Parsi-projekti

Juho Tammela
Olli Kauppinen
Vili Auvinen

Sovellusraportti

Versio 0.5.0

Julkinen

21.6.2011

Jyväskylän yliopisto
Tietotekniikan laitos
Jyväskylä

Hyväksyjä	Päivämäärä	Allekirjoitus	Nimenselvennys
Projektipäällikkö	___.___.2011		
Tilaaja	___.___.2011		
Ohjaaja	___.___.2011		

Tietoja dokumentista

Tekijät:

- Vili Auvinen (VA) vili.k.auvinen@jyu.fi 050-3233607
- Olli Kauppinen (OK) olli.kauppinen@jyu.fi 040-9107920
- Juho Tammela (JT) juho.i.tammela@jyu.fi 040-7605427

Dokumentin nimi: Parsi-projekti, Sovellusraportti

Sivumäärä: 22

Tiedosto: sovellusraportti_0.5.0.odt

Tiivistelmä

Parsi-projekti suunnitteli ja toteutti Jyväskylän yliopiston tietotekniikan laitokselle prototyypin sovelluksesta, jolla voidaan tarkastaa Microsoft Officen ja OpenOffice.orgin tekstinkäsittelyohjelmistoilla laadittujen XML-pohjaisten dokumenttien muotoiluja ja rakennetta. Sovellus on kehitetty kurssin *Tietokone ja tietoverkot työvälineenä* opiskelijoiden ja opettajien käyttöön. Harjoitustöistä tarkastettavat kohteet ja virheistä annettavat palautteet on määritelty XML-vaatimustiedostossa. Sovellusraportti kuvaa projektissa toteutetun sovelluksen käyttöliittymää, rakennetta, ohjelmointikäytänteitä ja työkaluja. Lisäksi siinä kuvataan tavoitteiden toteutumista sekä sovelluksen puutteita, virheitä ja jatkokehitysideoita.

Avainsanat

DOM, jatkokehitys, käyttöliittymä, Microsoft Office, ohjelmointikäytänteet, OpenOffice.org, Python, rakenne, sovellusraportti, tavoitteet, tiedostomuodot, toteutuminen, viat, XML.

Muutoshistoria

Versio	Päivämäärä	Muutokset	Muokkaaja
0.0.1	11.05.11	Dokumentin laatiminen on aloitettu ottamalla projektiraportti pohjaksi.	OK
0.0.2	23.05.11	Dokumentin rakennetta on muutettu, kirjoitettu lisää sisältöä sekä korjattu virheitä ja muotoilua.	OK
0.1.0	26.05.11	Kuvia on lisätty, viitteet on korjattu sekä viat ja jatkokehitysideat on lisätty.	OK
0.1.1	30.05.11	Kirjoitus- ja asiavirheitä on korjattu.	OK
0.2.0	31.05.11	Lähdeluettelo on korjattu, otsikoita on muutettu kuvaavimmiksi ja kappaleiden järjestystä on muutettu loogisemmaksi.	OK
0.2.1	07.06.11	Kirjoitusvirheitä on korjattu.	OK
0.3.0	14.06.11	Rajapintoja on kuvattu tarkemmin ja yhteenvetoa on kirjoitettu lisää.	OK
0.3.1	18.06.11	Lähdeluettelo ja kirjoitusvirheet on korjattu.	OK
0.4.0	19.06.11	Kuvaus käytetyistä valmiskomponenteista on lisätty.	OK
0.5.0	21.06.11	Testauksen tulokset on lisätty.	OK

Tietoa projektista

Parsi-projektissa toteutettiin kurssille *Tietokone ja tietoverkot työvälineenä* prototyyppi sovelluksesta, jolla voidaan tarkastaa Microsoft Officen ja OpenOffice.orgin tekstinkäsittelyohjelmistoilla laadittujen XML-pohjaisten dokumenttien muotoiluja ja rakennetta.

Tekijät:

- Vili Auvinen (VA) vili.k.auvinen@jyu.fi 050-3233607
- Olli Kauppinen (OK) olli.kauppinen@jyu.fi 040-9107920
- Juho Tammela (JT) juho.i.tammela@jyu.fi 040-7605427

Tilaaja:

- Antti Ekonoja antti.ekonoja@jyu.fi 014-2602746
- Tommi Lahtonen tommi.lahtonen@jyu.fi 014-2602746

Ohjaajat:

- Jukka-Pekka Santanen santanen@mit.jyu.fi 014-2602756
- Mikko Tyrväinen mikko.t.tyrvainen@jyu.fi 040-5926151

Yhteystiedot:

- Sähköpostilista parsi@korppi.jyu.fi
- Sähköpostiarkisto https://korppi.jyu.fi/list-archive/parsi

Sisällysluettelo

1 Johdanto

Jyväskylän yliopiston tietotekniikan laitos järjestää kurssia *Tietokone ja tietoverkot työvälineenä*. Kurssin harjoitustöissä opiskelijat kirjoittavat Microsoft Officen ja OpenOffice.orgin toimisto-ohjelmilla dokumentteja. Käsin harjoitustöiden tarkastaminen on hidasta ja työlästä, eikä kaikille eri tiedostoformaateille ole toteutettu tapaa koneelliseen tarkastamiseen.

Parsi-projekti suunnitteli ja toteutti tietotekniikan laitokselle prototyypin sovelluksesta, jolla voidaan ohjelmallisesti tarkastaa Microsoft Officen ja OpenOfficen toimisto-ohjelmilla laadittuja dokumentteja. Sovelluksella pystyy tarkastamaan tekstinkäsittelydokumenttien sisältöä, muotoa ja rakennetta. Jatkokehityksessä siihen tullaan toteuttamaan myös esitysgrafiikka- ja taulukkolaskentadokumenttien sekä WWW-sivujen tarkastaminen.

Sovellusraportti kuvaa, kuinka vaatimusmäärittelyn [9] toiminnalliset vaatimukset toteutettiin ohjelmallisesti. Lisäksi se kuvaa sovelluksen rakenteen, käyttöliittymän, toiminnan ja jatkokehitysideat. Raportin laatimisessa on hyödynnetty Parsi-projektin läpivientiä kuvaavaa projektiraporttia [3], järjestelmätestausraporttia [11], Tabu- ja Verso-projektin sovellusraportteja [14] ja [13], sekä tietotekniikan Sovellusprojektien ohjetta [15]. Lisäksi on hyödynnetty projektin aikana laadittuja tiedostomuotojen esittelydokumentteja [8] ja [10].

Luvussa 2 esitellään dokumentissa käytetyt termit ja niiden merkitys. Luvussa 3 kuvataan projektin taustaa ja tarkastettavia tiedostomuotoja. Luvussa 4 tarkastellaan sovelluksen käyttöliittymää, rakennetta ja rajapintoja. Luvussa 5 esitellään ohjelmointikäytänteet ja käytetyt kehitysvälineet. Luvussa 6 kuvataan sovelluksen tavoitteiden toteutumista projektissa sekä sovelluksen tunnettuja virheitä ja jatkokehitysideoita. Luvussa 7 annetaan sovelluksen käyttöön ja laajentamiseen liittyviä ohjeita ylläpitäjälle ja jatkokehittäjille.

2 Termit

Luvussa kuvataan dokumentissa esiintyviä aihealueeseen sekä ohjelmistoihin ja tekniikoihin liittyviä termejä.

2.1 Aihealue ja tiedostomuodot

docx	on Microsoft Office Word -tekstinkäsittelydokumenttien tiedostomuoto.
Harjoitustyö	sisältää *Tietokone ja tietoverkot työvälineenä* -kurssilla tehtävän yhden tai useamman dokumentin.
Jatkokehitys	on projektin jälkeen tapahtuvaa sovelluksen kehitystä.
odp	on OpenOffice.org Impress -esitysgrafiikkadokumenttien tiedostomuoto.
ods	on OpenOffice.org Calc -taulukkolaskentadokumenttien tiedostomuoto.
odt	on OpenOffice.org Writer -tekstinkäsittelydokumenttien tiedostomuoto.
Ohjain	on käyttöliittymän ja tarkastimien välillä oleva rajapinta, joka välittää tarkastettavat dokumentit ja niissä havaitut virheet.
pptx	on Microsoft Office PowerPoint -esitysgrafiikkadokumenttien tiedostomuoto.
Tarkastin	on tietylle tiedostomuodolle tarkoitettu virheiden etsijä.
Tiedosto-muodot	käsittävät kaikki harjoitustyöhön liittyvät toimisto-ohjelmien tiedostomuodot.
xlsx	on Microsoft Office Excel -taulukkolaskentadokumenttien tiedostomuoto.

