut jäsentimet luokittelevat nimiavaruuksien oikeinmuodostuneisuusvirheet virheinä. Jäsennin voi ilmoittaa tai olla ilmoittamatta poikkeuksesta, ja voi jatkaa jäsentämistä, tai keskeyttää jäsentämisen. Tämänkaltaiset virheet voivat aiheuttaa yhteensopivuusongelmia, koska kaksi jäsennintä voi käyttäytyä eri tavalla, vaikka käsiteltävä dokumentti on sama [1].

Warning ei ole varsinainen virhe vaan varoitus. Joka tapauksessa, se voi viitata jonkinlaiseen virheeseen dokumentissa. Esimerkiksi, jäsennin voi antaa varoituksen, jos se törmää elementtiin, jonka nimi on XMLDocument. Tästä seuraa varoitus, koska kaikki nimet, jotka alkavat ”XML”-merkkijonolla, on varattu W3C:n käyttöön tulevaisuuden standardeja varten. Jäsennin voi olla havaitsematta tämän tyyppisiä ongelmia, ja törmätessään sellaiseen jatkaa jäsentämistä, ilmoittamatta poikkeuksesta [1].

2.5. XML ja Internet

XML perustuu yksinkertaiseen tekstimuotoon. Vaikka tämä tarkoittaa Unicode-tekstimuotoa, ei vain yksinkertaista *ASCII* (American Standard Code for Information Interchange) -tekstiä, se voidaan muuttaa *UTF-8* (8-bit Unicode Transformation Format)-muotoon, tai ASCII-muotoon tiedonsiirtoa varten vanhimpia Internet-yhteyksiä ja laitteistoa varten. Tämä myös eliminoi joitakin huomattavia ongelmia, jotka liittyvät binääridatan tulkintaan erilaisilla alustoilla ja käyttöjärjestelmillä [5].

XML myös käyttää olemassa olevia Internet-protokollia, -ohjelmistoja ja -määrittelyjä milloin vain mahdollista, helpompaa datan lähetystä ja käsittelyä varten. Nämä ulottuvat

perussyntaksista, kuten *URI*:ista (Uniform Resource Indentifier), koodinumerohakemistoihin, kuten ISO:n määrittelemät maiden koodit. HTML voidaan myös esittää XML-yhteensopivalla tavalla XHTML:n avulla [5].

Kuten HTML, XML lähetetään useasti *HTTP* (Hyper Text Transfer Protocol)-protokollalla. Tämä tarkoittaa, että XML:ää voidaan käsitellä helposti olemassa olevilla web-palvelin-ohjelmistoilla, ja liikkua yritysten palomuurien läpi [5].

Vaikka XML ei ole suora korvaaja HTML:lle, tulevaisuuden HTML-versiot tullaan ilmaisemaan XML-syntaksilla, kuten XHTML. XML mahdollistaa kehittyneemmän verkkoarkkitehtuurin siirtämällä taakkaa palvelinpuolelta asiakkaan selaimelle tai toiselle sovellukselle. XML sisältää syntaksin, jota voidaan käyttää minkälaiselle datalle tahansa, sen kuvaavan metadatan, ja jopa viestiprotokollat, joita käytetään XML-tiedon siirtämiseen palvelimen ja asiakkaan välillä [5].

2.6. XML-syntaksi

Minimivaatimus oikein muodostetulle XML dokumentille on yksi elementti, joka on oikein aloitettu ja lopetettu. Tätä elementtiä kutsutaan juuri- tai dokumenttielementiksi. Se palvelee tilana muulle sisällölle. Dokumentin ulkoasu koostuu vaihtoehtoisesta prologista, dokumentin rungosta, joka koostuu dokumenttielementistä ja kaikesta, mitä se sisältää, sekä valinnaisesta epilogista [3].

Prologi antaa tietoa dokumentista. Prologi voi koostua seuraavista osista tässä luetellussa järjestyksessä:

1. XML-esittely
2. Kommentit
3. Prosessointiohjeet tai tyhjätilamerkki
4. DTD
5. Kommentit
6. Prosessointiohjeet tai tyhjätilamerkki

```
<?xml version="1.0"?>
<!--The previous line contains the XML declaration-->
<!--The following document type declaration contains no subsets-->
<!DOCTYPE foo [
]>
<!--This is the end of the prolog-->
```

Kuva 9. Esimerkkiprologi [3].

XML-esittely, ensimmäinen rivi kuvassa 9, antaa tietoa XML-määrityksen versiosta, jota on käytetty dokumentin muodostamisessa, dokumentin tekstin koodaustavan ja tiedon, tarvitseeko dokumentti ulkoisen DTD:n. Perussäännöt esittelyn tekemiseen ovat, että esittelyn pitää alkaa <?xml-merkinnällä, sen pitää kertoa XML:n versionumero, ja sen pitää päättyä ?> merkintään. Dokumentteja, joista puuttuu XML-esittely, käsitellään XML 1.0-määritysten mukaisesti. XML-esittelyn täytyy olla dokumentin ensimmäisellä rivillä [3].

Versio, joka on valinnainen, osoittaa, minkä XML-määrityksen mukaan dokumentti on muodostettu. Suurin ero XML 1.0:n ja 1.1:n välillä on, mitä merkkejä dokumentissa sallitaan [3].

Koodausesittely, mikä on myös valinnainen XML-esittelyssä, viittaa merkkitiedon koodaukseen dokumentissa. Koodaustavaksi voidaan määrittää esimerkiksi seuraavia enkoodausmuotoja: UTF-8, UTF-16, ISO-8859 ja ISO-2022-JP. Suositellaan, että käytetyt merkistöt ovat rekisteröity Internet Assigned Numbers Authority:ssa (IANA) [3].

Stand-alone-esittely, joka on myöskin valinnainen XML-esittelyssä, ilmoittaa, tarvitseeko dokumentti ulkopuolisia resursseja, kuten ulkoisen DTD:n. Dokumentit, jotka eivät sisällä stand-alone-esittelyä XML-esittelyssä, mutta sisältävät ulkoisia resursseja, automaattisesti olettavat arvon epätodeksi [3].

Dokumentin tyypin määrittely (DOCTYPE) tuo DTD:n dokumentille. Se voi sisältää sisäisen alijoukon, mikä tarkoittaa, että esittelyt tehtäisiin suoraan DOCTYPE:n sisällä, ja / tai lisäksi sisältää ulkoisen alijoukon, mikä tarkoittaa, että se voisi sisältää esittelyitä ulkoisista lähteistä.

Sisäiset ja ulkoiset alijoukot yhdessä muodostavat dokumentin DTD:n [3]. Kuvassa 4 on esimerkki sisäisestä ja kuvassa 5 ulkoisesta viittauksesta. Kuvassa 10 on esitelty molemmat tavat.

```
<!DOCTYPE foo SYSTEM "foo.dtd" [
    <!ELEMENT foo (#PCDATA)>
]>
```

Kuva 10. Dokumentin tyypin määrittely sisäisellä ja ulkoisella alijoukolla [3].

Runko XML-dokumentissa koostuu dokumenttielementistä ja sen sisällöstä. Yksinkertaisimmassa tapauksessa runko voi olla yksi tyhjä elementti. Termi dokumenttipuu tarkoittaa samaa asiaa, kuin dokumentin runko. Dokumenttielementti on puun juuri, joka haarautuu elementteihin, jotka sisältyvät dokumenttielementtiin [3].

Epilogi viittaa merkkaukseen, joka sijaitsee rungon sulkutunnisteen jälkeen. Se voi sisältää kommentteja, tyhjätilamerkkejä ja prosessointiohjeita. Epilogit eivät ole pakollisia, ja muut kuin sellaiset, jotka sisältävät mahdollisesti tyhjätilamerkkejä, eivät ole yleisiä. Monet jäsentimet eivät edes jäsennä dokumenttielementin sulkutunnisteen ulkopuolelta. Tämän rajoituksen takia epilogin mahdollinen käyttö on lisätä kommentteja jollekin, joka lukee XML-dokumenttia. Tämäntyylinen käyttö ei aiheuta ongelmia, jos jäsennin ei lue epilogia [3].

2.7. XML – elementit

Elementit ovat dokumentin perusta, ja ainakin yksi tarvitaan, jotta dokumenttia voidaan kutsua oikein muodostetuksi. Elementti koostuu aloitus-ja lopetusmerkinnästä, sekä sisällöstä, joka on määritelty näiden väliin. Elementit ilman sisältöä ovat poikkeus tähän sääntöön, koska elementti voi koostua tyhjä elementti-merkinnästä. Elementillä voi olla useita attribuutteja. Saman elementin attribuutit täytyy olla nimetty eri nimillä [3]. Elementin määrittely on muotoa:

<elementti>sisältö<elementti>

Aloitusmerkinnät koostuvat < -merkistä, nimestä, attribuuteista, sekä > -merkistä [3], esimerkiksi <elementti arvo='1'>.

Lopetusmerkinnät ovat muotoa </nimi>, jossa nimi on sama, kuin vastaavassa aloitusmerkinnässä [3].

Tyhjä elementti-merkintä eli elementti ilman sisältöä voidaan esittää muodossa aloitusmerkintä sekä lopetusmerkintä [3]. Voidaan myös käyttää merkintää <elementti/>.

Attribuutteja voi ajatella elementin ominaisuuksina. Attribuutit määritellään nimen ja arvon avulla käyttäen seuraavaa syntaksia: nimi = "arvo" tai nimi = 'arvo'. Arvo voidaan ympäröidä siis joko lainausmerkeillä tai heittomerkeillä, kunhan samaa merkintää käytetään saman attribuutin ympäröimiseen. Elementillä voi olla useita eri attribuutteja [3].

<elementti atttibuutti1="1" attribuutti2="2">sisältö<elementti>

Attribuutilla on oltava arvo, joka voi olla tyhjä. Attribuuttien arvot eivät voi sisältää < tai &-merkkejä, jos niitä ei edellä \-merkki [3].

CDATA-osuudet antavat mahdollisuuden käyttää kaikkia valideja Unicode-merkkejä niiden kirjallisessa muodossaan. CDATA-sisältö ohitetaan jäsennettäessä, joten sitä voidaan käyttää, kun halutaan sisältöä, jota halutaan käyttää sen kirjallisessa muodossa, ja sitä ei haluta käsiteltävän osana dokumenttia. CDATA-osuudet alkavat <![CDATA[-merkinnällä, jota seuraa sen sisältö, jonka jälkeen seuraa]]>-merkintä [3].

Esimerkiksi, jos kirjoitetaan artikkelia XML:stä käyttämällä XML:ää, CDATA-osuuksiin voitaisiin laittaa XML-tietoa ilman, että tarvitsisi huolehtia erikoismerkkien, kuten < käsittelystä [3].

Kommenttien avulla voidaan lisätä dokumenttiin muistiinpanoja. Tämä on verrattavissa siihen, että lisätään muistiinpanoja ohjelmakoodiin. Ne eivät vaikuta dokumenttiin, mutta niitä voidaan käyttää lisäämään muistiinpanoja tai muuta informaatiota henkilölle, joka dokumenttia lukee. Tästä syystä jäsentimien ei tarvitse jäsentää kommentteja. Monet

jäsentimet kuitenkin sallivat pääsyn kommenttien sisältöön [3]. Kommentin määrittely on muotoa:

<!--tämä on kommentti-->

Prosessointiohjeet antavat sovelluskohtaiset ohjeet XML-dokumenttia käsittelevälle ohjelmalle siitä, miten dokumenttia tulisi prosessoida. Prosessointiohjeet alkavat <?-merkinnällä, jota seuraa kohde, välilyönti ja sitten varsinainen ohje sekä lopetusmerkintä ?> [3]. Prosessointiohjeen määrittely on muotoa:

<?kohde ohjeet?>

2.7.1. Entiteetit

XML-dokumentit eivät välttämättä ole sama asia, kuin XML-tiedostot. XML-dokumentti voi koostua useista tiedostoista. Ne eivät ole välttämättä edes tiedostoja, vaan voivat olla esimerkiksi tietokantaan tallennettua tietoa, osa jotakin isompaa tiedostoa tai jotakin muuta [1].

Itsenäisiä tietoja tai tiedostoja, jotka koostavat XML-dokumentin, kutsutaan e*ntiteeteiksi*. Jokaisella XML-dokumentilla on ainakin yksi entiteetti, dokumenttientiteetti, joka on varastoyksikkö eli tiedosto tai joku muu tietovarasto, joka sisältää dokumentin juurielementin. Kaikilla muilla dokumentin entiteeteillä on oma nimensä. On olemassa viisi nimettyä entiteettityyppiä jotka on luokiteltu seuraavien kriteerien perusteella [1]:

- *Sisäisen entiteetin* korvaava teksti on määritelty merkkijonona dokumentin DTD:hen. Korvaava teksti *ulkoisessa entiteetissä* on määritelty toisessa tiedostossa.

- Jäse*nnetty entiteetti* sisältää XML-tietoa. Se on oikein muodostettu ja voi olla kokonainen XML-dokumentti, jos se sisältää juurielementin (joillakin entiteeteillä ei ole välttämättä juurielementtiä, koska niitä käytetään osana toista dokumenttia).

- *Jäsentämätön entiteetti* voi sisältää mitä vaan, kuten binääridataa. Jäsentämätön entiteetti voi sisältää tiedon, joka ilmaisee mitä sen sisältämä tieto esittää, kuten kuvadataa.

- *Yleistä entiteettiä* käytetään dokumentin ilmentymässä. Yleinen entiteettiviittaus alkaa ’&’-merkillä.

- *Parametrientiteettiä* käytetään DTD:n sisällä. Parametrientiteettiviittaus alkaa ’%’-merkillä.

Nimetyt viisi entiteettiä ovat [1]:

Sisäiset jäsennetyt yleiset entiteetit ovat tuttuja entiteettiviittauksia ovat esimerkiksi & ja ©. Ne on kuvattu DTD:n sisällä, ja ovat sisäisiä jäsennettyjä yleisiä entiteettejä:

<!ENTITY copy ”Copyright”>

Ulkoiset jäsennetyt yleiset entiteetit ovat kuten sisäiset jäsennetyt yleiset entiteetit, paitsi että niiden korvaava teksti luetaan ulkoisesta tiedostosta. Esimerkiksi legal-elementti määrittelee sisällön luettavaksi tietystä osoitteesta:

<!ENTITY legal SYSTEM ”http://www.example.com/legal.xml”>

Ulkoiset jäsentämättömät yleiset entiteetit viittaavat tiedostoihin, jotka eivät sisällä XML-tietoa. Ne on kuvattu samalla tavalla kuin ulkoiset jäsennetyt entiteetit, mutta niillä on myös notaatio. Esimerkiksi logo – entiteetti voidaan määritellä notaatiolla image/png:

<!NOTATION PNG SYSTEM ”image/png”>
<!ENTITY logo SYSTEM ”http://www.example.com/logo.png” NDATA PNG>

Sisäisiä jäsennettyjä parametrisia entiteettejä käytetään DTD:n sisällä. Korvaava teksti saadaan DTD:n sisältä. Viittaukset näihin entiteetteihin alkavat ’%’-merkillä. Niitä käytetään monesti parametrisoimaan sisältömalleja ja attribuuttityyppejä. Esimerkiksi DTD voisi määritellä intermod.redecl.module-parametrientiteetin sanaksi IGNORE:

<!ENTITY % intermod.redecl.module ”IGNORE”>

Toisin kuin yleisiä entiteettiviittauksia, indermod.redecl.module-parametrientiteettiviittausta voidaan käyttää vain DTD:ssä eikä dokumentin ilmentymässä.

Ulkoisia jäsennettyjä parametrisia entiteettejä käytetään vain DTD:n sisällä. Korvaavan tekstin tuo DTD:n osa annetusta osoitteesta. Viittaukset näihin entiteetteihin alkavat %-merkillä. Ne usein yhdistävät eri osia modulaarisesta DTD:stä yhdeksi kokonaisuudeksi.
Esimerkin DTD määrittelee dbpool:in, parametrisen entiteetin, käyttämällä julkista tunnistetta, joka lataa DTD-osuuden osoitteesta dbpoolx.mod:

<!ENTITY % dbpool PUBLIC
”-//OASIS//ELEMENTS DocBook XML Information Pool V4.1.2//EN”
“dbpoolx.mod”>

2.7.2. Nimiavaruudet XML:ssä

XML-dokumentit voivat koostua myös osaksi ulkopuolisista lähteistä. Elementtien ja attribuuttien nimet saattavat toistua. Tällöin attribuuttien ja elementtien tarkoitus pitäisi arvata riippuen kontekstista. Ikävä kyllä XML-sovellukset eivät osaa arvata kontekstia, joten dokumentti ei enää kuvaisi haluttua määrittelyä [3].

