Muistio: Parsi-projektin 1. koodikatselmointi

Aika: keskiviikkona 22.3.2011 klo 12:23 - 14:21
Paikka: sovellusprojektien kokoushuone Ag C226.2, Jyväskylän yliopisto

Paikalla
  Projektiryhmä
        Olli Kauppinen, puheenjohtaja
        Vili Auvinen
        Juho Tammela, sihteeri
  Tilaajat
        Tommi Lahtonen
  Ohjaajat
        Jukka-Pekka Santanen
        Mikko Tyrväinen

Aloitettiin koodikatselmointi. Käytiin lähdekoodia läpi komponentti kerrallaan ja esitettiin siitä huomioita.

controller.py

Importit voivat olla muotoa from zipfile import iszipfile, jos paketista ei käytetä muita funktioita. Se selkeyttää koodia.

Tiedostojen eli luokkien ja moduuleiden kommentit heti alusta alkaen yhtenäisiksi. Aliohjelmien alkuun kuvaavat kommentit.

checkZipFiles-aliohjelmassa kannattaa kaivaa tiedostopääte heti alussa, jotta sitä ei tarvitse etsiä erikseen joka vaiheessa.

getFileExtension-aliohjelmassa riittää yhden rivin return, ei tarvitse käyttää if-lausetta. Lisäksi kannattaa muuttaa merkkijono pieniksi kirjaimiksi, sillä tiedoston nimi voi olla kirjoitettu isoilla kirjaimilla. Kommentti on puutteellinen.

word_processing.py

Tarkistimet ja vaatimukset ovat olioita, mutta niiden ei välttämättä tarvitsisi olla. Periaatteessa word_processing voisi tehdä saamastaan tiedostosta esimerkiksi yleisen document-olion ja inspectorit voisivat olla pelkkiä moduuleita.

main-aliohjelman nimen voi vaihtaa.

word_processing voisi palauttaa palautteen Dom-puuna tai muulla tavalla ja antaa sen ohjainkomponentille, eikä tulosta mitään.

requirements.py

Ei tarvitse välttämättä olla olio.

Vaatimukset sisältävä xml-tiedostoformaatti pitäisi kirjoittaa uudelleen. Lahtonen voi uusia sen. Yleisten vaatimusten lisääminen tiedostoon voi olla haastavaa. Vaatimukset sisältävän xml-tiedoston tarkkaa muotoa harkitaan vielä, mutta tehdään siitä yleiskäyttöisempi.

Yleinen huomio: __init__ -metodia ei kannata kutsua konstruktoriksi, koska se ei täsmälleen ottaen ole sellainen.

odt_inspector.py

Ei tarvitse välttämättä olla olio.

Olisi hyvä jos odt_inspectorilla ja docx_inspectorilla olisi identtinen rajapinta. Silloin word_processing-komponentin ei edes tarvitse tietää mikä tekstinkäsittelyformaatti on kyseessä.

Try - except niihin paikkoihin, joissa voi tulla poikkeuksia.

Olion tyyppiä voi tutkia hasAttr -metodilla. Se tarkastaa onko muuttujalla olemassa joku tietty metodi. Esimerkiksi str.hasAttr(split).

Poikkeuksia voi käsitellä myös hasAttr-metodilla. Riippuu tilanteesta kannattaako käyttää try-except -rakennetta poikkeuksille vai jotain muuta keinoa.

checkHeaderAndFooter-aliohjelman voisi pilkkoa kahteen eri aliohjelmaan, jos mahdollista. Koska vaatimus on, että sivunumero ja tekijä on joko ylä- tai alatunnisteessa, niin tarkastus on nyt samassa aliohjelmassa. Aliohjelma voisi kuitenkin käyttää apufunktiota, sillä ylä- ja alatunnisteen tutkiminen on toteutettu melkein samalla tavalla.