2.2 Ohjelmistot ja tekniikat

DOM	(*Document Object Model*) on ohjelmointirajapinta, joka mahdollistaa XML-dokumenttien sisällön tarkastelun ja muokkauksen.
Eclipse	on avoimen lähdekoodin ohjelmointiympäristö.
Epydoc	on työkalu Pythonin API-dokumentaatioiden luontiin.
Git	on hajautettu versiohallintajärjestelmä.
PyDev	on Eclipse-laajennus Pythonille.
Python	on tulkattava ohjelmointikieli.
XML	*(eXtensible Markup Language*) on rakenteisten dokumenttien yleisin merkintäkieli, jolla tiedon merkitys on kuvattavissa tiedon sekaan.
YouSource	on WWW-pohjainen Git-versiohallintaohjelmistoa tukeva lähdekoodien julkaisujärjestelmä.

3 Projektin taustaa

Luvussa kuvataan projektin taustoja ja toteutetun sovelluksen tarpeita sekä T*ietokone ja tietoverkot työvälineenä* -kurssin harjoitustyön vaatimuksia. Sovelluksen toteuttamisen mahdollisti Microsoft Officen uusien avointen tiedostomuotojen julkaiseminen, jolloin dokumenttien lukeminen tiedostosta tuli mahdolliseksi.

3.1 Kurssin dokumenttien tarkastaminen

Tietokone ja tietoverkot työvälineenä on Jyväskylän yliopistossa järjestettävä kurssi, joka kuuluu tietotekniikan ja tietojärjestelmätieteen pakollisiin perusopintoihin. Kurssi kuuluu myös joidenkin muiden tiedekuntien tutkintovaatimuksiin joko pakollisena tai valinnaisena kurssina. Se luennoidaan kaksi kertaa vuodessa, ja sen voi suorittaa myös verkkokurssina, jolloin kurssin suorittaminen tapahtuu etätyöskentelynä.

Kurssi suoritetaan **harjoitustyöllä**, jossa opiskelija laatii vaatimusten mukaisen teksti- ja WWW-dokumentin sekä esitysgrafiikkaesityksen. Harjoitustyö toteutetaan kurssin yleisten sekä henkilökohtaisten vaatimusten mukaisesti. Henkilökohtaiset vaatimukset koskevat tyylien muotoilua.

Kurssista ja harjoitustyöstä pitää tulevaisuudessa antaa **arvosana** pelkän hyväksynnän sijaan. Kehitettävää tarkastussovellusta on tarkoitus käyttää myös arvostelun tukena. Harjoitustyön vaatimukset pisteytetään, ja sovellus antaa arvosanan pisteitten mukaan.

Harjoitustöitä tulee opiskelijoilta vuosittain useampi sata, sekä niiden tarkastaminen käsin on hidasta ja työlästä. Kurssin työkaluina käytetään sekä Microsoft Officen että OpenOfficen toimisto-ohjelmia. Tekstinkäsittely- ja esitysgrafiikkaosuuden tarkastamista varten Microsoft Office 2003:lle aiemmin kehitettyjä makroja ei ole ylläpidetty, eivätkä ne toimi kaikilla toimisto-ohjelmistoilla.

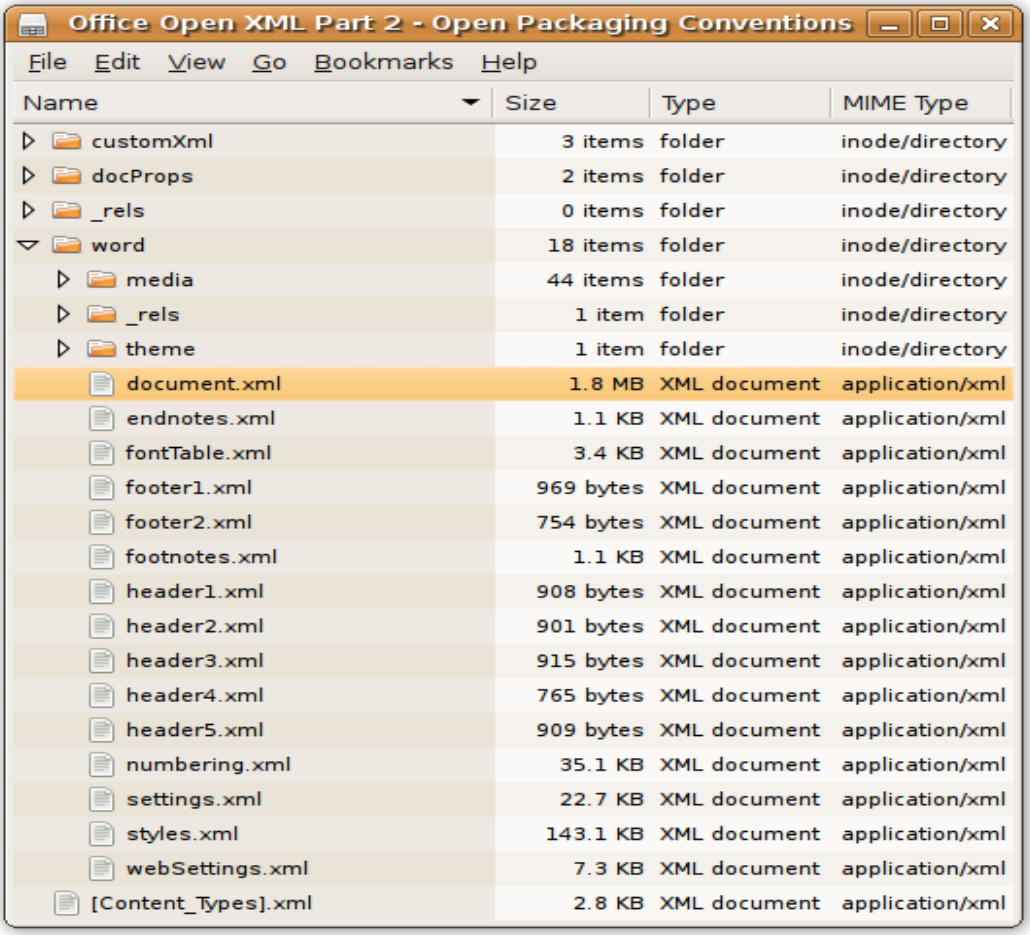
3.2 Tiedostomuodot

Kurssin työkaluina käytetään sekä Microsoft Officen että OpenOffice.orgin toimisto-ohjelmia. Microsoft Office 2007:n XML-formaatteja kutsutaan yleisemmin *Office Open XML*:ksi, kun OpenOffice.orgin vastaava formaatti on nimeltään *OpenOffice.org XML*. Myös kurssilla luotavat WWW-sivut ovat XML-dokumentteja.

Molempien ohjelmistoperheiden uusien toimisto-ohjelmaversioiden muodostamat tiedostot ovat käytännössä **purettavia zip-tiedostoja**, jotka koostuvat muutamasta hakemistosta ja useasta XML-tiedostosta. Keskeisimpiä tiedostoja molempien ohjelmistoperheiden XML-formaateissa ovat sisältö- (`document.xml` ja `content.xml`) ja tyylitiedostot (`styles.xml`). Sisältötiedostoihin tallennetaan kaikki tekstinkäsittelyohjelmalla kirjoitettu sisältö. Tyylitiedosto sisältää dokumentissa käytetyt tyylimääritykset.

Office Open XML:ssä dokumentin kaikki **XML-tiedostot** listataan tiedostossa `[Content_Types].xml`. OpenOffice.org XML:n vastaava tiedosto on `manifest.xml`. Kuvat ja muut mediatiedostot ovat `media`-kansion alla erillisinä tiedostoina. Ylä- ja alatunnisteet on eritelty omiin tiedostoihinsa `footerX.xml` ja `headerX.xml`. Wordissa `docProps`-kansion tiedostot `app.xml` ja `core.xml` sisältävät metatietoja. Niistä `app.xml` si-

sältää tietoa mm. sanamäärästä, ohjelmaversiosta ja muokkauskestosta. `core.xml` sisältää tiedot tekijästä, dokumentin luomisajasta sekä viimeisimmästä muokkaajasta ja muokkausajasta. Writerissa vastaavat tiedot löytyvät tiedostosta `meta.xml`. Kuvassa 1 esitetään esimerkkinä docx-tiedoston purettu hakemistorakenne.