Nimiavaruudet korjaavat tämän ongelman. Nimiavaruudet ovat kokoelma nimiä, jotka on määritelty URI:lla. Ne eivät ole osa XML-määrittelyä, mutta niillä on oma määrittelynsä, joka soveltuu XML:ään. Käyttämällä nimiavaruuksia, dokumentin sisällä olevat nimet kykenevät palauttamaan alkuperäisen tarkoituksensa, jopa yhdistettynä toiseen dokumenttiin joka sisältää joitakin samoja nimiä, joilla on täysin erilainen merkitys [3].

3. XML-rajapinnat

Yleensä XML-jäsentimet toimivat jonkin XML-rajapinnan mukaisesti. XML-rajapintoja on olemassa useita, mutta tässä tutkimuksessa keskitytään yleisimmin käytössä oleviin rajapintoihin, SAX ja DOM. SAX-rajapinnan toiminta perustuu tapahtumiin, joissa löytäessään dokumentista kiinnostavaa informaatiota jäsennin ilmoittaa sovellukselle. DOM-rajapinta taas mallintaa dokumentin puurakenteeseen, josta tietoa voidaan hakea.

3.1. SAX

SAX-lyhenne tulee sanoista Simple Api For XML. SAX:in kehitys alkoi vuonna 1997, kun Peter Murray-Rust ja monet muut, jotka kirjoittivat XML-jäsentimiä Javalle, päättivät, että monen samankaltaisen, mutta yhteen sopimattoman rajapinnan ylläpitäminen ei ollut järkevää [1]. Murray-Rust oli ensimmäinen ehdottamaan uutta rajapintaa, jota hän kutsui *YAXPAPI*:ksi (Yet Another XML Parser API). Syy siihen, että hän halusi uuden rajapinnan, oli siinä, että hän oli kyllästynyt ylläpitämään useita yhteensopimattomia rajapintoja, joita hänen JUMBO-sovelluksensa tarvitsi. Sen sijaan hän halusi rajapinnan, johon kaikki olisivat tyytyväisiä. Tim Bray sekä David Meggison liittyivät projektiin, ja työ alkoi xml-dev:in postituslistalla, jossa monet henkilöt osallistuivat. SAX 1.0 julkaistiin 11.5.1998.

Vuonna 1999, aloitettiin SAX2:en kehitys [1]. Muutokset SAX1:een olivat suuria. Vaikka sama tapahtumapohjainen arkkitehtuuri pidettiin, melkein kaikki SAX1:en luokat korvattiin. Suurin syy uuden version kehittämiseen oli, että haluttiin tuki nimiavaruuksille. Muitakin uusia ominaisuuksia lisättiin SAX2:een, kuten suodattimet ja tuki DTD:ille. SAX2 valmistui toukokuussa 2000 ja osoittautui jopa menestyneemmäksi kuin SAX1. Kaikki suuret jäsentimet, jotka tukevat SAX:ia, tukevat myös SAX2:ta [1].

SAX-rajapinnasta on myös olemassa toteutus useille muillekin kielille Javan lisäksi. SAX Project-internetsivuilta löytyy epävirallinen lista muista ohjelmointikielistä, joille rajapinta on olemassa, kuten Pascal, Perl, Python ja C++.

3.1.1. SAX-rajapinnan hyvät ja huonot puolet

Koska SAX on tapahtumapohjainen, SAX tarjoaa useita etuja käyttäjälleen. Samaan aikaan sen käyttöön liittyy haittoja. Molempia puolia esitellään tässä osuudessa [2].

SAX sisältää neljä erityistä etua [2]:

1. Välitön analyysi dokumentin sisällöstä

SAX luo tapahtumia jatkuvasti samalla, kun prosessoi dokumenttia. Dokumentin analysointi voidaan aloittaa välittömästi, ja koko dokumentin läpikäyntiä ei tarvitse odottaa ennen sisällön analysoinnin aloittamista. Tätä voidaan verrata mediavirtaan, missä median sisältö tuodaan näkyville välittömästi, eikä tarvitse odottaa, että koko media on luettu.

2. Vähemmän rajoituksia muistivaatimuksille

SAX tutkii dokumentin sisältöä kun se lukee dokumenttia ja samalla välittömästi luo tapahtumia käsittelevässä sovelluksessa. Tästä johtuen sen ei tarvitse tallentaa dataa, jonka se on jo käsitellyt, mihinkään erityiseen tietorakenteeseen.

3. Isojen dokumenttien helppo käsitteleminen

Koska dokumentin sisältöä ei tallenneta muistiin, on helppoa prosessoida hyvin suuria dokumentteja verrattuna toisiin prosessointimenetelmiin. Toiset tekniikat, jotka vaativat koko dokumentin tallentamista muistiin ennen prosessoinnin aloittamista, voivat joskus asettaa vakavia rajoituksia systeemin resursseille.

4. Nopea jäsentäminen

Sovelluksen ei tarvitse käsitellä koko dokumenttia, jos se on kiinnostunut tietämään täyttääkö dokumentti tiettyä ehtoa. Kun ehto on täytetty, enempää dokumentin käsittelyä ei tarvita, ja käsittely voidaan keskeyttää. Toiset tekniikat vaativat, että koko dokumentti käydään läpi, ennen kuin mitään muuta voidaan tehdä.

SAX sisältää myös huonoja puolia [2]:

1. Ei taaksepäin navigointia

SAX käy dokumentin vain kerran läpi. Kun se lukee dokumentin osia, se ei voi siirtyä taaksepäin uudelleenlukemaan dataa, jonka se on prosessoinut, ellei koko jäsentämisprosessia aloiteta alusta.

2. Ei datan manipulointia

Koska SAX ei tallenna dataa, jota se on käsitellyt, tätä dataa ei voida muokata ja sitten tallentaa takaisin alkuperäiseen dokumenttiin.

3. Ei dokumenttien luomista

Koska SAX ei luo muistiin mallia dokumentista, XML-dokumenttia ei voida rakentaa käyttämällä SAX-jäsennintä.

3.1.2. Jäsentäminen

Jäsentäminen on prosessi, jossa luetaan XML-dokumenttia ja raportoidaan sen sisällöstä asiakasohjelmistolle samalla, kun tarkistetaan, että dokumentti on oikein muodostettu. SAX esittää jäsentimet XMLReader-rajapinnan ilmentyminä. Luokka, joka toteuttaa tämän rajapinnan, vaihtelee eri jäsentimien välillä. Esimerkisi Xerces-jäsentimellä tämä luokka on org.apache.xerces.parsers.SAXParser, kun taas Crimson-jäsentimellä vastaava luokka on org.apache.crimson.parser.XMLReaderImpl. Jäsentimen ilmentymä tehdään yleensä kutsumalla XMLReaderFactory.createXMLReader()-metodia. Tälle jäsentimen ilmentymälle annetaan InputSource-objekti, joka sisältää XML-dokumentin, ja jäsentäminen tapahtuu kutsumalla jäsentimen parse()-metodia. Jäsennin lukee dokumentin, ja heittää poikkeuksen, jos se huomaa, että dokumentti ei ole oikein muodostettu XML-dokumentti [1]. Kuva 11 havainnollistaa jäsentämistä.

```
import org.xml.sax.*;
import org.xml.sax.helpers.XMLReaderFactory;
import java.io.IOException;

public class SAXChecker {
  public static void main(String[] args) {
    if (args.length <= 0) {
      System.out.println("Usage: java SAXChecker URL");
      return;
    }
    try {
      XMLReader parser = XMLReaderFactory.createXMLReader();
      parser.parse(args[0]);
      System.out.println(args[0] + " is well-formed.");
    }
    catch (SAXException e) {
      System.out.println(args[0] + " is not well-formed.");
    }
    catch (IOException e) {
      System.out.println(
       "Due to an IOException, the parser could not check "
       + args[0]
      );
    }
  }
}
```

Kuva 11. Ohjelma, joka jäsentää XML-dokumentin, ja ilmoittaa onko se oikein muodostettu [1].

3.1.3. ContentHandler-rajapinta

SAX ei mallinna dokumenttia tietorakenteeseen kuten DOM, vaan käsiteltäessä dokumenttia sille rekisteröidään erillinen kuuntelija, joka ilmoittaa jokaisen dokumentin solmun kohdalla erilaisista tapahtumista. Tämä tekee SAX-rajapinnasta erittäin vähän muistia vievän rajapinnan. Tapahtumat, joihin jäsennin voi törmätä, on määritelty SAX:in ContentHandler-rajapinnassa. Yhdelle XMLReader-luokalle ei voi rekisteröidä useampia kuuntelijoita [1]. ContentHandler-rajapinnassa (kuva 12) esitellään 11 tapahtumaa. Kun jäsennin lukee dokumenttia, niin se kutsuu tämän rajapinnan metodeja. Kun jäsennin törmää aloitusmerkintään, se kutsuu startElement-metodia. Kun jäsennin törmää tekstisisältöön, niin se kutsuu characters-metodia. Kun jäsennin törmää lopetusmerkintään, niin se kutsuu endElement-metodia jne. Tieto siitä, mitä tietoa metodille tulee, välittyy metodin parametrien välityksellä. Esimerkiksi characters-metodin parametrina tulevat merkit, joihin jäsennin on törmännyt. Elementtien attribuutit tulevat samalla tavalla startElement-metodin välittämien parametrien mukana [1].

```
public interface ContentHandler {
  public void setDocumentLocator(Locator locator);
  public void startDocument() throws SAXException;
  public void endDocument() throws SAXException;
  public void startPrefixMapping(String prefix, String uri)
   throws SAXException;
  public void endPrefixMapping(String prefix)
   throws SAXException;
  public void startElement(String namespaceURI, String localName,
   String qualifiedName, Attributes atts) throws SAXException;
  public void endElement(String namespaceURI, String localName,
   String qualifiedName) throws SAXException;
  public void characters(char[] text, int start, int length)
   throws SAXException;
  public void ignorableWhitespace(char[] text, int start,
   int length) throws SAXException;
  public void processingInstruction(String target, String data)
   throws SAXException;
  public void skippedEntity(String name)
   throws SAXException;
}
```

Kuva 12. ContentHandler-rajapinta [1].

3.1.4. ContentHandler-rajapinnan toteuttaminen

ContentHandler-rajapinnan avulla saamme tiedon tapahtumista, joihin jäsennin törmää dokumenttia läpikäydessään. Asiakasohjelma voi määritellä toimintoja, joita ohjelma tekee törmätessään tiettyyn tapahtumaan. Toiminnot määritellään ContentHandler-rajapinnan metodit toteuttavaan luokkaan. Kuvan 13 esimerkki lukee dokumenttia tulostaen löytämiensä elementtien aloitusmerkinnät ja niiden lopetusmerkinnät, ja tulostaa löytämiensä kirjaelementtien sisältä löytämänsä merkkitiedon. Esimerkkiluokka toteuttaa vain rajapinnan characters(), startDocument(), endDocument(), startElement() ja endElement()-metodit. Kaikki muutkin rajapinnan metodit täytyy kuitenkin esitellä.

Rajapinnassa määriteltyjen metodien lisäksi luokalla on oma konstruktori-metodinsa. Luokkaan voidaan vapaasti toteuttaa useampia muita metodeja tiedon käsittelemiseen. Luokalla voi olla useita konstruktori-metodeja [1].

Kun luokka törmää elementtiin, jonka sisältä löytyy title-merkkijono, asetetaan print-lippu todeksi. Kun jäsennin törmää merkkitietoon, joka sijaitsee elementin aloitus-ja lopetusmerkkien välissä, se kutsuu characters-metodia. Metodin parametrina tulee jono merkkejä, joihin jäsennin on törmännyt. Lisäksi tulee tieto siitä, mistä kohden taulukkoa alkaa uusimpien löydettyjen merkkien aloituskohta, sekä tieto siitä, miten pitkä uusi merkkijono on.

Jos halutaan tulostaa elementin aloitus-ja lopetusmerkintöjen sisältä löytyneet merkit, tulostetaan ne alkaen start-kohdasta ja lopettaen tulostus start + length-kohtaan. Törmättäessä elementin lopetusmerkintään asetetaan tulostus-lippu epätodeksi.

```
import org.xml.sax.*;
public class SAXHandler implements ContentHandler{
    boolean print=false;   //tulostetaanko merkkitieto
    public SAXHandler() {}
    public void characters(char[]text,int start,int length){
    if(print){
        for(int i=start;i<length+start;i++)
            System.out.print(text[i]);
        System.out.println();
        }
    }
    public void startDocument(){
    System.out.println("<dokumentti>");
    }
    public void endDocument(){
    System.out.println("</dokumentti>");
    }

    public void startElement(String namespaceURI, String localName, String
         qualifiedName, Attributes atts){
    System.out.println("<"+localName+">");
    if(localName.equals("title"))print=true;
    else print=false;
    }

    public void endElement(String namespaceURI, String localName, String
          qualifiedName){
    System.out.println("</"+localName+">");
    print=false;
    }
    //näitä ei käytetä esimerkissä, mutta ne pitää esitellä.
    public void startPrefixMapping(String prefix, String uri){}
    public void endPrefixMapping(String prefix){}
    public void ignorableWhitespace(char []text, int start, int length)
         throws SAXException{}
    public void processingInstruction(String target, String data){}
    public void setDocumentLocator(Locator locator){}
    public void skippedEntity(String name){}
    }
```

Kuva 13. Toteutus ContentHandler-rajapinnasta.

3.1.5. EntityResolver-rajapinta

XML-dokumentti koostuu entiteeteistä. Kukin entiteetti tunnistetaan *julkisella tunnisteella*, *systeemitunnisteella* tai molemmilla. Systeemitunnisteet ovat *URL* (Uniform Resource Locator)-osoitteita, ja julkiset tunnisteet tarvitsevat normaalisti jonkin katalogimenetelmän, jolla ne voidaan muuttaa URL-osoitteiksi. XML-jäsennin lukee kunkin entiteetin käyttämällä InputSource-rajapintaa, joka on kytketty oikeaan URL-osoitteeseen[1].

3.1.6. ContentHandler-rajapinnan käyttäminen

Pelkästään toteuttamalla ContentHandler-rajapinnan toteuttava luokka emme voi vielä käsitellä XML-dokumenttia. Lisäksi tarvitaan jäsennin, jolle asetetaan ContentHandler-rajapinnan toteuttava olio sen omaksi ContentHandler-olioksi. Tämä tapahtuu SAX-rajapinnassa määritellyllä setContentHandler-metodilla. Jäsentäminen tapahtuu parse()-metodia kutsumalla, jolloin jäsennin aloittaa dokumentin käsittelemisen. Jäsennettäessä pitää varautua poikkeustilanteisiin, joita voi tulla jäsennettäessä dokumenttia, sekä lisäksi varautua tiedoston avaamiseen liittyviin virheisiin. Tätä varten otetaan kiinni kaksi eri poikkeusta. SAXException seuraa, jos jäsentämisessä törmätään virhetilanteeseen, kuten virheellisesti muodostettuun dokumenttiin. IOException tapahtuu, jos tiedoston avaamisessa ilmenee ongelmia [1]. Kuva 14 havainnollistaa rajapinnan käyttöä.

```
public static void main(String[] args) {
    try {
         //luodaan jäsennin
        XMLReader parser = XMLReaderFactory.createXMLReader();
        //luodaan uusi ContentHandler
        ContentHandler handler = new SAXHandler();
        //asetetaan ContentHandler jäsentimelle
        parser.setContentHandler(handler);
        //jäsennetään annettu tiedosto
        parser.parse(args[0]);
    }catch(SAXException se){
    }catch (IOException ioe){}
```

Kuva 14. ContentHandler-rajapinnan käyttäminen.

Uusi jäsenninolio luodaan main()-metodin sisällä kutsumalla jäsenninolion ilmentymän palauttavaa XMLReaderFactory.createXMLReader()-metodia.

Seuraavaksi luodaan ContentHandler-olio, joka asetetaan XML-jäsentimelle. Tämän jälkeen dokumenttia ruvetaan käsittelemään.

Yksi huomionarvoinen asia on, että ei ole ohjelmakoodia, joka kutsuisi mitään SAXHandler-luokan metodia. Ohjelmakoodi, joka näitä metodeja kutsuu, on piilotettu luokkakirjaston sisälle [1].