Vaikka eri inspectoreilla onkin identtinen rajapinta, niillä voi silti olla "privaatteja" aliohjelmia. Privaattien aliohjelmien nimen ensimmäiseksi merkiksi kirjoitetaan alaviiva _.

Xpathia voi käyttää joissain tapauksissa. Jos Xpathia käyttää, sen toiminnan voi kommentoida viereen.

findEmptyParagraphs-aliohjelma ilmoittaa esimerkiksi tyhjästä solusta taulukossa. Pitää siis tarkastaa onko tyhjä kappale esimerkiksi taulukon sisällä. Lisäksi aliohjelma ei välttämättä huomaa, että kappale on tyhjä jos kappaleen sisältö on esimerkiksi välilyönti tai tabulaattori.

listUsedStyles-aliohjelma hakee nyt vain tietyt kovakoodatut tyylit. Voisi tehdä myös sellaisen aliohjelman, joka hakee ihan kaikki käytetyt tyylit ja listaa ne. Jos tehdään erillinen opettajaversio, tällaiset tiedot voisivat olla kiinnostavia.

findStringFromParagraphs-aliohjelmalle voisi antaa parametrina sen elementin, josta halutaan tarkistaa tiettyä merkkiä tai merkkijonoa.

Yleinen kommentti: kannattaa harkita että tekee aliohjelmista yksikkömuotoisia ja jos halutaan, kutsutaan sitä vain useita kertoja.

printParagraphStyleAttributes-aliohjelmassa kovakoodatut oletukset voisi lisätä suoraa dictiin ja käyttää aaltosulkusyntaksia. While-silmukka vaatii vähintään kommentin siitä, mitä se tekee. Ei ole tällaisenaan helposti luettava. Yhden tyylin selvittämiseksi tarvitaan vain yksi dict, johon aina ylikirjoitetaan ylemmän tason tyylimääritykset.

Yleinen huomio: vaatimusmäärittelyn mukaan voisi suunnitella docx- ja odt-tarkastimien yhteisen rajapinnan.

docx_inspector.py

Ei tarvitse välttämättä olla olio.

Sisältää mso_meta_inspectorin kanssa samoja funktioita, mutta se on tiedossa.

localName-attribuutin kanssa kannattaa varoa mahdollisia konflikteja.

Jos meta-tiedot ovat Microsoft Officella ja OpenOfficella erilaiset, eikä samanlaista rajapintaa ole helppo toteuttaa, tehdään vain aliohjelma joka palauttaa kaikki metatiedot kerralla.

Dokumenttien tyylien oletusarvoja joutuu kovakoodaamaan.

Koodin selkeyttämiseksi sisäkkäisiä silmukoita voi jakaa aliohjelmiin. Voi myös tehdä yleiskäyttöisiä aliohjelmia, jotka esimerkiksi hakevat kaikki jonkun elementin kolmannen tason lapset. Myös Xpathilla voi saada sisäkkäisiä silmukoita vähennettyä.

Microsoft Officessa käsin tehdyn ja automaattisesti luodun sisällysluettelon erot täytyy vielä selvittää tarkemmin.

ooo_meta_inspector.py

Kannattaa huomioida, että fnmatch-funktio voi käyttää sekä isoja että pieniä kirjaimia.

mso_meta_inspector.py

Jos tulee enemmän getElementValue-aliohjelman tapaisia geneerisempiä aliohjelmia, ne voi kerätä omaan moduuliin.

conversions.py

Lukujen pyöristykseen voisi käyttää valmiita funktioita, niin ne ovat luettavampia. Funktioille täytyy lisätä virheenkäsittelyä, esimerkiksi jos funktioille syötetään merkkijonoja. Parempi on kuitenkin, että sovellus kaatuu kylmästi jos tulee väärä syöte, eikä piilota virhettä toimimalla näennäisesti.

Funktioiden kirjoitusjärjestyksellä ei sinänsä ole väliä, kunhan ryhmä käyttää yhteinäistä tapaa.

Päätettiin koodikatselmointi.