Kuva 1: Puretun docx-tiedoston hakemistorakenne.

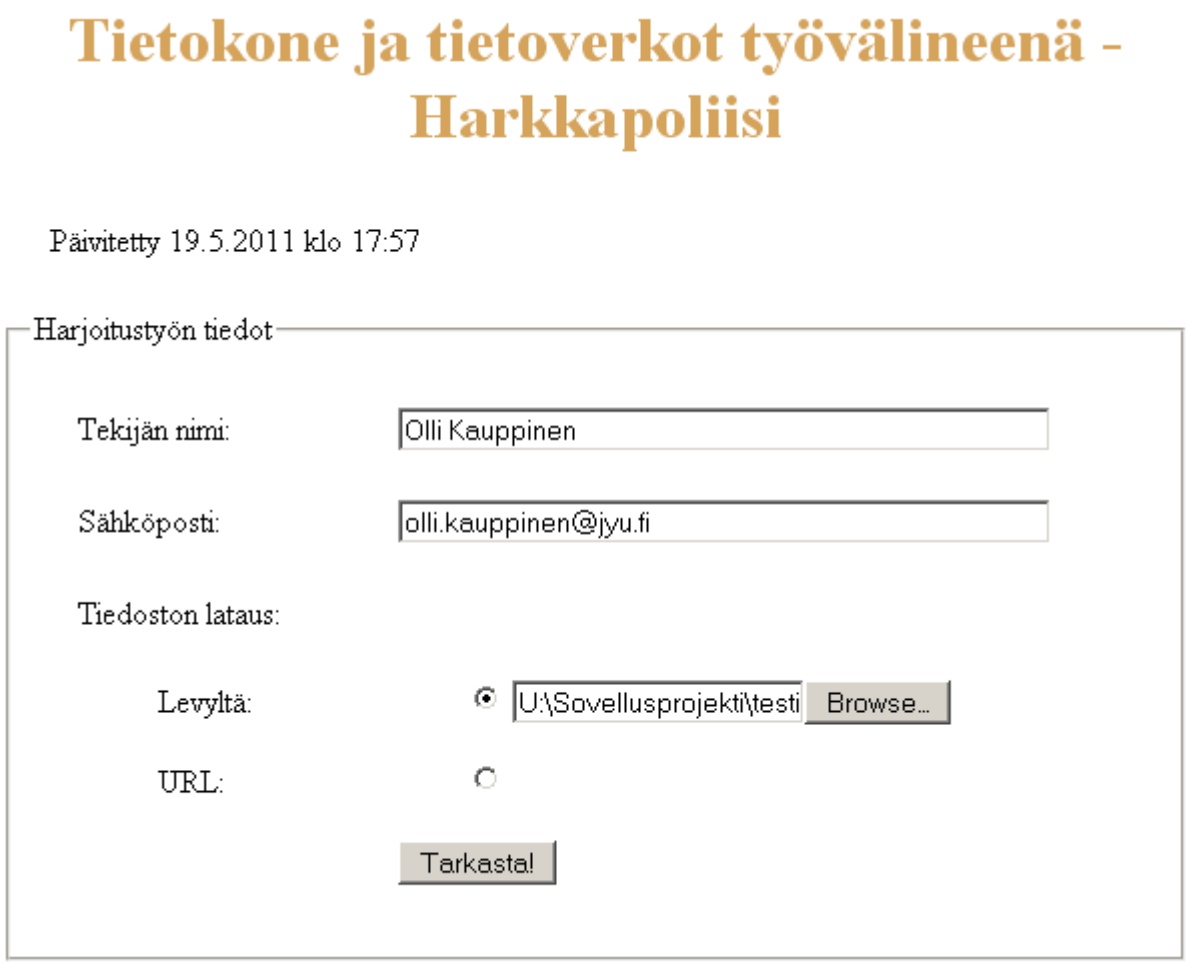
4 Sovelluksen toiminta ja rakenne

Luvussa kuvataan sovelluksen käyttöliittymää, toimintaa, rakennetta ja rajapintoja. Sovelluksen rakenne on kerroksellinen. Jokainen kerros suorittaa omat tehtävänsä ja välittää tiedon oikeassa muodossa seuraavalle kerrokselle. Sovellus on tarkoitettu sekä kurssin opettajien että oppilaiden käyttöön.

Sovelluksessa on vain kaksi eri näkymää: syötesivu ja palautesivu. Sovelluksen käyttöliittymä on toteutettu mahdollisimman yksinkertaiseksi, jotta se olisi laajan ja heterogeenisen käyttäjäjoukon helppo omaksua. Käyttäjälle annettava palaute harjoitustyöstä on ulkoasua ja toimintojen määrää huomattavasti tärkeämpi osa.

4.1 Sovelluksen syötesivu

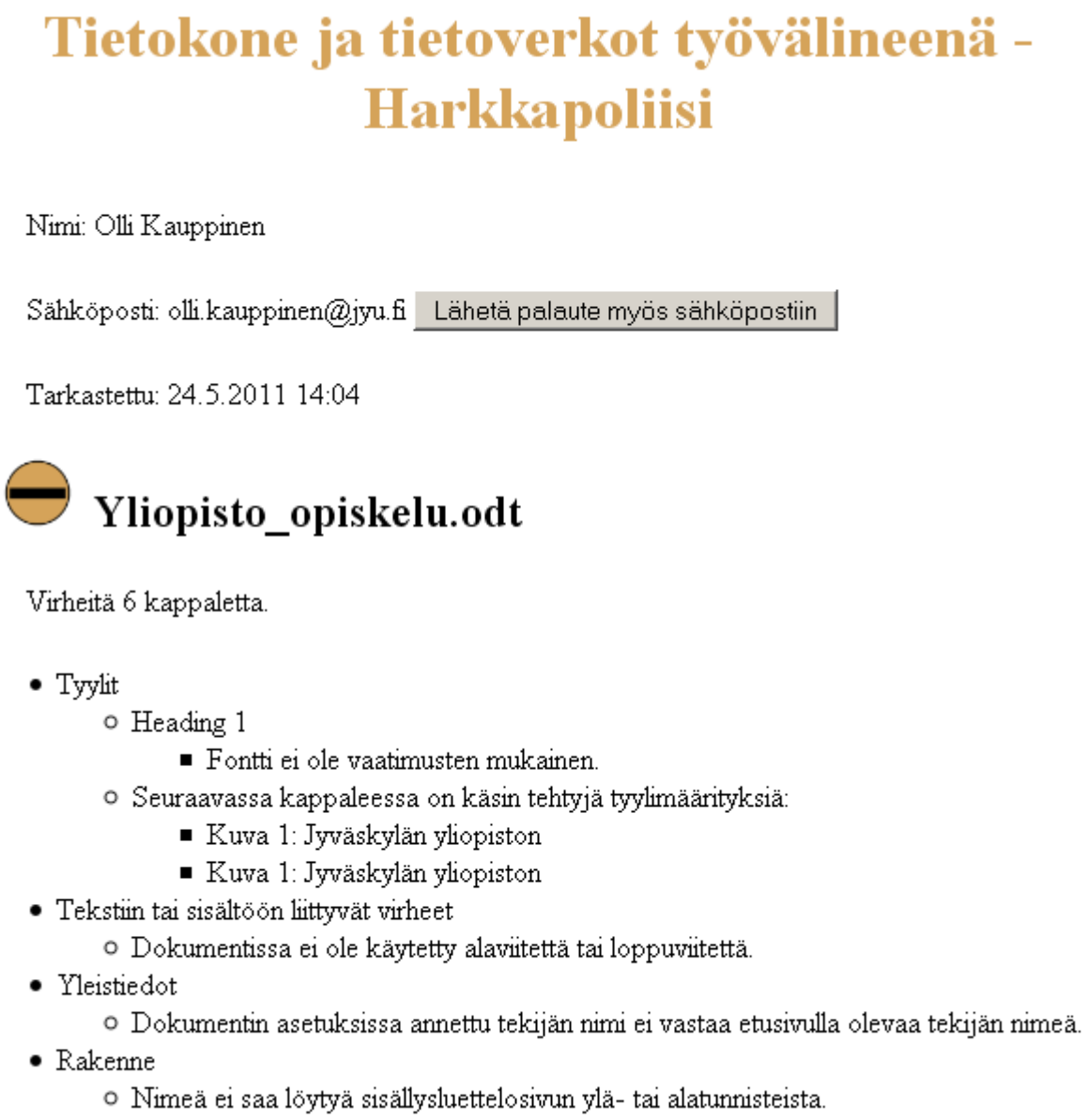
Kuvan 2 **syötesivulla** käyttäjä syöttää oman nimensä, sähköpostiosoitteensa ja tarkastettavan tiedoston. Tarkastettava tiedosto voi olla joko käyttäjän tietokoneella tai verkossa oleva tiedosto. Syötteiden jälkeen käyttäjän painettua *Tarkasta*-painiketta tarkastaminen alkaa.