3.1.7. XMLReader-rajapinta

XML-määrittely antaa jäsentimille joskus hämmentävän paljon vapauksia XML-dokumenttien prosessoinnissa. Jäsentimet voivat olla validoivia tai validoimattomia, tukea epästandardeja koodauksia, tarkistaa nimiavaruuksista oikeinmuodostuneisuuden tai olla tarkistamatta, ja paljon muuta. Riippuen siitä, mitä vaihtoehtoja jäsentimet noudattavat eri vaihtoehtojen välillä, ne saattavat valmistaa erilaisen näkymän samasta XML-dokumentista. Joissakin tapauksissa, jotkut jäsentimet saattavat ilmoittaa dokumentin olevan oikein muodostettu, ja toinen ilmoittaa, että dokumentti on virheellinen [1]. Tukeakseen eri mahdollisuuksia eri jäsentimien välillä XMLReader-rajapinta, joka esittää jäsennintä SAX-rajapinnassa, on tarkoituksella tehty epämääräiseksi. Se voidaan alustaa monella eri tavalla, se voi lukea XML-dokumentteja eri medioista, ja se voidaan alustaa tunnetuilla ja tuntemattomilla piirteillä ja ominaisuuksilla [1].

3.1.7.1. Jäsenninolioiden luominen

Koska XMLReader on rajapinta, sillä ei ole konstruktorimetodia. Sen sijaan se käyttää staattista metodia XMLReaderFactory.createXMLReader() hakeakseen XMLReader-luokan ilmentymän. XMLReaderFactory-luokassa on kaksi metodia jäsentimen hakemiseen [1]:

public static XMLReader createXMLReader() throws SAXException
public static XMLReader createXMLReader(String classname) throws SAXException

Parametriton metodi palauttaa oletusjäsentimen. Tämä on määritelty org.xml.sax.driver-järjestelmämuuttujassa. Jäsentimien valmistajien oletetaan muuttavan tätä metodia siten, että se palauttaa heidän jäsentimensä ilmentymän, mutta käytännössä vain muutama tekee niin. Tästä johtuen, jos ohjelmassa halutaan käyttää muita jäsentimiä, kannattaa asettaa org.xml.sax.driver-muuttuja komentoriviltä käyttäen -D optiota [1]:

java -Dorg.xml.sax.driver= net.sf.saxon.aelfred.SAXDriver

Jos luokkapolulla on useita versioita SAX-luokista, niin virtuaalikone asettaa jäsentimeksi ensimmäisenä löytämänsä jäsentimen. Jos halutaan käyttää jotakin tiettyä luokkaa, niin sitä

voidaan kysyä käyttämällä parametrillista metodia. Seuraava esimerkki pyytää Apachen XML-jäsennintä [1]:

XMLReaderFactory.createXMLReader(”org.apache.xerces.parsers.SAXParser”);

Jos kutsu ei onnistu, niin seuraa SAXException-poikkeus. Koska monesti ei ole takeita siitä, että tietty jäsennin on asennettu kaikille alustoille missä ohjelmaa käytetään, tähän kannattaa varautua [1].

3.1.7.2. Parse-metodi

Kun jäsentimestä on ilmentymä, sillä voidaan jäsentää dokumentti. Tätä varten on olemassa kaksi metodia [1].

public void parse (String systemID) throws SAXException
public void parse (InputSource IN) throw SAXException

Jäsentämismetodi ottaa parametrinaan joko merkkijonon tai InputSource-olion. InputSource on kääre useille erilaisille InputStream-olioille, joista XML-dokumentti voidaan lukea. Jos jäsentimelle annetaan merkkijono parametrina, sitä käytetään InputSource-olion luomiseen, joka välitetään edelleen toiselle parse()-metodille [1].

3.1.7.3. InputSource-luokka

InputSource on kääre kolmelle eri luokalle. XML-dokumentti voidaan lukea tästä luokasta. Välitettävä parametri voi olla joko InputStream, Reader tai String-tyyppiä. Sillä on myös konstruktori, joka ei ota vastaan mitään parametrina. Luokalla on useita set-metodeita, jonka avulla asetuksia voidaan myöhemmin muuttaa [1]. Kuva 15 havainnollistaa luokan sisältöä.

```
package org.xml.sax;
public class InputSource {
  public InputSource()
  public InputSource(String systemID)
  public InputSource(InputStream byteStream)
  public InputSource(Reader characterStream)
  public void        setPublicId(String publicID)
  public String      getPublicId()
  public void        setSystemId(String systemID)
  public String      getSystemId()
  public void        setByteStream(InputStream byteStream)
  public InputStream getByteStream()
  public void        setEncoding(String encoding)
  public String      getEncoding()
  public void        setCharacterStream(Reader characterStream)
  public Reader      getCharacterStream()
}
```

Kuva 15. InputSource-luokka [1].

3.1.7.4. Errorhandler-rajapinta

Poikkeuksen havaitseminen keskeyttää jäsentämisen, mutta kaikkiin törmättävissä oleviin ongelmiin ei tarvita niin radikaaleja toimenpiteitä. Erityisesti validisuusvirheistä ei seuraa poikkeusta, koska se seisauttaisi jäsentämisen. Jos halutaan, että ohjelma informoi myös virheistä, jotka eivät ole tyyppiä fatal, niin täytyy rekisteröidä virheenkäsittelijäolio XMLReader-oliolle [1].

```
package org.xml.sax;
public interface ErrorHandler {
  public void warning(SAXParseException exception)
   throws SAXException;
  public void error(SAXParseException exception)
   throws SAXException;
  public void fatalError(SAXParseException exception)
   throws SAXException;
}
```

Kuva 16. Errorhandler-rajapinta [1].

Kuten kuvan 16 metodeista ilmenee, ErrorHandler-rajapinnan metodit vastaavat XML-määrityksen mukaisia virhetilanteita. Kun jäsennin huomaa virhetilanteen, se välittää SAXParseException-poikkeuksen oikealle metodille [1]. Fatal-tyyppiseen virheeseen törmätessään jäsennin heittää aina poikkeuksen.

3.1.7.5. SAX2-rajapinnan piirteet ja ominaisuudet

SAX2 määrittelee standardeja metodeja XMLReader-olion piirteiden ja ominaisuuksien määrittelemiseen. On mahdollista vaihtaa jäsentimen käyttäytymistä, kuten pyytämällä XML-jäsennintä validoimaan tai olemaan validoimatta dokumentti, ja rekisteröidä uusia tapahtumakäsittelijöitä käyttämällä metodeita getFeature, setFeature, getProperty ja setProperty [8].

Piirteet ovat aina totuusarvoja. Ominaisuudet ovat olioita. Kaikki ominaisuudet ja piirteet on kuvattu URI:lla [8] , kuten http://www.acme.com/features/foo.

Kaikkien SAX2-jäsentimien täytyy tunnistaa ainakin rajapinnan piirteet http://xml.org/sax/features/namespaces sekä http://xml.org/sax/features/namespace-prefixes (ainakin pystyä hakemaan niiden ominaisuuksien arvo, jos ei asettamaan niitä) ja tukemaan ”tosi”-arvoa nimiavaruusominaisuudelle, ja ”epätosi”-arvoa namespace-prefixes-ominaisuudelle. Nämä vaatimukset takaavat, että kaikki SAX2-rajapinnan toteuttavat XML-jäsentimet tarjoavat ainakin pienimmän mahdollisen tuen nimiavaruuksille korkeamman tason määrityksiä varten. XML-jäsentimiä ei vaadita tunnistamaan, tai tukemaan mitään muita piirteitä tai ominaisuuksia [8]. Kuva 17 havainnollistaa, miten validointi asetetaan.

```
XMLReader r = new MySAXDriver();
 // try to activate validation
 try {
   r.setFeature("http://xml.org/sax/features/validation", true);
 } catch (SAXException e) {
   System.err.println("Cannot activate validation.");
 }
 // register event handlers
 r.setContentHandler(new MyContentHandler());
 r.setErrorHandler(new MyErrorHandler());
 // parse the first document
 try {
   r.parse("http://www.foo.com/mydoc.xml");
 } catch (IOException e) {
   System.err.println("I/O exception reading XML document");
 } catch (SAXException e) {
   System.err.println("XML exception reading document.");
 }
```

Kuva 17. Validoinnin asettaminen [15].

Taulukko 1 sisältää SAX2-rajapinnan piirteet ja talulukko 2 SAX2-rajapinnan ominaisuudet. Jäsentimien toteuttajat voivat vapaasti lisätä omia piirteitä [8].

Taulukko 1. SAX2-rajapinnan piirteet [8].

Piirteen tunniste	Kirjoitus-oikeudet	Oletusarvo	Kuvaus
external-general-entities	luku/kirjoitus	ei määritelty	Raportoi, prosessoiko jäsennin ulkoisia yleisiä entiteettejä; aina tosi, jos validoidaan.
external-parameter-entities	luku/kirjoitus	ei määritelty	Raportoi, jäsentääkö tämä jäsennin ulkoisia parametrisia entiteettejä; aina tosi, jos validoidaan.
is-standalone	(jäsentämisen aikana) vain luku, (kun ei jäsennetä) ei olemassa	ei olemassa	Voidaan tutkia vain jäsentämisen aikana sen jälkeen, kun startDocument-takaisinkutsu on suoritettu. Arvo on tosi, jos dokumentissa määriteltiin standalone=”yes” sen XML-esittelyssä, muuten se on epätosi.
lexical-handler/parameter-entities	luku/kirjoitus	ei määritelty	Arvo “tosi” viittaa siihen, että LexicalHandler raportoi parametrisien entiteettien alun ja lopun.
namespaces	luku/kirjoitus	tosi	Arvo “tosi” ilmoittaa, että nimiavaruus URI:t ja etuliitteettömät paikalliset nimet elementeille ja attribuuteille ovat mahdollisia.
namespace-prefixes	luku/kirjoitus	epätosi	Arvo “tosi” viittaa siihen, että XML:n kvalifioitujen nimien (etuliitteineen) ja attribuutit (sisältäen xmlns* attribuutit) ovat käytettävissä.
resolve-dtd-uris	luku/kirjoitus	tosi	Arvo “tosi” tarkoittaa että järjestelmätunnisteet esittelyissä absolutisoidaan (suhteessa niiden URI:in) ennen raportointia. Arvo ”epätosi” viittaa siihen, että noita tunnisteita ei absolutisoida; jäsentimet tarjoavat URI:n Locator.getSystemID()-kutsusta.
string-interning	luku/kirjoitus	ei määritelty	Sisältää arvon “tosi”, jos kaikilla XML-nimillä elementeille, etuliitteille, attribuuteille, entiteeteille, notaatioille ja paikallisille nimille, ja nimiavaruus-URI:lle tullaan soveltamaan java.lang.String.intern()-metodia. Tämä tukee nopeaa testausta merkkijonojen vastaavuudessa toisiinsa, jolloin vältytään hitaamman String.equals()-kutsun käytöstä.
unicode-normalization-checking	luku/kirjoitus	epätosi	Kontrolloi, raportoiko jäsennin Unicode-normalisointivirheet kuten kuvattu osuudessa 2.13 ja liitteessä B XML 1.1-suosituksessa. Jos tosi, Unicode–normalisointivirheet raportoidaan käyttämällä Errorhandler.error()-kutsua. Tällaiset virheet eivät itsessään ole fatal-tyyppisiä.

Piirteen tunniste	Kirjoitus-oikeudet	Oletusarvo	Kuvaus
use-attributes2	vain luku	ei olemassa	Palauttaa “tosi”, jos attribuuttioliot, jotka välitetään ContentHandler.startElement()-metodille tässä jäsentimessä toteuttavat org.xml.sax.ext.Attributes2-rajapinnan.Tämä rajapinta paljastaa lisää DTD-tietoa, kuten oliko attribuuti määritelty lähdetekstissä, vai oliko sille asetettu oletusarvo.
use-locator2	vain luku	ei olemassa	Palauttaa arvon tosi, jos jäsentimen ContentHandler.setDocumentLocator()-metodin välittämät Locator-olioit toteuttavat org.xml.sax.ext.Locator2-rajapinnan. Tämä rajapinta paljastaa lisää entiteetti-informaatiota, kuten merkkien koodauksen ja käytetyn XML-version.
use-entity-resolver2	luku/kirjoitus	tosi	Palauttaa “tosi” tilanteessa, jossa setEntityResolver-metodille annetaan olio, joka toteuttaa org.xml.sax.ext.EntityResolver2-rajapinnan, ja noita uusia metodeita tullaan käyttämään. Palauttaa ”epätosi” ilmoittaakseen, että noita metodeja ei käytetä.
validation	luku/kirjoitus	ei määritelty	Kontrolloi, raportoiko jäsennin kaikki validisuusvirheet. Jos tosi, kaikki ulkoiset entiteetit tullaan lukemaan.
xmlns-uris	luku/kirjoitus	epätosi	Kun namespace-prefixes-piirre on määritelty todeksi, jäsennin käsittelee nimiavaruusesittely-attribuutteja kuten ne olisivat http://www.w3.org/2000/xmlns/-nimiavaruudessa. Oletusarvolta SAX2 tukeutuu alkuperäiseen ”Nimiavaruudet XML:ssä”-suositukseen, joka eksplisiittisesti asettaa, että tällaiset attribuutit eivät ole missään nimiavaruudessa. Laittamalla tämän vaihtoehtoisen lipun todeksi saadaan SAX2-tapahtumat tukemaan myöhempää, taaksepäin yhteensopimatonta revisiota tuosta suosituksisesta asettamalla nuo arvot nimiavaruuteen.
xml-1.1	*vain luku*	ei sovellettavissa	Palauttaa toden, jos jäsennin tukee sekä XML 1.1 ja XML 1.0-määrityksiä. Palauttaa epätoden, jos jäsennin tukee vain XML 1.0:aa.

Taulukko 2. SAX2-rajapinnan ominaisuudet [8].

Ominaisuuden tunniste	Kuvaus
declaration-handler	Käytetään tutkimaan useimpia DTD-esittelyjä, paitsi niitä, joita käsitellään leksikaalisina, tai niitä, jotka ovat pakollisia kaikille SAX-jäsentimille (DTDHandler). Olion täytyy toteuttaa org.xml.sax.ext.DeclHandler.
document-xml-version	Voidaan tutkia vain jäsentämisen aikana, sen jälkeen, kun startDocument()-takaisinkutsu on suoritettu. Tämä on vain luku-ominaisuus. Tämä ominaisuus on merkkijono, joka kuvaa dokumentin XML-version, kuten “1.0”, tai “1.1”.
dom-node	Tämä ominaisuus on olemassa “DOM Walker”-tyylisiä jäsentimiä varten, jotka sivuuttavat niiden parser.parse()-metodin parametrit. Tätä on käytetty kuvaamaan DOM-(ali)puu, jota jäsennin kulkee. Olion täytyy toteuttaa org.w3c.dom.Node-rajapinta.
lexical-handler	Käytetään seuraamaan joitakin syntaksitapahtumia, jotka ovat oleellisia joissakin sovelluksissa: kommentit, CDATA-osuuksien erotinmerkit, valitut yleiset entiteettit, DTD:n alku ja loppu (ja dokumenttielementin nimen esittely). Olion täytyy toteuttaa org.xml.sax.ext.LexicalHandler.
xml-string	Luettavissa vain jäsennystapahtuman aikana. Tämä paljastaa TBS-joukon merkkejä, jotka ovat vastuussa kyseisestä tapahtumasta.

3.2. DOM-rajapinta

Document Object Model (DOM) on kieli-ja alustariippumaton rajapinta, joka sallii ohjelmille ja skripteille dynaamisen pääsyn sisältöön, rakenteeseen ja dokumentin tyyliin. Dokumenttia voidaan prosessoida edelleen ja sen prosessoinnin tulokset tuoda takaisin esiteltävälle sivulle [4].

DOM on abstrakti tietorakenne, joka esittää XML-dokumentit puurakenteena. Useat rajapinnat org.w3c.dom-paketissa esittelevät elementit, attribuutit, jäsennetyn merkkitiedon, kommentit sekä prosessointiohjeet. Kaikki nämä ovat Node-rajapinnan alirajapintoja, joka tarjoaa metodeja puun navigoimiseen [1].