Kuva 2: Sovelluksen syötesivu.

4.2 Sovelluksen palautesivu

Sovelluksen tarkastettua syötetyt dokumentit käyttäjälle näytetään kuvan 3 **palautesivu,** joka sisältää käyttäjän tiedot sekä kunkin tarkastetun dokumentin yksilöintitiedot ja sisältämät virheet. Käyttäjä voi halutessaan lähettää palautteen syötesivulla syöttämäänsä sähköpostiinsa. Harjoitustyö tarkastetaan sähköpostiosoitteen mukaan nimetyn vaatimustiedoston perusteella, joten saman sähköpostiosoitteen käyttö on välttämätöntä harjoitustyötä tilatessa, tarkastettaessa ja palautettaessa.



Kuva 3: Sovelluksen palautesivu.

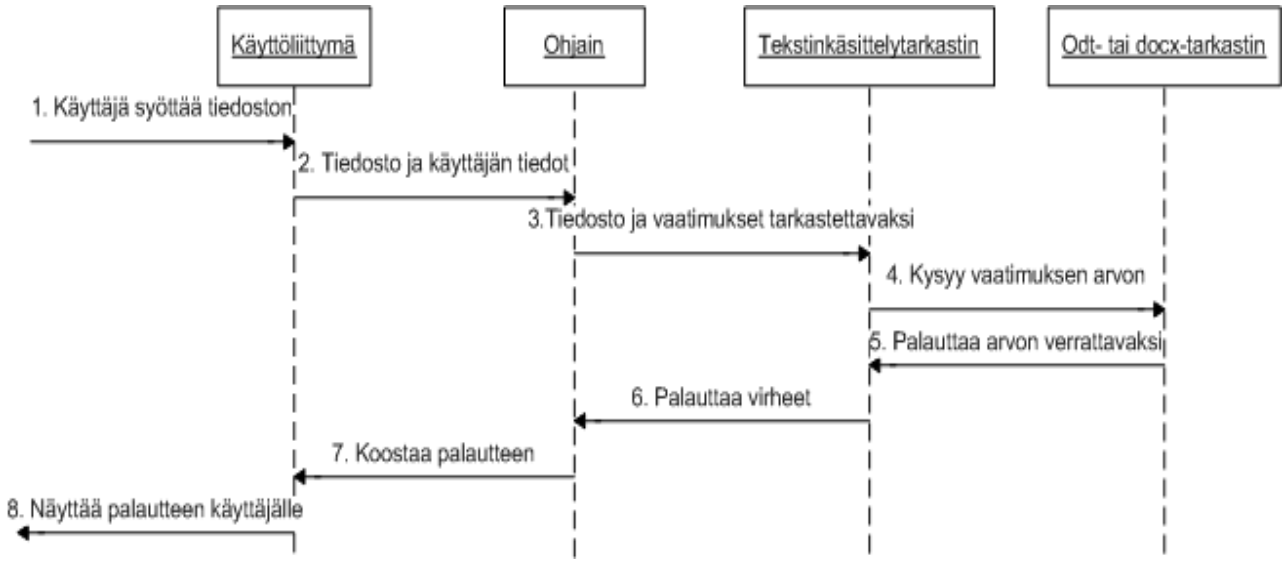
4.3 Tarkastimet kullekin ohjelmistolle

Microsoft Officen versio 2007 toi mukanaan **uudet tiedostomuodot**, joista tekstinkäsittelydokumenttien docx-tiedostot, esitysgrafiikkaesityksien pptx-tiedostot ja taulukkolaskentadokumenttien xlsx-tiedostot tallentavat tietoa XML-formaatissa. Myös OpenOfficen vastaavat tiedostoformaatit odt, odp ja ods sekä WWW-sivut koostuvat XML-dokumenteista. Eri ohjelmointikielissä XML-dokumenttien käsittelyä on tuettu hyvin esimerkiksi DOM-rajapinnan toteuttavien ohjelmointikirjastojen avulla. Tiedostoformaatit on esitelty tarkemmin esittelydokumenteissa [8] ja [10].

Kullekin toimisto-ohjelmistotyypille toteutetut tarkastimet vertaavat tarkastettavan dokumentin XML-tiedostoista löytyviä määrityksiä harjoitustyön vaatimukset sisältävään tiedostoon. **Jokaiselle tiedostoformaatille** (esim. odt) on oma tarkastin, joka ainoastaan hakee kyseisen tiedoston sisältämistä XML-tiedostoista tarkastettavien kohteiden arvot. Tekstinkäsittely-, esitysgrafiikka-, taulukkolaskenta- ja WWW-dokumenteille on oma tarkastin, joka suorittaa varsinaisen vertailun. Se vertaa alemmalta tiedostomuotokohtaiselta tarkastimelta saamiaan tietoja harjoitustyövaatimukset sisältävän tiedoston arvoihin. Mikäli arvot eroavat toisistaan, lisätään kyseistä virhettä vastaava virheilmoitus palautteeseen. Virheilmoitukset on määritetty harjoitustyövaatimusten XML-muotoisessa tiedostossa.

4.4 Sovelluksen käyttötapaus

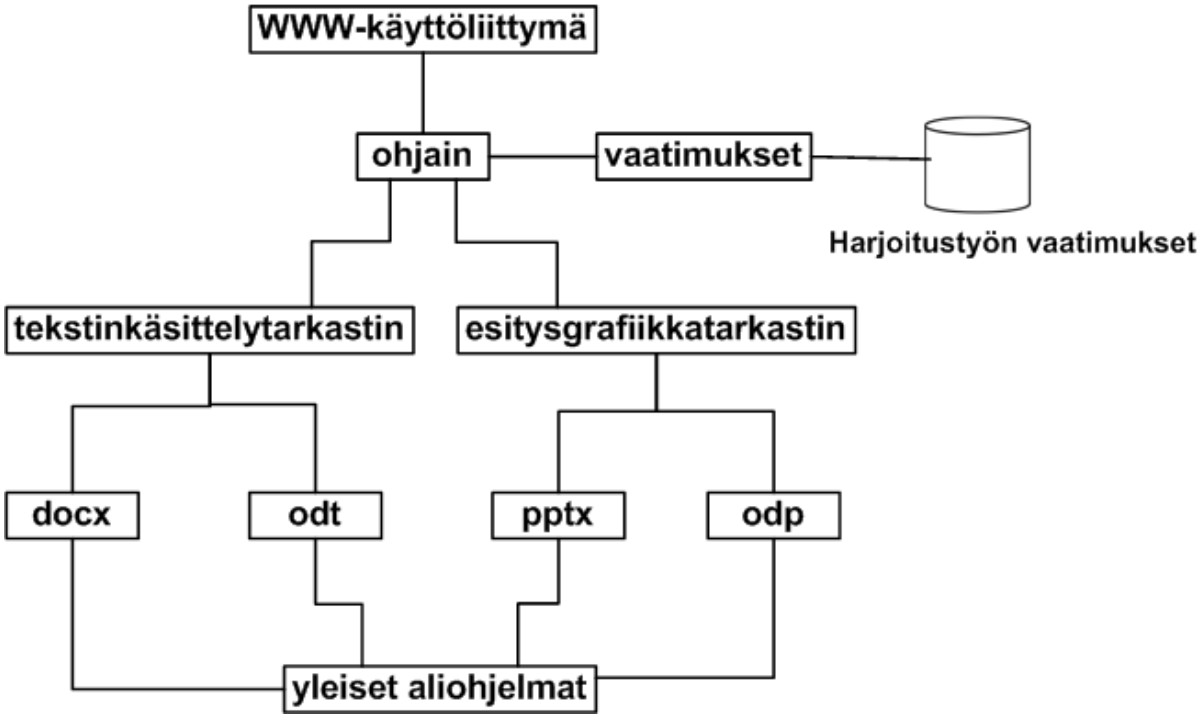
Sovellusta käyttävät yleisimmin kurssin opettajat ja oppilaat. Kuvassa 4 esitellään tavallisen tarkastuskerran kulku. Kohtia 4 ja 5 toistetaan niin kauan, että kaikki tarkastettavat kohdat on suoritettu. Kohtia 3 ja 6 toistetaan yhtä monta kertaa kuin tarkastettavia dokumentteja on tarkastettavana.



Kuva 4: Kuvaus käyttötapauksesta.

4.5 Sovelluksen rakenne ja rajapinnat

Sovelluksen rakenne on kuvan 5 mukaisesti kerroksellinen. Kuvassa esitetään yksinkertaisuuden vuoksi vain tekstinkäsittely- ja esitysgrafiikkatarkastimet.



Kuva 5: Sovelluksen rakenne.

Ohjain välittää käyttöliittymään syötetyt tiedot ja tarkastettavat vaatimukset oikealle tarkastimelle. Ohjain pyytää tarkastettavat vaatimukset vaatimukset-oliolta. Tarkastuksen jälkeen ohjain välittää palautteen käyttöliittymälle.

Vaatimukset-olio hakee harjoitustyön vaatimukset tekijän sähköpostin mukaan nimetystä tiedostosta. Olio koostaa yhteen tiedoston sisältämät useat vaatimukset.

Tekstinkäsittelytarkastin ei ota kantaa siihen, kumpaa tiedostoformaattia (docx, odt) tarkastettava tiedosto on. Se vertaa ohjaimen välittämiin vaatimuksiin tarkastettavan tiedoston arvoja, jotka se saa alemmalta tarkastimelta.

Esitysgrafiikkatarkastin toimii vastaavasti kuin tekstinkäsittelytarkastin eli se ei myöskään ota kantaa siihen, kumpaa tiedostoformaattia (pptx, odp) tarkastettava tiedosto on.

Tiedostomuotokohtaiset tarkastimet docx, odt, pptx ja odp hakevat halutun tiedon dokumentista ja välittää tiedon dokumenttityyppikohtaiselle tarkastimelle. Ne sisältävät dokumenttityyppikohtaisten tarkastimien käyttämien julkisten metodien lisäksi myös yksityisiä metodeja.

Yleiset aliohjelmat eivät ole sidonnaisia tiedostoformaattiin. Ne sisältävät mm. erilaisia muunnosaliohjelmia, joilla saadaan eri mittayksiköt (kuten tuumat, sentit ja pisteet) samoihin verrattaviin mittayksiköihin. Projektissa toteutettiin myös yleiseen DOM-puun käsittelyyn tarkoitettuja aliohjelmia mm. tekstin hakemiseen tekstisolmuista.

Harjoitustyön vaatimuksille määritettiin projektissa uusi **vaatimustiedosto**. Kyseinen XML-tiedosto sisältää tarkastettavan dokumentin tarkastettavat ominaisuudet vaatimuksineen ja virheilmoituksineen.

Rajapinnat on toteutettu käyttämällä samanlaisia muuttujia ja samannimisiä metodeja eri tiedostomuodoista hakevissa moduuleissa. Moduulien metodit välittävät tiedot tarkastimelle muuttujissa, joiden arvot ovat muutettu yhtenäisiksi.

Moduulit ja niiden sisältämät metodit on kuvattu tarkemmin luokkadokumentissa [1].

4.6 Käytetyt valmiskomponentit

Sovelluksen toteutuksessa käytettiin avuksi **DOM-ohjelmointirajapintaa**, joka mahdollistaa XML-dokumenttien sisällön tulkinnan ja muokkauksen. Siinä dokumentin tieto esitetään puumaisessa rakenteessa.

Sovelluksessa DOMin käytön tarve oli vähäistä, joten projektin tarpeisiin riitti kirjasto `xml.dom.minidom`. Projektin alussa päätettiin, että mikäli kirjastossa ei ole kaikkia tarvittavia toimintoja, siirrytään käyttämään varsinaista DOMia. Kirjaston funktioista sovelluksessa käytettiin seuraavia funktioita:

`childNodes`	palauttaa kaikki lapsisolmut.
`firstChild`	palauttaa ensimmäisen lapsisolmun.
`getAttribute`	antaa halutun attribuutin arvon.
`getElementsByTagName`	etsii ja listaa kaikki halutun nimiset elementit.
`hasAttribute`	tarkastaa onko solmulla kyseista attribuuttia.
`nextSibling`	palauttaa seuraavan sisarsolmun.
`parentNode`	palauttaa vanhemmansolmun.

5 Ohjelmointikäytänteet

Luvussa kuvataan ohjelmoinnissa noudatettuja käytänteitä, toteutusympäristöä ja käytettyjä työkaluja. Ohjelmointi suoritettiin yleisten Pythonin käytänteiden mukaisesti. Sovelluksen järjestelmällinen testaus jäi vähälle resurssien puutteen vuoksi.

5.1 Lähdekoodin kirjoituskäytänteet

Ryhmä noudatti seuraavia sopimiaan kirjoituskäytänteitä:

- Ei rikottu Pythonin omia kirjoituskäytänteitä, joita on määritetty PEP 8:ssa [16].
- Luokkien nimet kirjoitettiin isolla alkukirjaimella ja käytettiin CamelCase-kirjoitustapaa.
- Aliohjelmien ja muuttujien nimet kirjoitettiin pienellä alkukirjaimella ja käytettiin CamelCase-kirjoitustapaa.
- Eri tarkastimissa aliohjelmien nimet olivat yhtenäisiä.
- Eri tarkastimien muuttujien nimeämisessä otettiin huomioon tarkastettavan tiedostoformaatin omat nimeämiskäytänteet.
- Epydocilla generoitavat luokkadokumentit huomioitiin Pythonin Docstring-kommenteissa ja käytettiin PEP 257:ssä [12] määritettyjä käytänteitä.

5.2 Lähdekoodin kommentoiminen ja tiedostojen nimeäminen

Lähdekoodin kommenteissa on jokaisen tiedoston alussa tekijöiden nimet, päivämäärä sekä käytetty lisenssi. Metodien kommenteissa on ensin yleinen kuvaus yhdellä virkkeellä, jonka jälkeen on mahdollinen tarkempi kuvaus sekä parametrien ja palautusarvon kuvaus. Lisäksi metodien kommentit voivat sisältää esimerkkikoodia tiedostoformaatin XML-merkkauksesta.

Lähdekoodin kommentit kirjoitettiin englanniksi ja Epydoc-työkalua tukevaksi. Epydocilla luotiin sovelluksen lähdekoodidokumentaatio.

Lähdekoodien **tiedostot** nimettiin englanniksi. Tiedostojen ja hakemistojen nimeämisessä käytettiin vain pieniä kirjaimia. Skandinaavisia kirjaimia, kuten å, ä ja ö, ei kuitenkaan käytetty. Välilyönnit korvattiin alaviivalla.

5.3 Lähdekoodiesimerkki

Lähdekoodien käytänteitä havainnollistaa seuraava lähdekoodiesimerkki:

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
#
#The MIT License
#
#Copyright (c) 2011
#
#Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
#of this software and associated documentation files (the "Software"), to
#deal in the Software without restriction, including without limitation the
#rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
#sell copies of the Software, and to permit persons to whom the Software is
#furnished to do so, subject to the following conditions:
#
#The above copyright notice and this permission notice shall be included in
#all copies or substantial portions of the Software.
#
#THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
#IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
#AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
#FROM,OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
#IN THE SOFTWARE.
#
#Authors:
#   Vili Auvinen (vili.k.auvinen@jyu.fi)
#   Olli Kauppinen (olli.kauppinen@jyu.fi)
#   Juho Tammela (juho.i.tammela@jyu.fi)

'''
The module provides the methods for inspecting docx files.