Puurakenteen juuri on dokumentti-olio, joka esittää kokonaista, oikein muodostettua dokumenttia. Jäsennin lukee XML-dokumenttia tietovirrasta ja rakentaa dokumentti-olion. Asiakasohjelma kutsuu dokumentti-olion metodeja ja muita DOM-rajapinnan rajapintoja navigoidakseen puuta ja tuodakseen tarvitsemaansa tietoa dokumentti-oliosta. Ohjelmat voivat myös käsitellä muistissa olevaa puuta lisätäkseen, poistaakseen, liikuttaakseen tai muuttaakseen puun solmuja [1].

DOM on kuvattu *IDL*:n (Interface Definition Language), ja sen takia se on kieliriippumaton. DOM on olemassa useille ohjelmointikielille, kuten Java, JavaScript, C++, Python ja Perl [1].

3.2.1. DOM-rajapinnan hyvät ja huonot puolet

Kuten SAX-rajapinnalla, DOM-rajapinnallakin on hyvät ja huonot puolensa. DOM-rajapinta tarjoaa seuraavia hyötyjä [2]:

1. Kaksisuuntainen navigointi

Koska koko dokumentti tallennetaan muistiin ja järjestellään puurakenteeseen, on mahdollista navigoida dokumenttipuuta sekä eteen, että taaksepäin. SAX-rajapinnan kanssa tämä ei ollut mahdollista, SAX-rajapinta salli vain eteenpäin siirtymisen.

2. Tiedon muuntaminen

Samalla kun käydään läpi solmuja puusta, joka on tallennettu muistiin, voidaan helposti muuttaa elementtien dataa. Data, jota muunnetaan, tulee olemaan muistissa oleva kopio dokumentista. Myöhemmin voidaan sarjallistaa muunnettu data XML-tiedostoksi. Tämäkään ei ole mahdollista SAX-rajapinnan kanssa.

3. Rakenteen luominen ja muuttaminen

On mahdollista lisätä tai poistaa solmuja, jotka sisältävät elementtejä, attribuutteja, jne. muistissa olevasta puusta. On mahdollista jopa luoda uusi itsenäinen puurakenne. Sen jälkeen kun puu on rakennettu muistiin, se voidaan tallentaa ulkoiseen tiedostoon XML-dokumentiksi.

4. XML-dokumentin luominen

Datan muokkaamisen jälkeen olemassa olevasta puusta, tai uuden puurakenteen luomisen jälkeen, muistin sisältö voidaan helposti sarjallistaa XML-dokumentiksi.

5. Osittaisen puun manipulointi

On mahdollista manipuloida vain osaa puusta, ja sarjallistaa se ulkoiseen XML-tiedostoon, luomaan alijoukko suuresta XML-dokumentista.

DOM-rajapinnan huonot puolet [2]:

1. Resurssien käyttö

DOM käyttää paljon järjestelmän resursseja. Koska koko dokumentti luetaan muistiin, se voi viedä suuren osan muistista, erityisesti kun suuria dokumentteja jäsennetään käyttämällä DOM-rajapintaa. Tämän takia, DOM-rajapinnan käyttämistä suurien dokumenttien jäsentämiseen ei suositella. Sen, mikä on liian suurta määrittelee systeemin resurssit.

2. Suuri prosessointiaika ja prosessoritehonkulutus

Koko dokumentin rakentaminen muistiin voi vaatia paljon prosessointiaikaa, sekä prosessoritehoa. SAX on nopeampi, koska se prosessoi dokumenttia samalla, kun se lukee sitä. Kun käytetään DOM-rajapintaa, dokumenttia ei voida prosessoida, ennen kuin koko puurakenne on luettu muistiin.

3. Ei sovellu suurille dokumenteille

Edellä mainittujen ominaisuuksien takia DOM ei sovellu suurien dokumenttien käsittelemiseen. Suurten dokumenttien käsittelemiseen SAX-rajapinta tarjoaa paremman vaihdotehdon.

3.2.2. XML-Dokumentin jäsentäminen DOM-jäsentimellä

Toisin kuin SAX, DOM ei sisällä luokkaa tai rajapintaa joka esittää XML-jäsennintä. Kukin jäsennin sisältää oman luokkansa jäsentämistä varten [1].

Koska nämä luokat eivät jaa yhteistä rajapintaa tai superluokkaa, jäsentämismetodit vaihtelevat myös. Esimerkiksi Xerces tarjoaa 2 metodia, jotka jäsentävät XML-dokumentin [1]:

```
public void parse(InputSource source)
    throws SAXException, IOException;
public void parse(String systemID)
    throws SAXException, IOException;
```

Kuva 18. Xerces-jäsentimen parse()-metodit [1].

Saadakseen dokumenttiolion ilmentymän jäsentimeltä täytyy ensiksi käyttää toista esimerkin parse()-metodeista dokumentille ja sitten hakea dokumentti getDocument()-metodilla [1].

Jos jäsennin on Xercesin DOMParser-olio, esimerkin mukaisesti voidaan ladata onnistuneesti DOM Core 2.0-määrittely DOM dokumenttiolioon nimeltä spec [1]:

```
parser.parse("http://www.w3.org/TR/DOM-Level-2-Core");
Document spec = parser.getDocument();
```

Crimson-jäsentimellä sen sijaan parse()-metodi palauttaa dokumenttiolion suoraan, jolloin ei tarvita erillistä getDocument()-metodia [1]:

```
Document spec = parser.parse("http://www.w3.org/TR/DOM-Level-2-Core");
```

Lisäksi Crimson-jäsentimen parse()-metodi on viidellä tavalla ylikuormitettu kahden sijaan. Metodit on lueteltu kuvassa 19.

```
public Document parse(InputSource source)
    throws SAXException, IOException;
public Document parse(String uri)
    throws SAXException, IOException;
public Document parse(File file)
    throws SAXException, IOException;
public Document parse(InputStream in)
    throws SAXException, IOException;
public Document parse(InputStream in, String systemID)
    throws SAXException, IOException;
```

Kuva 19. Crimson-jäsentimen parse-metodit [1].

Kuva 20 on yksinkertainen esimerkkiohjelma, joka käyttää Xerces-jäsennintä tarkistamaan onko dokumentti oikein muodostettu.

```
import org.apache.xerces.parsers.DOMParser;
import org.xml.sax.SAXException;
import java.io.IOException;


public class XercesChecker {

  public static void main(String[] args) {

    if (args.length <= 0) {
      System.out.println("Usage: java XercesChecker URL");
      return;
    }
    String document = args[0];

    DOMParser parser = new DOMParser();
    try {
      parser.parse(document);
      System.out.println(document + " is well-formed.");
    }
    catch (SAXException e) {
      System.out.println(document + " is not well-formed.");
    }
    catch (IOException e) {
      System.out.println(
       "Due to an IOException, the parser could not check "
       + document
      );
    }

  }
}
```

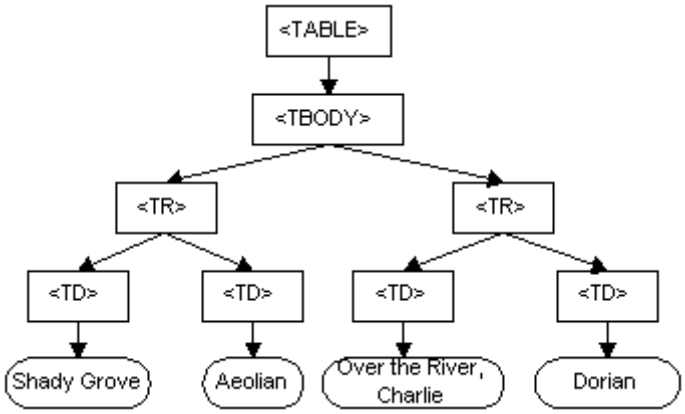
Kuva 20. Esimerkkiohjelma, joka tarkistaa onko dokumentti oikein muodostettu [1].

3.3. DOM-puu

DOM on ohjelmointirajapinta dokumenteille. Se perustuu objektirakenteeseen, joka läheisesti muistuttaa dokumentin rakennetta, jonka se mallintaa [4]. Kuvassa 22 on tehty DOM-puu kuvan 21 HTML – taulukosta.

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

Kuva 21. HTML-taulukko [4].



Kuva 22. Graafinen esitys DOM-puusta [4].

DOM-esitysmuodossa dokumenteilla on looginen rakenne, joka muistuttaa hyvin paljon metsää, joka voi sisältää usean puun. Jokainen dokumentti sisältää ei yhtään tai yhden doctype-solmun, yhden juurielementtisolmun, ja ei yhtään tai useamman kommentin tai prosessointiohjeen. Juurielementti toimii elementtipuun juurena koko dokumentille. DOM ei kuitenkaan määrittele, että dokumentit tulee toteuttaa puumallisesti, eikä se määrittele, miten suhteet objektien välillä tulisi toteuttaa. DOM on looginen malli, joka voidaan toteuttaa millä tahansa sopivalla tavalla [4].

DOM määrittelee myös käsitteen rakenteellinen isomorfismi: jos mitä tahansa kahta DOM-toteutusta käytetään esittämään samaa dokumenttia, ne tekevät saman rakenteellisen mallin [4].

Dokumentit mallinnetaan käyttämällä objekteja, ja malli sisältää myös ei vain dokumentin rakenteen, vaan myös dokumentin ja objektien käyttäytymisen, joista se koostuu. Toisin sanoen solmut eivät esitä vain tietorakennetta, vaan ne esittävät objekteja, joilla on funktioita ja oma identiteetti. Objektimallina DOM sisältää [4]:

- rajapinnat ja objektit, joita käytetään esittämään ja käsittelemään dokumenttia
- näiden objektien ja rajapintojen semantiikan, sisältäen attribuutit ja käyttäytymisen
- objektien ja rajapintojen väliset suhteet sekä vuorovaikutuksen.

3.2.3 DOM-puun osat

DOM-määrityksen mukaan XML-dokumentti on puu, joka koostuu usean tyyppisistä solmuista. Puulla on yksi juurisolmu, ja kaikkilla solmuilla tässä puussa, paitsi juurisolmulla, on yksi vanhempisolmu. Lisäksi kaikilla solmuilla on lista lapsisolmuja. Joissakin tapauksissa tämä lista voi olla tyhjä, jolloin solmua kutsutaan lehtisolmuksi [1]. Voi olla olemassa myös solmuja, jotka eivät ole osa puurakennetta. Esimerkiksi kaikki attribuuttisolmut kuuluvat yhteen elementtisolmuun, mutta sitä ei voida pitää sen lapsisolmuna [1].

Jokaisella solmulla on lisäksi paikallinen nimi, nimiavaruus-URI ja etuliite. Usean tyyppisille solmuille nämä arvot voivat olla määrittelemättömiä. Esimerkiksi kommenttisolmulle nämä ovat aina määrittelemättömiä (null). Kaikilla solmuilla on myös solmunimi. Attribuutille tai elementille solmun nimi on etuliitteellinen nimi. Nimetyille asioille, kuten notaatiolle tai entiteetille, solmun nimi on se, miksi nämä on määritelty. Solmuille, joille ei ole nimeä, kuten tekstisolmuille, solmun nimi on arvo seuraavasta listasta, joka vastaa solmun tyyppiä [1]:

#document
#comment
#text
#cdata-section
#document-fragment

Kaikilla solmuilla on myös merkkijonoarvo. Tekstityylisille asioille, kuten tekstisolmu ja kommentit, tämä on solmun sisältämä teksti. Attribuuteille se on attribuutin arvon normalisoitu muoto. Kaikille muille, sisältäen elementit ja dokumentit, arvo on määrittelemätön [1] .

DOM jakaa solmut kahteentoista tyyppiin, joista seitsemän voi potentiaalisesti olla osa DOM-puuta [1]:

- dokumenttisolmut
- elementtisolmut
- tekstisolmut
- attribuuttisolmut
- prosessointiohjesolmut
- kommenttisolmut
- dokumentin tyyppi-solmut
- dokumentin osasolmut
- notaatiosolmut
- CDATA-solmut
- entiteettisolmut
- entiteettiviittaussolmut

Näistä kahdestatoista ensimmäiset seitsemän ovat kaikkein tärkeimpiä, ja usein XML-jäsentimen rakentama puu sisältää vain näitä seitsemää solmua [1].

Dokumenttisolmu. Kaikilla DOM-puilla on yksi juuridokumenttisolmu. Tällä solmulla on lapsisolmuja. Koska kaikilla dokumenteilla on tasan yksi vanhempi, dokumenttisolmulla on aina yksi elementtilapsisolmu [1].

Elementtisolmu. Kaikilla elementtisolmuilla on nimi, paikallinen nimi, nimiavaruus-URI ja etuliite [1]. Kuvassa 23 on esitelty esimerkki elementtisolmusta.

```
<db:para xmlns:db="http://www.example.com/"
  xmlns="http://namespaces.cafeconleche.org/">
  Or consider this <markup>para</markup> element:
</db:para>
```

Kuva 23. Elementtisolmu [1].

Kuvan 23 elementtisolmun nimi on db:para, ja paikallinen nimi para. Elementtisolmun etuliite on db, ja nimiavaruus-URI on http://www.example.com. Sillä on kolme lasta [1]:

1. Tekstisolmu, joka sisältää tekstin ”Or consider this”.
2. Elementtisolmun, jonka nimi on markup, paikallinen nimi on markup, nimiavaruus-URI http://namespaces.cafeconleche.org, ja ei etuliitettä.
3. Toinen tekstisolmu, joka sisältää tekstin ”element:”.

Attribuuttisolmuilla on nimi, paikallinen nimi, etuliite, nimiavaruus-URI ja merkkijonoarvo. Attribuutteja ei pidetä lapsina elementille, johon ne liittyvät. Sen sijaan ne ovat osa erillistä solmujen joukkoa [1].

Tekstisolmut sisältävät merkkijonotietoa dokumentista [1].

Kommenttisolmulla on nimi (joka on aina #comment), merkkijonoarvo (joka on kommentin sisältämä teksti), ja vanhempi, jonka lapsi kommenttisolmu on [1].

Prosessointiohjesolmulla on nimi (prosessointiohjeen kohde), merkkijonoarvo (prosessointiohjeen tiedot) ja vanhempi, jonka lapsi prosessointiohjesolmu on [1].

CDATA-osuussolmu on erityinen tekstisolmu, joka esittää CDATA-osuuden sisällön. Sen nimi on #cdata-section, ja sen arvo on osuuden tekstisisältö [1].

Entiteettiviittaussolmut. Kun jäsennin törmää yleiseen entiteettiviittaukseen, kuten Æ tai &copyright_notice; se voi korvata tai olla korvaamatta entiteetin korvaavalla tekstillä. Validoivat jäsentimet aina korvaavat entiteettiviittaukset. Validoimattomat jäsentimet voivat tehdä niin halutessaan [1].

Jos jäsennin ei korvaa entiteettiviittauksia, niin DOM-puu sisältää entiteettiviittaussolmuja. Kaikilla entiteettiviittaussolmuilla on nimi, ja jos jäsennin on lukenut DTD:n, niin dokumentintyyppisolmusta pitäisi löytyä entiteettisolmujen kuvaus, joka sisältää julkiset ja järjestelmätunnisteet entiteettiviittaukselle [1].

Dokumentintyyppisolmulla on nimi, julkinen ID (ei pakollinen), pakollinen systeemi-ID, sisäinen DTD (ei pakollinen), vanhempi, sekä lista notaatioista ja yleisistä entiteeteistä, jotka on määritelty DTD:ssä. Dokumentintyyppisolmun merkkijonoarvo on aina null [1].

Entiteettisolmut esittävät jäsennettyjä ja jäsentämättömiä entiteettisolmuja, jotka on esitelty dokumentin DTD:ssä. Jos jäsennin lukee DTD:n, se liittää kuvauksen entiteettisolmuista dokumentintyyppisolmuun. Koska tämä kuvaus indeksoidaan entiteettien nimien mukaan, sitä voidaan käyttää sovittamaan entiteettiviittaussolmut entiteettisolmuihin [1].

Kaikilla entiteettisolmuilla on nimi ja järjestelmätunniste. Sillä voi myös olla julkinen tunniste, jos DTD:ssä käytettiin sellaista. Lisäksi, jos jäsennin lukee entiteetin, entiteetillä on lista solmuja, jotka sisältävät entiteetin korvaavan tekstin [1].

Notaatiosolmut esittävät notaatioita, jotka on esitelty dokumentin DTD:ssä. Jos jäsennin lukee DTD:n, sen täytyy liittää kartoitus notaatiosolmuista dokumentintyyppisolmuun. Tämä kartoitus indeksoidaan notaatioiden nimien mukaan. Sitä voidaan käyttää etsimään notaatio jokaiselle entiteettisolmulle, joka vastaa jäsentämätöntä entiteettiä tai notaatioita, jotka liittyvät tiettyyn prosessointiohjeeseen [1].

Nimen lisäksi kaikilla notaatiosolmuilla on julkinen tunniste tai järjestelmätunniste riippuen siitä, kumpaa DTD:ssä käytettiin määrittelyssä. Notaatiolla ei ole lapsia [1].

Dokumentin osasolmu on vaihtoehtoinen juurisolmu DOM-puulle. Se voi sisältää kaikkea, mitä elementti voi sisältää. Vaikka jäsennin ei koskaan luo tällaista solmua, muut sovellukset voivat luoda niitä ottaessaan osan dokumentista talteen viedäkseen sen muualle [1]. Taulukkoon 3 on koottu solmujen ominaisuudet [1].

Talukko 3. Solmujen ominaisuudet [1].

Solmun tyyppi	Nimi	Arvo	Vanhempi	Lapset
Dokumentti	#document	null	null	Kommentti, Prosessointiohje, 1 Elementti.
Dokumentin tyyppi	DOCTYPE-määritelmän esittelemä Juurielementin nimi	null	Dokumentti	ei ole.
Elementti	etuliitteellinen nimi	null	Elementti, Dokumentti tai Dokumentin osa.	Kommentti, Prosessointiohje, Teksti, Elementti, Entiteettiviittaus, ja CDATA-osio
Teksti	#text	solmun teksti	Elementti, Attribuutti, Entiteetti tai Entiteettiviittaus	ei ole
Attribuutti	etuliitteellinen nimi	attribuutin normalisoitu muoto	Elementti	Teksti, Entiteettiviittaus
Kommentti	#comment	kommentin tekstisisältö	Elementti, Dokumentti, tai Dokumentin osa.	ei ole
Prosessointiohje	kohde	tiedot	Elementti, Dokumentti, tai Dokumentin osa.	ei ole
Entiteettiviittaus	nimi	null	Elementti tai Dokumentin osa	Kommentti, Prosessointiohje, Teksti, Elementti, Entiteettiviittaus, CDATA-osio.
Entiteetti	entiteetin nimi	null	null	Kommentti, Prosessointiohje, Teksti, Elementti, Entiteettiviittaus, CDATA-osio.
CDATA-osio	#cdata-section	osion tekstisisältö	Elementti, Entiteetti tai Entiteettiviittaus	ei ole
Notaatio	notaation nimi	null	null	ei ole
Dokumentin osasolmu	#document-fragment	null	null	Kommentti, Prosessointiohje, Teksti, Elementti, Entiteettiviittaus, CDATA-osio.

3.3.1. DOM-rajapinnan kehityshistoria

Ensimmäinen versio DOM:ista, jota kutsutaan myös DOM Level 0:ksi, ei ollut virallinen määrittely, vaan objektimalli, jonka Netscape Navigator 3 ja Internet Explorer 3 toteuttivat [1].

DOM Level 0 toimi vain HTML-dokumenttien kanssa ja JavaScriptin kanssa. JavaScriptin hyödyllisyys ja näiden kahden selaimen kasvava yhteensopimattomuus vaikutti uuden standardin syntymiseen. W3C aloitti työskentelyn DOM Level 1:n parissa. DOM1 oli pyrkimys tehdä mahdollisimman nopeasti määrittely, joka yhdistäisi olemassa olevia käytäntöjä, ja saisi myös aikaan jonkintasoista yhteensopivuutta selaimien välillä [1].

DOM Level 2 siivosi DOM1:n rajapintoja. Isoin muutos oli tuki nimiavaruuksille element ja attr-rajapinnoissa. Lisäksi, DOM2 lisäsi useita tukevia rajapintoja event, traversal, range, views, ja stylesheets-rajapinnoille. Vuoteen 2002 mennessä, kaikki merkittävät XML-jäsentimet, jotka tukivat DOM:ia tukivat myös DOM2:ta [1].

DOM Level 3 lisää osia, joita tarvitaan XML:n Information Set-ominaisuuksien tukemiseen. DOM3 lisää myös tärkeitä toiminnallisuuksia, joita DOM2:sta puuttuu. Erityisesti DOM2 ei tarjoa jäsenninriippumatonta tapaa uuden dokumenttiolion luomiselle, jäsentämällä tiedostoa tai rakentamalla uuden muistista. DOM3 tarjoaa standardin tavan molemmille tavoille. Huolimatta muutoksista, DOM3 ei korvaa DOM2:ta. Kaikki, mikä toimii DOM2:lla, toimii edelleen myös DOM3:lla [1].

3.3.2. DOM Level 2 moduulit

DOM2 on jaettu neljääntoista moduuliin kahdeksassa eri paketissa. DOM1 vastaa Core- ja XML-moduuleja. Loput 12 ovat uusia DOM2:ssa [1].

*Core s*isältää perusrajapinnat, joita voidaan käyttää esittämään mitä tahansa SGML-mäistä, hierarkista puurakenteista dokumenttia [1].

*XML s*isältää alirajapintoja Node-rajapinnalle pelkästään XML-dokumenttien käyttöön. HTML DOM ei välttämättä toteuta näitä [1].

HTML sisältää rajapintoja, jotka suunniteltiin esittämään erityisesti HTML-dokumenin osia. Nämä kaikki ovat alirajapintoja Core-rajapinnoille. Nämä eivät ole normaalisti oleellisia XML-dokumenttien käsittelemiseen [1].

Views sisältää rajapinnat, joita käytetään assosioimaan erilaisia näkymiä dokumentille. Esimerkiksi kahden tyylitiedoston käyttö yhdelle dokumentille voisi luoda kaksi näkymää. Suurin osa XML-jäsentimiä ei tue tätä [1].

StyleSheets sisältää perusrajapinnat tyylitiedostojen esittämiseen, sisältäen StyleSheet, StyleSheetList, MediaList, LinkStyle ja DocumentStyle-rajapinnat [1].

CSS sisältää rajapinnat, jotka esittävät CSS-tyylitiedostoja [1].

CSS2 on ominaisuusluokka, joka sallii suorat metodit, joiden avulla voidaan asettaa kaikki mahdolliset CSS2-tyyliominaisuudet [1].

Events-luokan rajapinnat luovat geneerisen järjestelmän, joka sallii tapahtumakuuntelijoiden kiinnityksen solmuihin. Tapahtumasolmut voivat vastata käyttöliittymätapahtumiin, kuten hiiren painalluksiin, rakenteenmuutostapahtumiin, kuten dokumentin editointiin, ja kaikkeen, mitä toteutus haluaa tukea. Seuraavat 4 moduulia määrittelevät tarkat tapahtumat. Sen lisäksi, sovellus voi määritellä myös omia tapahtumiaan. Käytännössä tapahtumat ovat oleellisimpia JavaScriptille ja muille selainpohjaisille skriptijärjestelmille kuin suurimmalle osalle XML-ohjelmia. Ne voivat kuitenkin olla käytännöllisiä, kun XML-dokumentti näytetään selaimessa, tai se esitetään muulla tavalla käyttäjälle [1].

UIEvent-rajapinta ilmoittaa, kun näytöllä esitettävään solmuun jossakin käyttöliittymän kaaviossa kohdistuu fokus, se on hävittänyt fokuksen, tai se on aktivoitu [1].

MouseEvent-rajapinta ilmoittaa milloin, missä, ja mitä nappeja käyttäjä on klikannut hiirellään, painaa nappia, irroittaa napista, liikuttanut hiirtä, tai siirtänyt kursorin yli elementin graafisen esityksen [1].

MutationEvent-rajapinta ilmoittaa, jos solmu on lisätty, poistettu, tai sitä on muutettu [1].

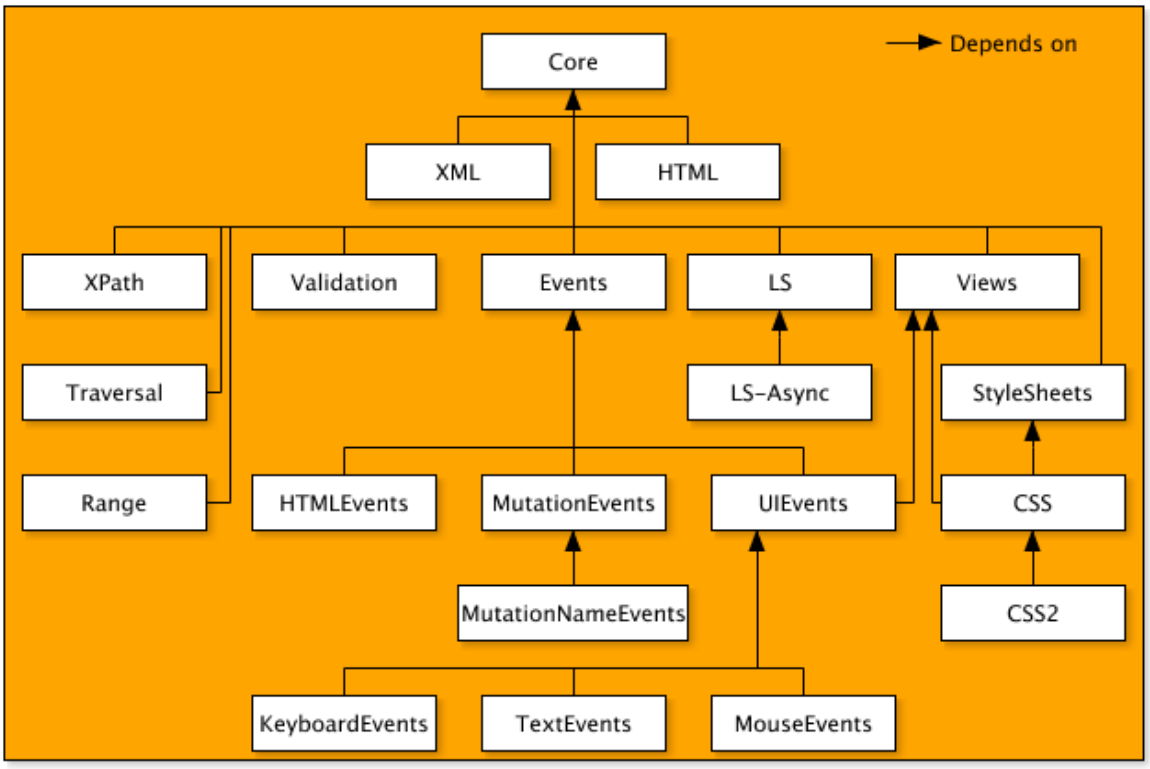
HTMLEvents -moduuli ei määrittele uusia rajapintoja. Sen sijaan se käyttää DOMEvent-rajapintaa raportoimaan useista tapahtumista, jotka ovat selaimille ominaisia [1].

Traversal-paketti tuo käyttöön yksinkertaisia työkaluluokkia operaatioille, joita suoritetaan puulle, kuten koko puun läpikäynti, tai vain sellaisten solmujen valinta, jotka täyttävät tietyt ehdot [1].

Range on vaihtoehtoinen moduuli, joka laajentaa DOM:ia kattamaan alueet dokumenteista, jotka eivät täysin täsmää elementtien rajoja. Esimerkiksi, voi olla käytännöllistä viitata tekstin osuuteen, jonka käyttäjä on valinnut hiirellään [1].

3.3.3. DOM Level 3 moduulit

DOM Level 3 esittelee laajennuksia jo olemassa oleviin moduuleihin, sekä täysin uusia moduuleja. Tässä kohdassa luetellut moduulit ovat uusia moduuleja. Kuvassa 4 on esiteltynä DOM Level 3-moduulit.



Kuva 24. DOM Level 3-moduulit [4].

KeyboardEvent on UIEvents-moduulin laajennus. KeyboardEvent-rajapinta tarjoaa tarkkaa näppäinlaitteisiin liittyvää kontekstuaalista informaatiota [4], kuten tietoa siitä, kun näppäintä painetaan.

TextEvent on myös UIEvent-moduulin laajennus. TextEvent-rajapinta tarjoaa sisällöllistä informaatiota, joka liittyy tekstitapahtumiin [9].

MutationNameEvents on MutationEvents-rajapinnan laajennus. MutationNameEvents-rajapinta tarjoaa tietoa muutoksista elementtisolmujen tai attribuuttisolmujen nimissä [4].

Load And Save (LS)-moduuli sisältää rajapintoja, joiden avulla voidaan ladata tai tallentaa dokumenttiolioita [4].

Asynchronous load module (LS-Async) on Load And Save-moduulin asynkroninen toteutus. Tukeakseen täysin asynkronista toteutusta, jäsentimen täytyy tukea myös LS-moduulia [4].

XPath on kyselykieli, jonka avulla voidaan saada tietoa XML-dokumentin osien sijainnista. Se myös tarjoaa perustoiminnallisuuden merkkijonojen, kokonaislukujen, sekä totuusarvojen käsittelemiseen [4].

Validation-moduuli sisältää rajapintoja, jotka ohjaavat XML-dokumenttien muokkaamista ja editointia. Esimerkki tuollaisesta ohjatusta muokkaamisesta voisi olla kysely, jossa halutaan tietää, mitä skeema antaa lisätä tai poistaa, tai, onko dokumentti lisäyksen jälkeen edelleen validi [4].

4. XML-jäsentimet

XML-dokumentteja käsittelevien sovellusten täytyy pystyä oikealla tavalla tulkitsemaan ja muokkaamaan niitä. Tässä XML-jäsentimet tulevat mukaan toimintaan. XML-jäsennin ottaa vastaan XML-dokumentin syötteenään ja käsittelee sitä tulkitakseen tietoa, jota se sisältää. Jäsennin voi jopa muuntaa dokumentin sisällä olevaa tietoa tai luoda uuden XML-dokumentin toiselle sovellukselle käytettäväksi [2].

4.1. Jäsentämisen tarve

XML-dokumentteja käytetään monissa erilaisissa sovelluksissa. Vaikka näitä dokumentteja voidaan helposti tulkita lukemalla, niiden todellinen käyttötarkoitus on elektronisessa prosessoinnissa ja manipuloinnissa. Jäsennin hoitaa tämän tehtävän, joka on vastuussa dokumentin tutkimisesta, elementtien tunnistamisessa, ja elementtien sisältämän tiedon tuominen esille elektronista käsittelyä varten [2].

4.2. Jäsentimien hyödyt

Jäsennin suojaa asiakassovellusta epäoleellisien yksityiskohtien käsittelystä, kuten [1]:

- dokumentin koodaaminen Unicode-muotoon
- dokumentin kokoamisen hajautetuista entiteeteistä
- merkkiviittausten selvittäminen
- CDATA-osuuksien ymmärtäminen
- satojen oikeinmuodostuneisuusrajoitteiden tarkistaminen
- nimiavaruuslistan ylläpitäminen elementeille
- dokumentin validoinnin sen DTD:tä tai skeemaa vasten
- jäsentämättömien entiteettien liittäminen URL-osoitteesiin ja notaatioihin
- attribuuttien tyypin määritteleminen

4.3. XML-jäsentimen valinta

Kun aloitetaan uusi XML-projekti, ensimmäinen asia on valita siihen sopiva rajapinta. Rajapinnan valinnan jälkeen pitää valita jäsennin, joka parhaiten soveltuu projektin toteuttamiseen. XML-jäsentimet poikkeavat toisistaan ominaisuuksiltaan, tehokkuudessa, tuessa eri rajapinnoille, lisensseissään sekä toteutuksissaan [1].

4.3.1. Ominaisuudet

XML 1.0 määritys antaa jäsentimille vapauksia siinä, miten suurta osaa määrittelyä niiden tulee toteuttaa. Jäsentimet voidaan jakaa karkeasti kolmeen tyyppiin [1]:

1. Täysin validoivat jäsentimet.
2. Jäsentimet, jotka eivät validoi, mutta lukevat ulkoisen DTD:n ja ulkoisen DTD:n parametriset entiteettiviittaukset tukeakseen entiteettien korvausta ja attribuuttien tyyppien määrittelemistä.
3. Jäsentimet, jotka lukevat vain sisäisen DTD:n ja eivät suorita validoimista.

Käytännössä on vielä olemassa neljäs jäsennintyyppi, jotka lukevat dokumentin, mutta eivät suorita kaikkia pakollisia oikeinmuodostuneisuus-tarkistuksia. Teknisesti XML-määrittely ei salli tällaisia jäsentimiä, mutta niitä on silti paljon olemassa [1].