@author: Vili Auvinen, Juho Tammela
'''

from common_methods import *
from xml.dom import minidom
from math import fabs
from conversions import convertTwipToCm, convertTwipToPt


def getStyleElementById(styleId, styleXml):
    ''' Return the style-element with a given style id.'''
    styleElements = getElements('w:style', styleXml)
    for element in styleElements:
        if (element.getAttribute('w:styleId') == styleId):
            return element

    return None
```

5.4 Kehitysympäristö

Ohjelmointiin käytettiin Eclipse-ohjelmointiympäristöä ja sen PyDev-laajennusta, joka mahdollisti Python-ohjelmointikielen kirjoittamisen Eclipsellä. Ryhmän jäsenet olivat tyytyväisiä Eclipseen. Käyttöliittymän ohjelmointiin käytettiin myös JavaScriptiä ja CSS:ä.

5.5 Testauksen käytänteet

Yksikkö- ja integraatiotestausta suunniteltiin suoritettavan aina toteutusvaiheen lopussa sen tuloksille. Pääsääntöisesti kunkin jäsenen kuului testata itse toteuttamansa ohjelmakoodin. Yksikkötestausta suoritettiin toteutusvaiheessa tiedostokohtaisille tarkastimille. Integraatiotestausta suoritettiin docx- ja odt-tarkastimien yhteisen rajapinnan toteutuksen yhteydessä.

Järjestelmätestaus suoritettiin toteutusvaiheiden jälkeen sovellusta viimeisteltäessä. Sovelluksen tarkastimet eivät ole riippuvaisia toisistaan, joten järjestelmätestaus oli kohtuullisen turvallista jättää sovelluksen viimeistelyn yhteyteen. Vastaava ohjaaja ja tilaajat koekäyttivät sovellusta heti, kun ensimmäiset toimivat tarkastukset oli toteutettu. Tämä jälkeen he koekäyttivät sovellusta aina uusien ominaisuuksien valmistuttua.

Sovellusta testattiin kunkin toteutusvaiheen lopussa oikeilla *Tietokone ja tietoverkot työvälineenä* -kurssin harjoitustöillä. Lisäksi ryhmä laati omia **testidokumentteja**, joista taulukoitiin niiden ominaisuudet ja testauksen oletetut tulokset testausraportteihin. Myös projektiryhmän laatimia projektiin liittyviä dokumentteja käytettiin paljon testimateriaalina.

Järjestelmätestauksen menetelmät ja tulokset kuvataan tarkemmin järjestelmätestauksen raportissa [11].

6 Tavoitteiden toteutuminen

Luvussa kuvataan sovellukselle määriteltyjen vaatimusten toteutuminen sekä sovelluksen tunnetut puutteet, viat ja jatkokehitysideat. Projektisuunnitelmassa [2] ja projektin alussa todettiin, ettei kaikkia vaatimuksia ehditä toteuttamaan. Sovelluksen toiminnalliset ja tekniset vaatimukset sekä niiden päivitetyt tilat ja prioriteetit esitetään vaatimusmäärittelyssä [9].

Käyttöliittymän, ohjainkomponentin ja tekstinkäsittelytarkastimen osalta projektin tavoitteet toteutuivat. Esitysgrafiikka- ja taulukkolaskentatarkastimien toteuttaminen sovittiin tilaajan kanssa siirrettäväksi projektin jälkeiseen jatkokehitykseen. Näihin formaatteihin kuitenkin tutustuttiin ja niistä laadittiin lyhyet esittelydokumentit. Sovelluksen rakenteessa ja toiminnassa on huomioitu puuttuvien tarkastimien lisääminen jatkokehityksessä.

6.1 Sovellukselle asetetut tavoitteet

Projektissa toteutettavan sovelluksen ensisijaisena tavoitteena oli pystyä **ohjelmallisesti tarkastamaan tekstinkäsittelydokumenttien ja esitysgrafiikkaesitysten rakenteita ja muotoiluja** *Tietokone ja tietoverkot työvälineenä* -kurssin vaatimuksiin. Sovelluksen tuli siis tukea sekä Microsoft Officen docx- ja pptx-tiedostoja että OpenOfficen odt- ja odp-tiedostoja. Alemman prioriteetin tavoitteita olivat edellä mainittujen toimisto-ohjelmistojen taulukkolaskentadokumenttien ja WWW-sivujen rakenteiden tarkastaminen.

Sovellukseen tuli toteuttaa **WWW-käyttöliittymä**, jolla kurssin opiskelijat voivat itse tarkastaa harjoitustöitään. Sovellus käy läpi heidän harjoitustyönsä dokumentit, vertaa niitä kurssin vaatimuksiin ja antaa lopuksi palautetta. Kurssin opettajat tulevat myös käyttämään sovellusta harjoitustöiden tarkastamisen tukena.

Sovelluksen toteutuksessa tuli myös ottaa huomioon se, että sovellusta halutaan tulevaisuudessa laajentaa, muokata tai räätälöidä muihin tarkoituksiin. Näitä ovat muun muassa harjoitustöiden arvostelu perustuen tarkastettavien vaatimusten pisteytykseen sekä muiden (kurssin ulkopuolisten) dokumenttien tarkistaminen.

Projektin edetessä huomattiin, että projektin puitteissa ehditään toteuttamaan ainoastaan tekstinkäsittelyn tarkastimet. Sovelluksen osien toteutusjärjestystä muutettiin siten, että tekstinkäsittelytarkastimien toteutus nousi tärkeimmäksi tavoitteeksi ennen käyttöliittymää. **Sovellusosiot** toteutettiin Parsi-projektissa ja toteutetaan sen jälkeisessä jatkokehityksessä siis **seuraavassa järjestyksessä**:

1. tekstinkäsittelydokumenttien tarkastimet sisältäen
   1.1. docx (Microsoft Word) ja
   1.2. odt (OpenOffice.org Writer),
2. käyttöliittymä, ohjain ja tekstinkäsittelyn vertailukomponentti,
3. esitysgrafiikkadokumenttien tarkastimet sisältäen
   3.1. pptx (Microsoft PowerPoint) ja
   3.2. odp (OpenOffice.org Presentation),

4. pisteytys,
5. taulukkolaskennan tarkastimet sisältäen
   5.1. xlsx (Microsoft Excel) ja
   5.2. ods (OpenOffice.org Calc) sekä
6. WWW-sivujen tarkastimet.

Em. tavoitteista Parsi-projektissa toteutettiin kaksi ensimmäistä.

6.2 Sovelluksen tavoitteiden toteutuminen

Projektissa ehdittiin toteuttamaan ainoastaan **osa tilaajan sovellukselle asettamista tavoitteista**, koska tekstinkäsittelytarkastimien toteutus osoittautui yllättävän työlääksi. Lisäksi kurssin harjoitustyön vaatimukset lisääntyivät ja tarkentuivat projektin edetessä, sekä tarvittavan vaatimustiedoston määrittäminen siirtyi ryhmän vastuulle. Ryhmä toteutti sovellukseen tekstinkäsittelytarkastimen, käyttöliittymän ja ohjainkomponentin.

Harjoitustyön vaatimukset sisältävä tiedosto muuttui koko projektin ajan. Alkuperäinen tiedosto hylättiin projektin puolivälissä, koska sen määrittelytapa ei tukenut sovellusta. Ryhmä kehitti tilaajan ja teknisen ohjaajan tukemana kokonaan uuden tiedoston. Uuden tiedoston luonti ja sen käsittely yhdessä muiden XML-formaattien kanssa toi omat haasteensa, koska kolmesta erilaisesta tiedostomuodosta löytyviä tietoja täytyi pystyä vertaamaan keskenään.

Sovelluksen arkkitehtuuria oli vaikea suunnitella etukäteen projektin ja sovelluksen luonteen vuoksi. Ryhmän toteuttamaan runkoon on kuitenkin helppo jatkokehityksessä lisätä esitysgrafiikka-, taulukkolaskenta- ja WWW-dokumentin tarkastimet.

Ryhmä otti sovelluksen toteutuksessa huomioon sen, että sovellusta halutaan tulevaisuudessa laajentaa, muokata tai räätälöidä tarkastamaan dokumentteja erilaisiin tarpeisiin. Sovelluksella ei voida tarkastaa ilman muutoksia kurssin ulkopuolisia dokumentteja johtuen kurssin harjoitustöiden vaatimusten luonteesta.

Harjoitustyön vaatimusten pisteytys sovittiin projektin jälkeiseen jatkokehitykseen. Projektin aikana ryhmällä ei ollut edes mahdollisuutta toteuttaa pisteytystä, koska tilaajallakaan ei ollut tarkkaa kuvaa vaatimusten arvostelusta. Pisteytyksen lisääminen onnistunee vaivattomasti harjoitustyön vaatimukset sisältävään tiedostoon tai erilliseen tiedostoon, josta ne haetaan tietyn vaatimuksen tai virheen tunnisteella. Pisteytystä hyödynnetään kurssin arvosanoja annettaessa.

6.3 Testauksen tulokset

Järjestelmätestaus suoritettiin luodun suunnitelman mukaisesti. Testaukseen käytettiin tiedostoja `Yliopisto_opiskelu.odt` ja `kaikki_oikein.docx`. Testauksessa testattiin ainoastaan sovellukseen toteutettuja vaatimuksia. Testattavia vaatimuksia oli 29 kappaletta ja virheitä löytyi yksi kappale.

Virhe syntyi tiedoston `Yliopisto_opiskelu.odt` otsikkotyylien tarkastuksessa. Virhettä ei korjattu projektin aikana, vaan se korjataan jatkokehityksessä. Virheen korjaukseen on tarkemmat ohjeet luvussa 6.4.

Järjestelmätestauksen menetelmät ja tulokset kuvataan tarkemmin järjestelmätestauksen raportissa [11].

6.4 Puutteelliset ja virheelliset toteutusratkaisut

Sovellus sisältää joitain puutteellisia ja virheellisiä toteutusratkaisuja. Luvussa kuvattavia korjausehdotuksia ei toteuteta projektin aikana, vaan ne on kirjattu tiedoksi jatkokehittäjille.