Jos käsiteltävä dokumentti sisältää DTD:n, kannattaa käyttää täysin validoivaa jäsennintä. Validointia ei välttämättä tarvitse kytkeä päälle, jos sitä ei tarvita. Joka tapauksessa XML on suunniteltu siten, että välttämättä ei saada koko dokumentin sisältöä, ilman että sen DTD luetaan. Joissakin tapauksissa, erot samalla dokumentilla, jonka DTD on käsitelty, ja dokumentilla, jonka DTD:tä ei luettu, voivat olla suuria. Esimerkiksi, jäsennin, joka lukee DTD:n, ilmoittaa attribuuttien oletusarvot, mutta sellainen, joka ei lue, ei näitä ilmoita. Vältettävien tyhjätilamerkkien käsittely voi vaihdella validoivalla jäsentimellä ja jäsentimellä, joka vain lukee sisäisen DTD:n ja ei validoi. Lisäksi, ulkoisia entiteettiviittauksia ei välttämättä käsittele jäsennin, joka ei validoi. Validointiin kykenemätöntä jäsennintä kannattaa käyttää vain, jos on varma, että mikään käsiteltävä dokumentti ei sisällä Document Type-esittelyä [1].

Jotkut jäsentimet tarjoavat myös lisätietoa, mitä ei ole vaadittu normaalissa XML-jäsentämisessä. Esimerkiksi Xerces voi tarjota tietoa ELEMENT, ATTLIST, sekä ENTITY-määrityksistä DTD:n sisällä [1].

4.3.2. Tuki Rajapinnoille

Suurimmat jäsentimet tukevat sekä SAX- että DOM-rajapintoja. Jotkut jäsentimet tukevat vain SAX-rajapintaa, ja muutamat tukevat vain omia rajapintojaan [1].

SAX sisältää useita vaihtoehtoisia ominaisuuksia, joita jäsentimien ei ole pakko tukea. Näitä ovat esimerkiksi validointi, kommenttien reportointi, DTD-määrittelyn sisällön raportointi, dokumentin alkuperäisen tekstin raportointi ennen jäsentämistä jne. Jos näitä vaihtoehtoisia ominaisuuksia tarvitaan, tulee valita jäsennin, joka tukee näitä [1].

4.3.3. Lisenssi

Jäsentimet ovat yleensä kohtuullisen vapaiden lisenssien alla. Ostettavia jäsentimiä ei ole kovin paljoa ja ilmaiset jäsentimet ovat yleensä enemmän kuin riittäviä [1]. Tähän tutkimukseen valittujen jäsentimien lisenssi mainitaan, mutta sen sisältöön ei perehdytä tarkemmin

4.3.4. Virheettömyys

Monesti ylenkatsottu kriteeri jäsentimen valinnalle on sen virheettömyys: kuinka paljon oleellisista ominaisuuksista on toteutettu, ja kuinka hyvin. Vaikka mikään jäsennin ei ole täydellinen, jotkut jäsentimet ovat luotettavampia kuin toiset. Yleisimmät virheet toteutuksissa ovat [1]:

1. Oikeiden rakenteiden raportointi virheinä.
2. Jäsennin ei tunnista väärää syntaksia.

On vaikea sanoa, kumpi on pahempi virhe. Oikein muodostettujen dokumenttien hylkääminen virheellisenä estää toisilta osapuolilta saatujen dokumenttien käsittelyn. Toisaalta, jos jäsennin ei osaa ilmoittaa virheistä dokumentissa, se voi aiheuttaa ongelmia osapuolille, jotka ottavat vastaan ja käsittelevät luotuja dokumentteja [1].

Validisuusvirheiden tunnistamiseen liittyvät ongelmat eivät ole yhtä kriittisiä, mutta ne ovat silti merkittäviä ongelmia. Koska monet ohjelmat voivat sivuuttaa validisuusvirheet ja validaattoreiden viallisesta toiminnasta johtuvat virheet, tämä ongelma ei ole yhtä merkittävä [1].

4.3.5. Tehokkuus

Jäsentimet eroavat toteutukseltaan myös järjestelmäresurssien käytön osalta. SAX-jäsentimet ovat erittäin tehokkaita jäsentämisessä ajankäytön ja muistinkulutuksen osalta, kun taas DOM-jäsentimiä käytettäessä järjestelmämuisti saattaa tulla vastaan dokumentin puun luomisessa. Eri toteutusten välillä voi olla suuriakin eroja muistinkulutuksen osalta. DOM-jäsentimet ovat myös jäsentämisnopeudessa SAX-jäsentimiä heikompia.

Jäsentämisnopeuteen vaikuttaa myös, suoritetaanko dokumentille validointi. Kaikilla jäsentimillä, jotka valittiin tutkimukseen, validoinnin suorittaminen hidasti jäsentimen toimintaa. SAX-jäsentimillä nämä erot eivät ole kovin suuria, kun taas DOM-jäsentimien suorittama validointi voi olla erittäin hidasta.

4.4. Tutkimukseen valitut jäsentimet

Tutkimukseen valitut jäsentimet ovat yleisesti käytössä olevia ja tunnettuja jäsentimiä. SAX-rajapinnan toteuttavia jäsentimiä on olemassa paljon, ja tutkimukseen otettiin mukaan näistä tunnetuimpia. Tähän tutkimukseen valitut jäsentimet ovat ilmaiseksi saatavilla niiden valmistajilta.

Xerces2 on Apachen kehittämä jäsennin, joka tukee erinomaisesti sekä SAX-että DOM-rajapintoja. Xerces2 tukee myös JAXP-rajapintaa. Xerces2 on täysin validoiva jäsennin, joka tukee validointia myös skeeman mukaisesti [10]. Xerces2-kirjastot tulevat JDK1.5:n mukana. Jäsentimen lisenssi on Apache Software Licence. Xerces2 on saatavilla JAR-pakettina valmistajan sivuilta, sekä lähdekoodina.

Crimson on XML-jäsennin Javalle, joka tukee XML1.0:aa. Crimson tukee SAX,DOM ja JAXP-rajapintaa [11]. Crimson on myös Apachen kehittämä jäsennin. Crimson pystyy myös

validoimaan dokumentin. Jäsentimen lisenssi on Apache Software Licence. Crimson perustuu Sun Project X-jäsentimeen, ja se tulee Sunin tuotteiden mukana.

Piccolo on pieni, erittäin nopea XML-jäsennin Javalle. Se toteuttaa SAX 1, SAX 2.0.1, ja JAXP 1.1-rajapinnat. Jäsennin pyrkii huomaamaan kaikki oikeinmuodostuneisuusvirheet, mutta ei suorita validointia [12]. Piccolon kehitti Yuval Oren. Valmistajan sivuilta saa ladattua valmiin JAR-paketin, sekä lähdekoodit. Jäsentimen lisenssi on Apache Software Licence 2.0.

Gnu Jaxp, joka oli alun perin osa Gnu Classpath Extensions-projektia, on nykyään osa Gnu Classpath-projektia, ja on ilmainen toteutus standardeista XML-jäsentimien rajapinnoista Javalle [13].

Gnu Jaxp-paketti sisältää Aelfred2 SAX2-jäsentimen, joka on konfiguroitu käyttämään vaihtoehtoista validointimoduuliaan oletusasetuksena [13]. Se sisältää myös toteutuksen W3C:n DOM3-rajapinnoista. Gnu Jaxp tukee lisäksi JAXP 1.3-rajapintaa.

Gnu Jaxp-paketti täytyy itse kääntää, ja valmistajan sivuilta saa ladattua lähdekoodit, sekä kääntämistä varten tarvittavat skrpitit. Gnu Jaxp-paketin DOM-jäsennin ei osaa vielä tutkimuksen kirjoittamisen aikana suorittaa validointia. Paketin lisenssi on GNU General Public License.

Aelfred on SAX ja JAXP-rajapintoja tukeva XML-jäsennin. Sillä ei ole DOM-tukea, ja se ei tue validointia. Se on niin riisuttu XML-jäsennin kuin mahdollista. Jäsentimen lisenssi on Microstar's original terms and conditions. Valmistajan sivuilta saa valmiin JAR-paketin, sekä lähdekoodit. Tämä versio on haarautunut David Brownellin kehittämästä versiosta, joka sisältyy GNU JAXP-pakettiin [14].

4.5. Jäsentimien tuki DOM-rajapinnan moduuleille

Taulukkoon 4 on koottu jäsentimien tuki DOM level 2-moduuleille ja taulukkoon 5 DOM level 3 – moduuleille. Jäsentimien tarjoamaa tukea DOM-rajapinnan moduuleille tutkittiin pyytämällä jäsentimeltä sen tukemat moduulit ohjelmallisesti. DOM-rajapinta tarjoaa tähän

oman luokkansa ja metodin, johon syötteenä annetaan halutun moduulin nimi sekä versionumero merkkijonona. Nämä tunnisteet on esillä W3C:n sivuilla [4], josta ne on tutkimukseen haettu. Gnu Jaxp-projektin [13] kotisivuilla mainitaan DOM Level 2-moduuleille vähemmän tuettuja moduuleja kuin mitä tähän kyselyyn tarkoitettu metodi palauttaa. Tämä metodi ei GNU JAXP-paketin osalta anna mitään tietoja DOM3-moduuleiden tuesta, mutta tuetut moduulit on mainittu projektin internetsivuilla. Gnu Jaxp-paketin osalta DOM2-tason moduulit on haettu ohjelmallisesti, ja tuetut DOM3-tason moduulit on katsottu projektin kotisivuilta.

Taulukko 4. Jäsentimien tuki DOM Level 2-moduuleille.

	Xerces	Crimson	Gnu
Core	X	X	X
XML	X	X	X
HTML			
Views			
Stylesheets			
CSS			
CSS2			
Events	X		X
UIEvents			X
MouseEvents			
MutationEvents	X		X
HTMLEvents			X
Traversal	X		X
Range	X		

Taulukko 5. Jäsentimien tuki DOM Level 3-moduuleille.

	Xerces	Crimson	Gnu
Core	X		X
XML	X		
Events			
UIEvents			
MouseEvents			
TextEvents			
KeyboardEvents			
MutationEvents			
MutationNameEvents			
HTMLEvents			
Load and save	X		X
LS-Async			
Xpath			X
Validation			

4.6. Jäsentimien tuki SAX2-rajapinnan piirteille ja ominaisuuksille

Jäsentimien tunnistamat SAX2-rajapinnan ominaisuudet ja piirteet on esiteltynä taulukossa 6. Jäsentimien tunnistamat piirteet on selvitetty ohjelmallisesti pyytämällä tietoa jäsentimeltä tutkimuksessa käytetyn mittausohjelmistaon avulla.

Taulukko 6. Jäsentimien tunnistamat SAX2-rajapinnan ominaisuudet ja piirteet.

	Crimson	Xerces	Aelfred	Piccolo	Aelfred2
external-general-entities	X	X	X	X	X
external-parameter-entities	X	X	X	X	X
is-standalone		X		X	X
parameter-entities	X	X		X	X
namespaces	X	X	X	X	X
namespace-prefixes	X	X	X	X	X
resolve-dtd-uris		X		X	X
string-interning	X	X	X	X	X
unicode-normalization checking		X			
use-attributes2		X		X	X
use-locator2		X		X	
use-entity-resolver2		X			X
validation	X	X	X	X	X
xmlns-uris		X			
xml-1.1		X			
declaration-handler	X	X	X	X	X
document-xml-version		X			
dom-node		X			
lexical-handler	X	X	X	X	X
xml-string		X			

5. Jäsentimien tehokkuuden mittaus

Tutkimuksen mittauksissa tarkastellaan jäsentimien tehokkuutta tarkastelemalla muistin kulutusta sekä jäsentämiseen käytettyä aikaa. Lähdemateriaalina käytetään ohjelmallisesti generoitua XML-dataa. XML-data on generoitu Python-skriptillä, jonka lähdekoodi on liitteessä 1. XML-tiedosto sisältää katalogin, johon kuuluu useita kirjaelementtejä.

Tiedosto sisältää myös sisäisen DTD:n, jonka rakenne on kuten kuvassa 25.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE catalog[
<!ELEMENT catalog (book+)>
<!ELEMENT book (title,authors,publisher,year,listprice,isbn)>
<!ATTLIST book cover (hard|paper) "hard">
<!ELEMENT title (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT author (last,first)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT listprice (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ATTLIST isbn id ID #REQUIRED>
]>
```

Kuva 25. Dokumenttien sisäinen DTD.

Esitelty DTD määrää, että catalog-juurielementtiin tulee kuulua ainakin yksi kirjaelementti. Kirjaelementille on löydyttävä teoksen nimi, yksi tai useampi tekijä, julkaisija, julkaisuvuosi, hinta, sekä sen yksilöivä ISBN-tunnus. Dokumentin validointi suoritetaan tämän määrityksen mukaisesti.

DTD:n mukainen tieto ilmenee XML-tiedostossa kuvan 26 mukaisesti.

```
<catalog>
<book cover="hard">
<title>book 0</title>
<authors>
    <author>
        <last> lastname </last>
        <first> firstname </first>
    </author>
</authors>
<publisher> PUBLISHER </publisher>
<year>0</year>
<listprice>100</listprice>
<isbn id = "book-0"></isbn>
</book>
</catalog>
```

Kuva 26. Kirja-elementti sisältöineen.

5.1. Tutkimuksessa käsiteltävät tiedostot

Käsiteltävät tiedostot vaihtelevat kooltaan. Erikokoisilla tiedostoilla ajettaessa saadaan tietää, vaihteleeko jäsentimien toimintatehokkuus erikokoisilla tiedostoilla, ja missä vaiheessa tiedoston koko alkaa vaikuttaa oleellisesti jäsentämiseen, ja miten suuria tiedostoja jäsentimet pystyvät käsittelemään. Käsiteltävät tiedostot sisältävät 10 (4kt), 100 (27kt), 1000(259kt), 10 000(2,615kt) tai 100 000(26,433kt) kirjaelementtiä.

Nopeutta mitatessa 10-1000 kappaletta kirjaelementtejä sisältävät tiedostot käsitellään 1000 kertaa, ja näistä kerroista lasketaan keskiarvo käytetylle ajalle. 10 000 kappaletta kirjaelementtejä sisältävä tiedosto jäsennetään 100 kertaa, ja 100 000 kappaletta kirjaelementtejä sisältävä tiedosto 10 kertaa.

Pienimmät tiedostot jäsennetään useammin kuin isommat, koska järjestelmässä tapahtuvat muut prosessit saattavat vaikuttaa jäsentämisnopeuteen. Ohjelmisto tulostaa näytölle käytetyn ajan keskiarvon. Jäsentämiskerrat ovat samat sekä SAX-että DOM-jäsentimiä testatessa. Tulokset eivät ole täsmälleen samoja joka kerta siitä huolimatta, että jäsentämistä toistetaan

useasti, vaan saattavat heitellä joitakin sadasosia. Tutkimuksessa saadut tulokset kuitenkin tuovat erot selkeästi esille eri jäsenten välillä.

Lähdemateriaali käydään läpi kaikilla jäsentimillä. Kaikki tutkimuksen jäsentimet tukevat ainakin SAX2-rajapintaa ja kykenevät tarkistamaan, että dokumentti on oikein muodostettu. Jäsentimillä, jotka kykenevät suorittamaan myös dokumentin validoinnin, suoritetaan dokumentin validointi esitellyn DTD:n mukaisesti.

DOM-rajapintaa tukeville jäsentimille testataan myös niiden tehokkuus DOM-olion luomisessa. Tehokkuus tässä tarkoittaa muistinkulutusta sekä ajan käyttöä DOM-olion luomiseen. DOM-jäsentimet suorittavat myös validoinnin dokumentille, mikäli ne tukevat validointia.

Itse testausohjelmisto on toteutettu Java-ohjelmointikielellä. Testausohjelmisto luo jäsentämiseen tarvittavat oliot ja käynnistää jäsentimen sekä hoitaa tulosten tuonnin näytölle. Testausohjelmiston lähdekoodit ovat liitteessä 2.

Piirteisiin, joihin SAX2-rajapinta ei määrittele oletusarvoa, määritellään sama arvo kaikille jäsentimille siltä varalta, että näiden piirteiden eri asetuksilla on vaikutusta tehokkuuteen.

5.2. Laitteisto, tulokset ja mittausmenetelmät

Laitteisto, jolla jäsentimiä testattiin, on seuraavanlainen:

Prosessori: AMD Athlon XP 2600+
Järjestelmämuisti: 1 Gb
Käyttöjärjestelmä: Microsoft Windows XP Service Pack 2
Java Virtuaalikone: Sun Microsystems Inc. 1.5.0
JDK: Java JDK: Java™ 2 Platform Standard Edition Development Kit 5.0

Koska jäsennettävästä tiedostosta muodostetun DOM-olion käyttämä muisti ylittää virtuaalikoneen oletusasetuksien mukaan käytössä olevan järjestelmämuistin, varataan sitä lisää virtuaalikoneesta DOM-jäsentämistä varten. Java-virtuaalikoneen käytössä olevaa

muistia voidaan varata enemmän komentoriviltä -Xmx<size> -optiolla. Jos järjestelmämuisti ei riitä, antaa ohjelma poikkeuksen ja keskeyttää jäsentämistehtävän.

Muistia mitatessa SAX-jäsentimillä saadaan tieto käytetystä muistista tarkastelemalla käytetyn muistin määrää, kun sille rekisteröity tapahtumakuuntelija törmää characters-metodiin, tai EndElement-metodiin. Käytetyn muistin määrä saadaan selville pyytämällä runtime-luokalta käytettävissä olevan muistin määrää ja vähentämällä siitä vapaan muistin määrä. Ohjelmiston tiedostossa MemoryMeasureThread.java tämän suorittaa lause

```
memory=runtime.totalMemory()-runtime.freeMemory();
```

DOM-jäsentimillä muistia mitatessa on käynnistettävä säie jäsentämisen ajaksi, joka seuraa muistin kulutusta samalla, kun jäsennin luo DOM-puuta muistiin. Muistin kulutus saattaa olla puun luomisen aikana isompi, kuin mitä se on sen rakennuttua. Säie ottaa näytteitä muistinkulutuksesta ohjelmistossa määrätyin väliajoin. Lisäksi katsotaan muistinkulutus vielä koko puun ollessa muistissa. Tällä tavalla mitatessa saadaan parempi tulos, kuin katsomalla pelkästään luomisen jälkeen käytössä olevaa muistia, mutta säikeet eivät aina välttämättä suoriudu tasaisesti tutkimuksessa käytetyllä laitteistolla, joten tutkimuksessa on valittu suoritukset, joissa säie on saanut tasaisesti suoriutumisaikaa.

Ajan mittaminen tehdään kysymällä Javan Runtime-luokalta ajanhetkeä ennen jäsentämistä ja ajanhetkeä jäsentämisen jälkeen. System-luokan metodi currentTimeMillis() palauttaa kyseisen ajanhetken, joka vähennetään aloitusajanhetkestä. Tuloksena saadaan aika millisekunteina, joka muunnetaan sekunneiksi jakamalla se tuhannella.

```
currenttime=System.currentTimeMillis()-starttime;
float timeInSecs=(float)currenttime/1000;
```

5.2.1. SAX-jäsentämisnopeus ilman validointia

Taulukossa 7 esitellään SAX-jäsentämisen tulokset ilman validointia. Pienimmillä tiedostoilla jäsennettäessä eroja jäsentimien välillä ei ollut, eroja alkoi näkyä vasta keskisuurilla tiedostoilla. Suurella tiedostokoolla jäsentämistehtävästä nopeiten selvisi Piccolo.

Taulukko 7. SAX-jäsentämisnopeus ilman validointia.

	10 kpl (1000 iteraatiota)	100 kpl(1000 iteraatiota)	1000 kpl (1000 iteraatiota)	10 000 kpl (100 iteraatiota)	100 000kpl (10 iteraatiota)
Crimson	0.016s	0.016s	0.026s	0.257s	2.614s
Xerces	0.016s	0.016s	0.033s	0.309s	3.101s
Piccolo	0.016s	0.016s	0.017s	0.164s	1.615s
Aelfred	0.016s	0.016s	0.027s	0.258s	2.593s
Aelfred2	0.016s	0.016s	0.028s	0.272s	2.745s

5.2.2. SAX-jäsentämisnopeus ja dokumentin validointi

Taulukossa 8 esitellään SAX-jäsentämisen tulokset validoinnin kanssa. Tässäkään tapauksessa pieniä tiedostoja jäsennettäessä ei ollut juurikaan eroja, vaan erot alkoivat näkyä vasta suurilla tiedostoilla. Kaikilla jäsentimillä validointi vaikutti lisäävästi tehtävään käytettyä aikaa.

Taulukko 8. SAX-jäsentämisnopeus ja dokumentin validointi.

	10 kpl (1000 iteraatiota)	100 kpl(1000 iteraatiota)	1000 kpl (1000 iteraatiota)	10 000 kpl (100 iteraatiota)	100 000kpl (10 iteraatiota)
Crimson	0.016s	0.016s	0.030s	0.312s	3.898s
Xerces	0.016s	0.016s	0.036s	0.371s	3.506s
Aelfred2	0.016s	0.016s	0.040s	0.413s	3.912s

5.2.3. SAX-muistinkulutus ilman validointia

Taulukossa 9 esitellään SAX-jäsentimien muistinkulutus ilman validointia. Crimson-jäsennin kulutti kaikkein vähiten muistia. Aelfred-jäsennin rupesi isommilla tiedostoilla kuluttamaan muistia huomattavasti muita enemmän ja erot rupesivat näkymään jo keskisuurissa tiedostoissa selkeästi.

Taulukko 9. SAX-muistinkulutus ilman validointia.

	10 kpl	100 kpl	1000 kpl	10 000 kpl	100 000kpl
Crimson	0.336Mb	0.368Mb	0.686Mb	0.701Mb	0.701Mb
Xerces	0.517Mb	0.571Mb	0.719Mb	0.815Mb	0.815Mb
Piccolo	0.769Mb	0.774Mb	0.872Mb	1.104Mb	1.104Mb
Aelfred	0.495Mb	0.624Mb	1.050Mb	5.033Mb	46.531Mb
Aelfred2	0.290Mb	0.343Mb	0.507Mb	0.788Mb	0.788Mb

5.2.4. SAX-muistinkulutus ja dokumentin validointi

Taulukossa 10 esitellään SAX-jäsentimien käyttämä muisti jäsentämistehtävässä dokumentin validointi valittuna. Selkeästi vähiten muistia vei Crimson, joka isoimmilla tiedostoilla vei noin puolet vähemmän muistia kuin toiseksi tehokkain Aelfred2. Kaikilla jäsentimillä muistinkulutus kasvoi huomattavasti, kun dokumentti validoitiin jäsentämisen aikana. Eniten muistia kulutti Xerces kaikilla tiedostoilla.

Taulukko 10. SAX-muistinkulutus ja dokumentin validointi.

	10 kpl	100 kpl	1000 kpl	10 000 kpl	100 000kpl
Crimson	0.399Mb	0.466Mb	0.671Mb	0.979Mb	4.122Mb
Xerces	0.574Mb	0.638Mb	0.818Mb	2.000Mb	13.734Mb
Aelfred2	0.358Mb	0.426Mb	0.731Mb	1.498Mb	9.447Mb

5.2.5. DOM-puun luomisnopeus ilman validointia

Taulukossa 11 esitellään DOM-puun luomisnopeus ilman dokumentin validointia. Pienillä tiedostoilla eroja jäsentimien välillä ei DOM – toteutuksilla ollut. Vastaaviin SAX-jäsentimiin verrattuna eroja oli pienilläkin tiedostoilla. Suurissa dokumenteissa Xerces oli tehokkain.

Taulukko 11. DOM-puun luomisnopeus ilman validointia.

	10 kpl (1000 iteraatiota)	100 kpl(1000 iteraatiota)	1000 kpl (1000 iteraatiota)	10 000 kpl (100 iteraatiota)	100 000kpl (10 iteraatiota)
Crimson	0.016s	0.020s	0.109s	0.888s	8.437s
Xerces	0.017s	0.021s	0.096s	0.676s	6.245s
Gnu	0.016s	0.020s	0.113s	0.892s	9.095s

5.2.6. DOM-puun luomisnopeus ja dokumentin validointi

Taulukosta 12 ilmenevät DOM-puun luomisnopeudet kahdella eri jäsentimillä validointi asetettuna päälle. Selkeästi tehokkaammaksi osoittautui Xerces. Molemmilla jäsentimillä validointi lisäsi tehtävään käytettyä aikaa.

Taulukko 12. DOM-puun luomisnopeus ja dokumentin validointi.

	10 kpl (1000 iteraatiota)	100 kpl(1000 iteraatiota)	1000 kpl (1000 iteraatiota)	10 000 kpl (100 iteraatiota)	100 000kpl (10 iteraatiota)
Crimson	0.016s	0.016s	0.114s	0.991s	10.127s
Xerces	0.017s	0.021s	0.096s	0.730s	7.325s

5.2.7. DOM-muistinkulutus ilman validointia

Taulukossa 13 esitellään jäsentimien kuluttama muisti, jos dokumenttia ei validoida jäsennettäessä. Jäsentimet olivat testissä melko tasaisia.

Taulukko 13. DOM-muistinkulutus ilman validointia.

	10 kpl	100 kpl	1000 kpl	10 000 kpl	100 000kpl
Crimson	0.657Mb	0.663Mb	2.410Mb	21.928Mb	215.662Mb
Xerces	0.710Mb	0.791Mb	2.382Mb	20.162Mb	195.420Mb
Gnu	0.678Mb	0.679Mb	2.473Mb	21.93Mb	215.429Mb

5.2.8. DOM-muistinkulutus ja dokumentin validointi

Taulukossa 14 esitellään DOM-muistinkulutus dokumenttia validoidessa. Tätä ominaisuutta tuki vain 2 tutkimuksen jäsennintä. Validointi vaikutti kasvattavasti muistin kulutukseen molemmilla jäsentimillä.

Taulukko 14. DOM-muistinkulutus ja dokumentin validointi.

	10 kpl	100 kpl	1000 kpl	10 000 kpl	100 000kpl
Crimson	0.659Mb	0.654Mb	2.511Mb	22.280Mb	224.813Mb
Xerces	0.717Mb	0.742Mb	2.467Mb	20.929Mb	204.961Mb

6. Yhteenveto

Tämän luvun sisältämiin taulukoihin 15 ja 16 on kerätty yhteenveto tehokkaimmista jäsentimistä nopeuden ja muistinkulutuksen perusteella. Mielenkiintoisesti SAX-jäsentimistä Aelfred2-jäsennin käytti vähiten muistia pieniä tiedostoja käsitellessä jos validointi oli asetettu, mutta tiedostokokojen kasvaessa Crimson osoittautui tehokkaammaksi jäsentimeksi. Toinen huomionarvoinen havainto on, että dokumentin validointia suoritettaessa Nopeudeltaan paras DOM-jäsennin oli selkeästi Xerces, joka selvisi lähes jokaisesta tiedostosta muita nopeammin.

Taulukko 15. Yhteenvetotaulukko SAX-jäsentimien tuloksista.

	10	100	1000	10 000	100 000
Nopein, ei validointia	Kaikki	Kaikki	Piccolo	Piccolo	Piccolo
Nopein, validointi	Kaikki	Kaikki	Crimson	Crimson	Xerces
Vähiten muistia käyttävä, ei validointia	Aelfred2	Aelfred2	Aelfred2	Crimson	Crimson
Vähiten muistia käyttävä, validointi	Aelfred2	Aelfred2	Crimson	Crimson	Crimson

Taulukko 16. Yhteenvetotaulukko DOM-jäsentimien tuloksista.

	10	100	1000	10 000	100 000
Nopein, ei validointia	Gnu, Crimson, Xerces	Gnu, Crimson, Xerces	Xerces	Xerces	Xerces
Nopein, validointi	Crimson, Xerces	Crimson	Xerces	Xerces	Xerces
Vähiten muistia käyttävä, ei validointia	Crimson	Crimson	Xerces	Xerces	Xerces
Vähiten muistia käyttävä, validointi	Crimson	Crimson	Xerces	Xerces	Xerces

Viiteluettelo

[1] Elliotte R. Harold (2005), *Processing XML with Java.* Wesley, USA

[2] Poornachandra Sarang (2006), *Pro Apache XML.* Apress, USA

[3] Robert Richard (2006), *Pro PHP XML and Web Services.* Apress, USA

[4] World Wide Web Consortium, `http://www.w3.org/DOM` (1.3-30.3.2007)

[5] Mark Birbeck (2004), *Professional XML, 2nd Edition.* Apress USA

[6] W3 Schools Online Web Tutorials, `http://www.w3schools.com` (1.3.-30.3.2007)

[7] Oracle, `http://www.oracle.com` (1.3.-30.3.2007)

[8] Sax Project, `http://www.saxproject.org` (1.3.-30.3.2007)

[9] The Apache XML Graphics Project, `http://xmlgraphics.apache.org` (1.3.-30.3.2007)

[10] The Apache XML Project, `http://xerces.apache.org` (1.3.-30.3.2007)

[11] The Apache XML Project, `http://xml.apache.org/crimson` (1.3.-30.3.2007)

[12] Piccolo XML Parser for Java, `http://piccolo.sourceforge.net` (24.4.2008)

[13] The GNU JAXP Project, `http://www.gnu.org/software/classpathx/jaxp` (1.3.-30.3.2007)

[14] The Aelfred XML Parser, `http://saxon.sourceforge.net/aelfred.html` (1.3.-30.3.2007)

[15] Developer Resources for Java Technology, `http://java.sun.com` (28.4.2008)

Liite 1: Python-kielinen skripti, joka luo tutkimuksessa käytettyjä XML-dokumentteja

```
class XMLCreator:
 def __init__(self,filename,size):

  f=open(filename,"w");
  f.write("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"
standalone=\"yes\"?>\n")
  f.write("<!DOCTYPE catalog[\n");
  f.write("<!ELEMENT catalog (book+)>\n");
  f.write("<!ELEMENT book
(title,authors,publisher,year,listprice,isbn)>\n");
  f.write("<!ATTLIST book cover (hard|paper) \"hard\">\n");
  f.write("<!ELEMENT title (#PCDATA)>\n");
  f.write("<!ELEMENT authors (author+)>\n");
  f.write("<!ELEMENT author (last,first)>\n");
  f.write("<!ELEMENT first (#PCDATA)>\n");
  f.write("<!ELEMENT last (#PCDATA)>\n");
  f.write("<!ELEMENT publisher (#PCDATA)>\n");
  f.write("<!ELEMENT year (#PCDATA)>\n");
  f.write("<!ELEMENT listprice (#PCDATA)>\n");
  f.write("<!ELEMENT isbn (#PCDATA)>\n");
  f.write("<!ATTLIST isbn id ID #REQUIRED>\n");

  f.write("]>\n")

  f.write("<catalog>\n")
  for i in range(size):
      f.write("<book cover=\"hard\">\n")
      f.write("<title>book "+repr(i)+"</title>\n")
      f.write("<authors>\n")
      f.write("<author>\n")
      f.write("<last> lastname </last>\n");
      f.write("<first> firstname </first>\n");
      f.write("</author>\n")
      f.write("</authors>\n")
      f.write("<publisher> PUBLISHER </publisher>\n")
      f.write("<year>"+repr(i)+"</year>\n");
      f.write("<listprice>100</listprice>\n");
      f.write("<isbn id = \"book-"+repr(i)+"\"></isbn>\n");
      f.write("</book>\n")
      f.write("</catalog>\n");
      f.close()
```

Liite 2. Mittausohjelmiston tärkeimmät osat

```
/*
 * CrimsonDomProperties.java
 *
 * Created on March 2, 2007, 3:36 PM
 */

package parserbenchmark;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.DOMImplementation;

public class CrimsonDomProperties {

public CrimsonDomProperties() {

  DocumentBuilderFactory dbf = javax.xml.parsers.DocumentBuilderFactory
    .newInstance();
  try {
   DocumentBuilder b = dbf.newDocumentBuilder();

   DOMImplementation dimpl = b.getDOMImplementation();
   for (int i = 0; i < Globals.domFeatures.length; i++)

    if (dimpl.hasFeature(Globals.domFeatures[i], "2.0"))
     System.out.println("supports " + Globals.domFeatures[i]);
    else
     System.out.println(" ** Doesn't support "
       + Globals.domFeatures[i]);

   for (int i = 0; i < Globals.domFeatures.length; i++)
    if (dimpl.hasFeature(Globals.dom3Features[i], "3.0"))
     System.out.println("supports 3.0 "
       + Globals.dom3Features[i]);
    else
     System.out.println(" ** Doesn't support 3.0 "
       + Globals.dom3Features[i]);
  } catch (Exception e) {
  }
 }
}
```