Eri merkistöjen tukea ei ole toteutettu sovelluksessa parhaalla mahdollisella tavalla. Viimeistelyvaiheessa löytyi uusia ongelmia merkistöjen kanssa. Jatkokehittäjien kannattaa tarkistaa, etteivät viat johdu Windowsin ja Linuxin käytöstä ristiin.

Sovelluksen tiedostoja ei ole nimetty kuvaavimmalla tavalla. Tiedosto `Word_processing.py` kannattaisi nimetä uudelleen esimerkiksi `text_document_inspector.py`. Vastaavasti esitysgrafiikkatarkastin voisi olla `presentation_document_inspector.py` ja taulukkolaskentatarkastin `spreadsheet_document_inspector.py`. Em. nimistä `document`-sanan voisi jättää pois, jotta tiedostonimet olisivat lyhyempiä. Tiedostojen `docx_inspector.py` ja `odt_inspector.py` nimet voi säilyttää sellaisenaan, tai ne voi vaihtaa muotoon `docx_checker.py` ja `odt_checker.py`, koska ne vain hakevat tietoa.

Etusivulla syötetyn **sähköpostiosoitteen tarkastus** on toteutettu etsimällä `@`-merkin sisältävää sanaa. Sähköpostiosoite pitäisi kuljettaa syötesivulta tarkastimelle ja etsiä, löytyykö kyseinen osoite kansisivulta.

Sovellus kaatuu, mikäli **harjoitustyön vaatimukset sisältävä XML-tiedosto on virheellinen**. Jos jatkokehityksessä toteutetaan vaatimustiedostojen generointityökalu, ja se toimii oikein, ei tarkastuksia tarvitse tehdä. Mikäli työkalussa on puutteita tai tiedostoja muutetaan käsin, täytyy tarkastuksia tehdä, jotta sovellus olisi vakaa.

Moduulin `odt_inspector.py` metodi `checkHeadersAndFooters` tarkastaa sisällysluettelon ja varsinaisen tekstiosion **sivunumeron numerointitavan** eron ainoastaan alatunnisteesta. Vertailu on vaikea toteuttaa vertaamalla suoraan eri tunnisteiden sisältämiä sivunumeroita, koska ylä- ja alatunnisteessa voi olla eri numerointitapoja. Yksi vaihtoehto on määrätä harjoitustyön vaatimuksissa, että sivunumeron on oltava alatunnisteessa.

Pitkillä dokumenteilla tarkastaminen kestää kauan. Yli sadan sivun dokumentin tarkastaminen kestää useita minuutteja. Pitkissä dokumenteissa on tarkastettavia tekstikappaleita niin paljon, että niiden useaan kertaan läpikäyntiin menee aikaa. Dokumentin tekstikappaleista tarkastettavien merkkien ja sanojen etsimistä voi yrittää lisätä listaan ja tarkastaa yhdellä kertaa. Toisaalta kurssin osalta tarkastettavat harjoitustyöt ovat lyhyitä, ja niiden tarkastaminen ei kestä liian kauaa, joten optimoinnilla ei ole kiirettä.

Käytettäessä **Odt-formaatissa Arial-fonttia** saattaa tulla XML-tiedostoon fontin nimeksi *Arial1* tai *Arial2*, vaikka Writer-ohjelma näyttää *Arial*-nimeä. Muiden fonttien kanssa tätä ongelmaa ei ole projektin aikana esiintynyt. Ongelmaan ei ehditty syventyä tarpeeksi projektin aikana, koska se löytyi sovelluksen viimeistelyvaiheessa. Kirjasinta lihavoitaessa ja kursivoitaessa nimi muuttui välillä, mutta mitään pitävää logiikkaa ei löydetty.

Käyttöliittymän toteutuksessa luokka `gui.py` palauttaa selaimelle merkkijonon, joka sisältää HTML-koodia. Lähdekoodipaketista löytyy kuitenkin myös tiedosto `gui_DOM.py`, joka sisältää käyttöliittymän toteutuksen DOM-rajapintaa käyttäen. Merkistöongelmien takia ei kuitenkaan voitu projektin aikana siirtyä käyttämään sitä.

Palautteen lähettämistä sähköpostina kannattaa muuttaa siten, että sähköpostiviesti koostetaan palvelimella. Tällä hetkellä viesti koostetaan JavaScriptissä tiedostossa `feedback_sender.js`.

Käyttöliittymän lomake kannattaa **POST-menetelmän** sijaan muuttaa käyttämään **GET-menetelmää.** Tällöin voidaan paremmin tallentaa ja hyödyntää käyttäjän syöttämiä tietoja ja valintoja.

Palautteessa ei esitetä **tarkastettavan dokumentin viimeisen muokkauksen ajankohtaa**. Ajankohdan näyttäminen helpottaisi käyttäjää varmistamaan tarkastetun version oikeellisuus. Viimeisen muokkauksen ajankohdan hakeminen on toteutettu sovelluksessa, mutta sen parsiminen oikeaan muotoon ja kuljettaminen palautesivulle on toteuttamatta.

6.5 Jatkokehitysideat

Projektin jälkeiseen jatkokehitykseen sovittiin tilaajan kanssa **esitysgrafiikka-, taulukkolaskenta- ja WWW-sivujen tarkastimet**. Esitysgrafiikka- ja taulukkolaskentatarkastimet pystytään lisäämään suoraviivaisesti tekstinkäsittelytarkistimen rinnalle. WWW-sivujen tarkastamista ei käsitelty projektin aikana, joten siihen liittyviä ideoita ja huomioita ei ollut mahdollista kirjata.

Harjoitustyön vaatimuksille esitysgrafiikan ja taulukkolaskennan osalta kannattanee **lisätä omat tiedostot**. Tällöin kukin dokumenttitarkastin käsittelee yhtä harjoitustyön vaatimustiedostoa. Tällöin tiedoston `controller.py` metodia `beginInspection` on muokattava siten, että sovellus hakee harjoitustyön vaatimukset tiedostomuodon mukaan.

Virheilmoitusten tarkempi määrittäminen sovittiin tilaajan vastuulle jatkokehitykseen. **Ryhmän toteuttamat virheilmoitukset ovat suuntaa-antavia**. Kurssin opettajina toimivat tilaajan edustajat pystyvät laatimaan opiskelijoille yksikäsitteiset virheilmoitukset. Myös virheisiin liittyvät **korjausneuvot ja linkit ohjesivuihin** onnistuvat tilaajilta paremmin. Virheilmoitukset kannattanee erottaa vaatimuksista, koska jokaiselle opiskelijalle toimitetaan henkilökohtaiset vaatimustiedostot, mutta virheilmoitukset ovat henkilöstä ja vaatimustiedostosta riippumattomat. Lisäksi virheilmoitusten päivitys ja hallinta on paljon helpompaa yhdestä tiedostosta.

6.6 Toteutusratkaisujen kehittyminen

OpenOffice.orgin ja Microsoft Officen erilaisia **tiedostomuotoja lähdettiin tarkastelemaan ensin erillään**. Odt-tiedostojen tarkastaminen aloitettiin kaksi viikkoa aikaisemmin kuin docx-tiedostojen, joten odt-tiedoston rakenne oli suurelta osin jo tiedossa ennen kuin docx-tiedostojen tarkastinta alettiin kehittämään. Tarkastimien poikkeavuudet aiheuttivat yhdessä arkkitehtuurin suunnittelun venymisen kanssa sen, että koodia piti yhdenmukaistaa ja muuttaa aika paljon vielä viimeisessä toteutusvaiheessa.

Arkkitehtuuri suunniteltiin projektin alussa toteutettavaksi siten, että tiedostomuotokohtaiset `odt_inspector.py` ja `docx_inspector.py` toteuttavat tarkastuksen. Projektin aikana päädyttiin lisäämään ohjelmakohtainen tarkastin `word_processing.py` ohjaimen ja tiedostomuotokohtaisten tarkastimien väliin. Sen tehtävänä on suorittaa varsinainen vertailu harjoitustyön vaatimuksiin, kun tiedostomuotokohtaiset tarkastimet hakevat vain tiedot vertailtavaksi. Syynä lisäykseen oli kummallekin tiedostomuodolle yhteiset vaatimukset ja vertailevan moduulin toteuttaminen yleiskäyttöisemmäksi. Tällöin vertailevan moduulin ei tarvitse ottaa kantaa siihen, mitä tiedostomuotoa käsitellään.

7 Käyttöohjeita ylläpitäjälle

Luvussa kuvataan lyhyesti, miten sovellukseen voidaan lisätä uusia tarkastettavia kohteita ja dokumenttityyppejä. Projektissa toteutettu sovelluksen versio soveltuu vain *Tietokone ja tietoverkot työvälineenä* -kurssin dokumenttien tarkastamiseen.

7.1 Uuden tarkastettavan kohteen lisääminen

Harjoitustöiden vaatimustiedosto määrittää kaikki dokumentista tarkastettavat asiat. Uusi tarkastettava kohde saadaan lisäämällä vaatimustiedostoon kohdetta vastaava XML-määrittely ja toteuttamalla kyseinen tarkastusmetodi sovellukseen. Toisaalta esimerkiksi tarkastettavia tyylejä voi lisätä sovellukseen pelkästään määrittämällä uuden tyylin vaatimustiedostoon, koska tyylien tarkastaminen on jo toteutettu sovelluksessa.

Mikäli haluaa lisätä kokonaan **uuden tarkastettavan vaatimuksen,** on sovelluksen koodia muokattava. Dokumenttikohtaiseen tarkastimeen on lisättävä metodi, jossa määritellään formaattikohtaisen metodin nimi. Lisäksi `_checkers`-muuttujaan on lisättävä metodin nimi. Kumpaankin tiedostomuotokohtaiseen tarkastimeen lisätään saman niminen metodi, jotka palauttavat samalla tavalla tarkastettavan arvon.

`_checkers`-muuttuja on hakemisto, jolla on avain-arvo -pareja. Seuraavassa esimerkissä tarkastimella on vain yksi tarkastettava vaatimus:

```
_checkers = { 'Styles': checkRequirementStyles}
```

Esimerkin `Styles`-avainta vastaava vaatimus löytyy vaatimustiedostosta ja `checkRequirementStyles`-metodi suorittaa tietojen vertailun.

7.2 Uuden dokumenttityypin lisääminen

Uuden tuettavan dokumenttityypin lisääminen vaatii uuden metodin luomista kumpaankin tiedostomuotoa tarkastavaan moduuliin ja lisäksi yhteisen metodin dokumenttityyppikohtaiseen moduuliin. Lisäksi lähdekoodiin tulee lisätä `controller`-moduulin `checkZipFiles`-metodiin tiedostopäätteen mukainen ohjaus uuteen dokumenttikohtaiseen tarkastimeen. Tarkastettavien vaatimusten lisäämistä uudelle dokumenttityypille käsiteltiin myös jatkokehitysideoissa luvussa 6.5.

7.3 Sovelluksen siirtäminen uuteen käyttöympäristöön

Mikäli sovellusta halutaan käyttää kurssin ulkopuolisten dokumenttien tarkastamiseen, on tehtävä isoja muutoksia lähdekoodiin. Ensinnäkin tarkastettavat asiat sisältävä vaatimustiedosto on luotava uudestaan. Yleiseen käyttöön soveltuvat joidenkin tarkastettavien kohteiden metodit, kuten paperinkoon, marginaalien, sisällysluettelon, ylimääräisten välien ja tyylien tarkastaminen. Lähdekoodia muokkaamalla saa yleiskäyttöisiksi muitakin tarkastettavia asioita, kuten ylä- ja alatunnisteen sekä kansilehden tarkastamisen.

Osa sovelluksen tarkastamista vaatimuksista on kurssin oppimistavoitteista johtuen ainoastaan kurssin käyttöön soveltuvia. Näitä ovat mm. vaatimukset yhden kuvan löytymisestä, listojen muodostamisesta asteriskin avulla ja vähintään kolmen osion sisällyttämisestä.

8 Yhteenveto

Projekti toteutti prototyypin sovelluksesta, jolla voidaan tarkastaa OpenOffice.orgin ja Microsoft Officen toimisto-ohjelmistoilla kirjoitettujen dokumenttien rakennetta ja muotoilua. Sovellus vertaa dokumenttien tietoja sille annettuihin vaatimuksiin ja antaa käyttäjälle palautteen löydetyistä virheistä.

Sovelluksen vaatimuksista ehdittiin toteuttamaan projektin aikana tekstinkäsittelydokumenttien tarkastaminen. Esitysgrafiikka- ja taulukkolaskentadokumenttien sekä WWW-sivujen tarkastimet sovittiin tilaajan kanssa jatkokehitykseen. Sovelluksen toiminnalliset ja tekniset vaatimukset sekä niiden toteutuminen ja priorisointi on kuvattu tarkemmin vaatimusmäärittelyssä [9].

Sovellukseen on lisättävä puuttuvat esitysgrafiikka-, taulukkolaskenta- ja WWW-dokumenttien tarkastaminen, jotta se voidaan ottaa käyttöön kurssilla. Tiedostomuodoista luodut esittelydokumentit [4], [5], [6], [7], [8] ja [10] tukevat jatkokehittäjää tiedostojen sisällön tulkitsemisessa. Sovelluksen antamat palautteet kaipaavat sisällöllistä muokkausta sekä ohjeiden lisäystä.

Sovelluksessa ilmenneet puutteelliset tai vialliset toteutukset on suhteellisen helposti korjattavissa jatkokehityksessä. Vain merkistö- ja optimointiongelmista ei ole valmiita korjausratkaisuja.

Lähteet

[1] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, Luokkadokumentit", saatavilla <URL: http://sovellusprojektit.it.jyu.fi/luokkadokumentit/html>, Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[2] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, Projektisuunnitelma", saatavilla PDF-muodossa <URL: http://sovellusprojektit.it.jyu.fi/parsi/dokumentit/projektisuunnitelma/projektisuunnitelma_1.0.0.pdf>, Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[3] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, Projektiraportti", saatavilla PDF-muodossa <URL: http://sovellusprojektit.it.jyu.fi/parsi/dokumentit/projektiraportti/projektiraportti_0.3.0.pdf>, Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[4] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, MS Office XML-formaatti: Excel", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[5] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, MS Office XML-formaatti: PowerPoint", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[6] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, OpenOffice XML-formaatti: Calc", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[7] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, OpenOffice XML-formaatti: Impress", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[8] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, OpenOffice XML-formaatti: Writer", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[9] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, Vaatimusmäärittely", saatavilla PDF-muodossa <URL: http://sovellusprojektit.it.jyu.fi/parsi/dokumentit/vaatimusmaarittely/vaatimusmaarittely_0.5.0.pdf>, Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[10] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, MS Office XML-formaatti: Word", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[11] Auvinen Vili, Kauppinen Olli ja Tammela Juho, "Parsi-projekti, Järjestelmätestausraportti", Jyväskylän yliopisto, tietotekniikan laitos, 2011.

[12] Goodger David, "Docstring conventions", saatavilla <URL: http://www.python.org/dev/peps/pep-0257/>, Pyhon Software Foundation, viitattu 21.3.2011.

[13] Hänninen Tero, Nieminen Juho, Peltola Marko ja Salo Heikki, "Verso-projekti, Application Report", saatavilla PDF-muodossa <URL: http://sovellusprojektit.it.jyu.fi/verso/dokumentit/application_report_1.0.pdf>, Jyväskylän yliopisto, tietotekniikan laitos, 14.12.2010.

[14] Kumpulainen Tuomas, Tuurihalme Kari, Valkama Outa ja Virtanen Tuomas, "Tabu-projekti, Sovellusraportti", Jyväskylän yliopisto, tietotekniikan laitos, 10.8.2009.

[15] Santanen Jukka-Pekka, "Tietotekniikan sovellusprojektien ohje", Jyväskylän yliopisto, tietotekniikan laitos, 11.9.2006.

[16] van Rossum Guido ja Warsaw Barry, "Style Guide for Python Code", saatavilla <URL: http://www.python.org/dev/peps/pep-0008/>, Pyhon Software Foundation, viitattu 21.3.2011.