```
/*
 * MeasureThread.java
 *
 * Created on January 23, 2007, 7:51 PM
 *
 */

package parserbenchmark;

import java.io.BufferedWriter;

public class MemoryMeasureThread extends Thread {
 public Runtime runtime;
 public long memory;
 public long starttime;
 private long currenttime;
 private String resultfile;
 public Thread parserthread;
 public long maxmem;
 private BufferedWriter out;
 public String fileToHandle;

 /** Creates a new instance of MeasureThread */
 public MemoryMeasureThread() {
  runtime = Runtime.getRuntime();
  maxmem = 0;
 }

 public MemoryMeasureThread(Thread t, String filename) {
 }

 public MemoryMeasureThread(Thread t) {

  runtime = Runtime.getRuntime();

  maxmem = 0;
  parserthread = t;

 }

 public void writeToFile(String s) {

  try {
   out.append(s);
  } catch (Exception e) {
   System.out.println("failed to write to file");
  }
 }

 @Override
 public void start() {
  runtime.gc();
  starttime = System.currentTimeMillis();
  this.run();
  memory = runtime.freeMemory();
 }

 @Override
 public void run() {
```

```
  while (parserthread.isAlive()) {
   memory = runtime.totalMemory() - runtime.freeMemory();

   try {
    this.sleep(Globals.memThreadSleepTime);
   } catch (Exception e) {
   }

   if (memory > maxmem) {
    maxmem = memory;
    System.out.println("Current highest amount of used memory: "
      + (float) maxmem / (1024 * 1024));
    if (maxmem > ThreadEngine.mostmem)
     ThreadEngine.mostmem = maxmem;
   }
  } // END OF WHILE

  try {

  } catch (Exception e) {
  }
 }
}
```

```
/*
 * SaxHandler.java
 * Created on January 23, 2007, 8:45 PM
 */

package parserbenchmark;

import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.Locator;
import org.xml.sax.SAXException


public class SaxHandler implements ContentHandler {
 public Runtime runtime;
 public long memory;
 public long maxmem;

 /** Creates a new instance of SaxHandler */
 public SaxHandler() {
  runtime = Runtime.getRuntime();
  maxmem = 0;
  runtime.gc();
 }

 public void characters(char[] text, int start, int length) {
  if (Main.measuremem) {

   memory = runtime.totalMemory() - runtime.freeMemory();
   if (memory > maxmem) {
    maxmem = memory;
    if (maxmem > ThreadEngine.mostmem)
     ThreadEngine.mostmem = maxmem;
    System.out.println("mostmem : " + (float) ThreadEngine.mostmem
      / (1024 * 1024) + "Mb");
   }
  }

 }

 public void startDocument() {
 }

 public void endDocument() {
  memory = runtime.totalMemory() - runtime.freeMemory();

  if (memory > maxmem && Main.measuremem) {
   maxmem = memory;
   if (maxmem > ThreadEngine.mostmem)
    ThreadEngine.mostmem = maxmem;
   System.out.println("mostmem : " + (float) ThreadEngine.mostmem
     / (1024 * 1024) + "Mb");
  }

 }

 public void startElement(String namespaceURI, String localName,
   String qualifiedName, Attributes atts) {
 }
```

```
  public void endElement(String namespaceURI, String localName,
    String qualifiedName) {
  }

  public void startPrefixMapping(String prefix, String uri) {
  }

  public void endPrefixMapping(String prefix) {
  }

  public void ignorableWhitespace(char[] text, int start, int length)
    throws SAXException {
  }

  public void processingInstruction(String target, String data) {
  }

  public void setDocumentLocator(Locator locator) {
  }

  public void skippedEntity(String name) {
  }
}
```

```
/*
 * SaxParserThread.java
 *
 * Created on January 23, 2007, 7:32 PM
 *
 */

package parserbenchmark;

import java.io.FileReader;
import java.io.IOException;

import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;
 public class SaxParserThread {
 public XMLReader xmlreader;

 public String filename;
 private final ContentHandler saxhandler;
 public float time;
 private final ErrorHandler errorhandler;
 private InputSource in;
 /** Creates a new instance of SaxParserThread */
 public String driver;

 public SaxParserThread(String drivername, String fn) {
  filename = fn;
  saxhandler = new SaxHandler();
  driver = drivername;
  errorhandler = new ErrorReporter();

  try {
   FileReader r = new FileReader(filename);
   InputSource in = new InputSource(r);
  } catch (Exception e) {
   System.out.println("ERROR...");
  }

  try {
   xmlreader = XMLReaderFactory.createXMLReader(drivername);
   xmlreader.setContentHandler(saxhandler);
   xmlreader.setErrorHandler(errorhandler);

  } catch (SAXException se) {
   System.out.println("virhe tapahtui jasentimen valinnassa");
   Throwable t = new Throwable();
   t.printStackTrace();
   System.exit(0);
  }
 }

 public void run() {
  try {
  } catch (Exception e) {
  }
```

```
  try {
   xmlreader.parse(filename);
  } catch (SAXException e) {
   System.out.println("SAX exception");

   Throwable t = new Throwable();
   t.printStackTrace();
   System.exit(0);

  } catch (IOException ie) {
   System.out.println("IOEXCEPTION :");
   ie.printStackTrace();
  }

 }

 public String getFileName() {
  return filename;
 }

 public void start() {
  run();
 }
}
```

```
/*
 * ThreadEngine.java
 *
 * Created on January 23, 2007, 7:57 PM
 *
 */

package parserbenchmark;

import java.util.Calendar;

public class ThreadEngine {

 private String filename; // käsiteltävä tiedosto
 private String driver; // XML ajuri
 private String api; // rajapinta, jota käytetään
 public float time; // aikamittaus- muuttuja
 public boolean validate = false; // validointi dokumentille
 public boolean namespaces = false; // namespaces on / off
 public boolean namespace_prefixes = false; // ns prefixes on / off

 public static long mostmem = 0;

 private SaxParserThread saxthread; // jäsenninsäie
 private MemoryMeasureThread memthread; // muistimittaussäie

 /** Creates a new instance of ThreadEngine */
 public ThreadEngine() {
 }

 public String makeFileNameFromDate(String FileName, String drivername) {
  String resultfile = "";


  Calendar C = Calendar.getInstance();

  int day = C.get(Calendar.DAY_OF_MONTH);
  int month = C.get(Calendar.MONTH);
  int hour = C.get(Calendar.HOUR_OF_DAY);
  int minute = C.get(Calendar.MINUTE);
  String ff = FileName.replace('/', '.');
  ff = ff.replace(':', '.');
  resultfile += ff + "_" + drivername + "_" + day + "." + month + "_"
    + hour + "." + minute + ".xml";

  return resultfile;
 }

 public ThreadEngine(String FileName, String DriverName, String Api) {

  filename = FileName;
  api = Api;
  driver = DriverName;
 }

 public void startEngine() {
  /**********************************************************************
         * DOM RAJAPINTA
   **********************************************************************/
  String resultfile = "";
```

```
String DriverName = driver;
String FileName = filename;
time = 0;

long starttime, endtime;
Calendar C = Calendar.getInstance();

if (api.equals(Globals.API_DOM) && (Main.measuremem)) {
 System.out.println("Measuring dom mem..");
 /*****************************************************************
  * MITATAAN MUISTIN KÄYTTÖ, DOM
  *****************************************************************/

 Runtime r = Runtime.getRuntime();
 r.gc();

 Thread parser = new Thread();

 if (DriverName.equals(Globals.DRIVER_XERCES_DOM)) {
  DOMDocumentParserThread domparser = new DOMDocumentParserThread(
    filename, "XERCES");

  parser = new Thread(domparser);

  if (validate)
   try {
    domparser.xercesparser.setFeature(
      "http://xml.org/sax/features/validation", true);
   } catch (Exception e) {
    System.out.println("error setting validation on ");
   }

 }


 // CRIMSON DOM
 if (DriverName.equals(Globals.DRIVER_CRIMSON_DOM)) {
  DOMDocumentParserThread domparser = new DOMDocumentParserThread(
    filename, "CRIMSON");

  parser = new Thread(domparser);
  if (validate) {
   try {
    domparser.factory.setValidating(true);

   } catch (Exception e) {
    System.out.println("ERROR SETTING VALIDATION ON...");
   }
  } else {
   try {
    domparser.factory.setValidating(false);

   } catch (Exception e) {
    System.out.println("ERROR SETTING VALIDATION FALSE...");
   }

  }
 }

 // GNU DOM
 if (DriverName.equals(Globals.DRIVER_GNU_DOM)) {
```

```
  DOMDocumentParserThread domparser = new DOMDocumentParserThread(
    filename, "GNUDOM");

  parser = new Thread(domparser);
  if (validate) {
   try {
    domparser.factory.setValidating(true);
   } catch (Exception e) {
    System.out.println("ERROR SETTING VALIDATION ON...");
   }
  } else {
   try {
    domparser.factory.setValidating(false);

   } catch (Exception e) {
    System.out.println("ERROR SETTING VALIDATION FALSE...");
   }
  }
 }

 MemoryMeasureThread memthread = new MemoryMeasureThread(parser);

 parser.start();
 memthread.start();
 try {
  parser.join();
 } catch (Exception e) {
 }

 long memory = r.totalMemory() - r.freeMemory();

 System.out.println("las mem : " + (float) memory / (1024 * 1024));

}// END OF

/**********************************************************************
 * DOM Ajanmittaukset
 **********************************************************************/

if (api.equals(Globals.API_DOM) && (!Main.measuremem)) {
 DOMDocumentParser parser = new DOMDocumentParser();

 resultfile += "DOM results\\";

 resultfile += makeFileNameFromDate(FileName, driver);

 System.out
   .println("kaynnistetaan jäsennysthread...parser passes = "
     + Main.parser_passes);

 Runtime r = Runtime.getRuntime();

 r.gc();

 // XERCES DOM
 if (DriverName.equals(Globals.DRIVER_XERCES_DOM)) {
  parser = new DOMDocumentParser(filename, "XERCES");
  if (validate)
   try {
    parser.xercesparser.setFeature(
      "http://xml.org/sax/features/validation", true);
```

```
  } catch (Exception e) {
   System.out.println("error setting validation on ");
  }
}


// CRIMSON DOM
if (DriverName.equals(Globals.DRIVER_CRIMSON_DOM)) {
 parser = new DOMDocumentParser(filename, "CRIMSON");
 if (validate) {
  try {
   parser.factory.setValidating(true);

  } catch (Exception e) {
   System.out.println("ERROR SETTING VALIDATION ON...");
  }
 } else {
  try {
   parser.factory.setValidating(false);

  } catch (Exception e) {
   System.out.println("ERROR SETTING VALIDATION FALSE...");
  }
 }
}

// GNU DOM
if (DriverName.equals(Globals.DRIVER_GNU_DOM)) {
 parser = new DOMDocumentParser(filename, "GNUDOM");
 if (validate) {
  try {
   parser.factory.setValidating(true);
  } catch (Exception e) {
   System.out.println("ERROR SETTING VALIDATION ON...");
  }
 } else {
  try {
   parser.factory.setValidating(false);

  } catch (Exception e) {
   System.out.println("ERROR SETTING VALIDATION FALSE...");
  }
 }
}

if (!Main.measuremem)
 parser.start(); // 1 jäsennys ennen aloitusta

time = 0;

for (int i = 1; i <= Main.parser_passes; i++) {
 starttime = System.currentTimeMillis();
 parser.start();
 endtime = System.currentTimeMillis();

 if (endtime - starttime > 0)
  time += (endtime - starttime);
 else
  i -= 1;
}
```

```
 time /= Main.parser_passes;
 time /= 1000;

 long totmem = r.totalMemory();
 long freemem = r.freeMemory();

 long mem = totmem - freemem;

 System.out
   .println("Memory used : " + ((float) mem / (1024 * 1024)));

 System.out.println("Aika (AVG) :" + time + " s");

}

/**********************************************************************
 * Jos SAX rajapinta.
 **********************************************************************/

if (api.equals(Globals.API_SAX)) {

 resultfile += "SAX results\\";

 int day = C.get(Calendar.DAY_OF_MONTH);
 int month = C.get(Calendar.MONTH);
 int hour = C.get(Calendar.HOUR_OF_DAY);
 int minute = C.get(Calendar.MINUTE);
 String drivername = "";
 if (driver == Globals.DRIVER_AELFRED)
  drivername = Globals.PARSER_AELFRED;
 if (driver == Globals.DRIVER_CRIMSON)
  drivername = Globals.PARSER_CRIMSON;
 if (driver == Globals.DRIVER_XERCES_SAX)
  drivername = Globals.PARSER_XERCES;
 if (driver == Globals.DRIVER_PICCOLO)
  drivername = Globals.PARSER_PICCOLO;
 String ff = FileName.replace('/', '.');
 ff = ff.replace(':', '.');
 resultfile += ff + "_" + drivername + "_" + day + "." + month + "_"
   + hour + "." + minute + ".xml";

 System.out
   .println("kaynnistetaan jäsennysthread...parser passes = "
     + Main.parser_passes);

 System.out.println("driver package :" + driver);

 boolean firstpass = true;
 saxthread = new SaxParserThread(driver, filename);

 // asetetaan jäsentimien piirteet

 if (validate) {
  try {
   saxthread.xmlreader.setFeature(
     "http://xml.org/sax/features/validation", true);
  } catch (Exception e) {
   System.out
     .println(" -- ERROR setting feature VALIDATION ON...");
  }
 }
```

```
try {
} catch (Exception e) {
}

if (!validate)
 try {
  saxthread.xmlreader.setFeature(
    "http://xml.org/sax/features/validation", false);

 } catch (Exception e) {
  System.out
    .println(" -- ERROR setting feature VALIDATION OFF...");
 }

if (!namespaces)
 try {
  saxthread.xmlreader.setFeature(
    "http://xml.org/sax/features/namespaces", false);
 } catch (Exception e) {
  System.out
    .println(" -- ERROR setting feature NAMESPACES OFF..");
 }

if (namespaces)
 try {
  saxthread.xmlreader.setFeature(
    "http://xml.org/sax/features/namespaces", true);
 } catch (Exception e) {
  System.out
    .println(" -- ERROR setting feature NAMESPACES ON..");
 }

if (!namespace_prefixes)
 try {
  saxthread.xmlreader.setFeature(
    "http://xml.org/sax/features/namespace-prefixes",
    false);
 } catch (Exception e) {
  System.out
    .println(" -- ERROR setting property NS PREFIXES OFF..");
 }

if (namespace_prefixes)
 try {
  saxthread.xmlreader.setFeature(
    "http://xml.org/sax/features/namespace-prefixes",
    true);
 } catch (Exception e) {
  System.out
    .println(" -- ERROR setting feature NS PREFIXES ON..");
 }

if (!Main.measuremem)
 saxthread.start(); // 1 jäsennys ennen aloitusta

endtime = 0;

for (int i = 1; i <= Main.parser_passes; i++) {
 starttime = System.currentTimeMillis();
```

```
    saxthread.start();
    endtime = System.currentTimeMillis();

    if (endtime - starttime == 0)
     i -= 1;
    else
     time += endtime - starttime;

   } // end of loop i

   time /= Main.parser_passes;
   time /= 1000;
   System.out.println("Aika (avg) :" + time + " s");
  }
 }

 public void setAPI(String APIName) {
  api = APIName;
 }

 public void setFileName(String FileName) {
  filename = FileName;
 }

 public void setDriver(String DriverName) {
  driver = DriverName;
 }

}
```

```
/*
 * XercesParserProperties.java
 * Created on March 2, 2007, 2:49 PM
 */

package parserbenchmark;
import org.w3c.dom.DOMImplementation;
import com.sun.org.apache.xerces.internal.dom.DOMImplementationImpl;

public class XercesDomProperties {
 /** Creates a new instance of XercesParserProperties */
 public XercesDomProperties() {
  DOMImplementation impl = DOMImplementationImpl.getDOMImplementation();
  for (int i = 0; i < Globals.domFeatures.length; i++)
   if (impl.hasFeature(Globals.domFeatures[i], "2.0"))
    System.out.println("supports " + Globals.domFeatures[i]);
   else
    System.out.println(" ** Doesn't support "
      + Globals.domFeatures[i]);

  for (int i = 0; i < Globals.dom3Features.length; i++)

   if (impl.hasFeature(Globals.dom3Features[i], "3.0"))
    System.out.println("supports 3.0 " + Globals.dom3Features[i]);
   else
    System.out.println(" ** Doesn't support 3.0 "
      + Globals.dom3Features[i]);

 }
}
```