T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool
Informaatika eriala

Geily Niinemets

XML põhine klient-server kasutajaliideste arhitektuur

Bakalaureusetöö

Juhendaja: Jaak Vilo

Autor: ..... "....." mai 2004
Juhendaja: ..... "...." mai 2004
Õppetooli juhataja: ..... "...." mai 2004

Tartu 2004

Sisukord

Sissejuhatus

Kuni viimase ajani olid veebirakendustes segamini nii programmi kui ka kasutajaliidese loogika. Sellisel kujul on aga rakendusi üsna raske hallata.

W3C standardid pakuvad nüüd piisavalt võimalusi rakenduste kasutajaliidese ehitamiseks.

XML'i kasutades on lihtne luua keel kasutajaliideste defineerimiseks. See aga võimaldab kirjeldada eraldi kasutajaliidese komponendid ja sisu, mille saab XSLT ja lisaprogrammide abil teisendada kasutajaliideseks.

Käesolev bakalaureusetöö sisaldab endas XML põhise kasutajaliidese arhitektuuri kirjeldust ning erinevaid lahendusi.

Põhieesmärgiks on ära kirjeldada Expression Profiler: Next Generation kasutajaliidese arhitektuur, kuid samas tuua ka näiteid teiste samalaadsete arhitektuuride kohta. Ühesõnaga töö annab ülevaate ka Mozilla platvormile loodud XUL'ist (*XML User-interface Language*) ja bioinformaatika spetsialistidele mõeldud Talisman'ist.

Esimene peatükk selgitab antud töö juures olulisi mõisteid. Ülevaade antakse XML'ist, XSLT'st ja ka XML põhisest kasutajaliidesest lähemalt.

Teine ja kolmas peatükk räägivad vastavalt XUL'ist ja Talisman'ist ning elementaarsete kasutajaliidese elementide loomisest nendes.

Viimane peatükk räägib lühidalt, mis Expression Profiler üldse on, kirjeldab ära selle graafilise kasutajaliidese arhitektuuri ning seletab, kuidas seda luua.

Peatükk 1

Mõisted

1.1 XML

XML (*eXtensible Markup Language*) on märgendkeel, mis defineeriti W3C (*World Wide Web Consortium*)[1] XML töögrupi poolt. See on standardne andmete esitamise süntaks, mis on arusaadav nii masinatele kui inimestele.

Põhilised kasutusvaldkonnad on:

- andmevahetus;
- dokumendi salvestus;
- XML'il põhinevate keelte defineerimine.

XML dokument koosneb sarnaselt HTML'ile põhiliselt elementide hierarhilisest hulgast ja nende atribuutidest, kuid XML'is ei ole elementide hulk ette defineeritud. Samas, kui mingi kindla probleemivaldkonna jaoks on sobiv komplekt silte (*tags*) olemas, siis on selles valdkonnas rakenduste loomine väga mugav.

Põhireeglid

- Kõigil XML elementidel peab olema ka lõpusilt;

  `<silt></silt>` või `<silt />`

[1] http://www.w3.org

- XML sildid on tõstutundlikud;
- Kõik XML sildid peavad olema korralikult pesastatud (*nested*);
- Atribuutide väärtused peavad olema jutumärkide vahel;
- XML'is säilitatakse tühikud;
- Igal XML dokumenil peab olema defineeritud üks juurelement;
- Kommentaarid: `<!-- See on kommentaar -->`

XML failid peaksid (ei ole kohustuslik) algama XML'i deklaratsiooniga: `<?xml version="1.0">`.

Deklaratsiooni sees võivad olla ka atribuudid `standalone` ja `encoding`. Esimene neist näitab, kas dokument on kirjeldatud tervikuna selles failis ning teine määrab ära, millist kodeeringut dokumendis kasutatakse.

Näide tudengi kirjeldamiseks:

```
<?xml version="1.0">
<!-- tudengi kirjeldus -->
<tudeng eriala="informaatika">
  <name>
   <eesnimi>Geily</eesnimi>
   <perekonnanimi>Niinemets</perekonnanimi>
  </name>
  <matr_nr>A01998</matr_nr>
</tudeng>
```

Esitatud näites esimene rida on XML'i deklaratsioon, seejärel on kommentaar. Järgmiseks defineeritakse juurelement `tudeng`, mille on atribuut `eriala`, väärtusega `informaatika`. Defineeritakse ka tudengi `nimi` (`eesnimi` ja `perekonnanimi`) ning matriklinumber, mille väärtused on vastavalt `Geily`, `Niinemets` ja `A01998`.

1.2 XSLT (XSL Transformations)

XML dokumendile on vaja lisaks midagi, mis näitab kuidas seda dokumenti kuvada. Selleks on XSL (*eXtensible Stylesheet Language*). XSL loodi, kuna tekkis vajadus XML-põhise laaditabeli (*stylesheet*) keele järele.

XSL koosneb kolmest osast:

- XSLT (XSL Transformations) - keel XML dokumentide muundamiseks;
- XPath (XML Path Language) - keel XML dokumendi osade defineerimiseks;
- XSL-FO (XSL Formatting Objects) - keel XML dokumentide vormindamiseks.

Seega XSLT on mõeldud XML dokumentide muundamiseks teisteks XML dokumentideks või mingiks teist tüüpi dokumendiks, mis on brauseri poolt tunnustatud. XSLT saab ka lisada või eemaldada elemente väljundfailist, elemente ümber paigutada, sorteerida, testida, otsustada, milliseid elemente üldse kuvada jne. Võib öelda, et XSLT muundab XML lähtepuu (*source tree*) tulemipuuks (*result tree*).

Korrektne laaditabeli deklaratsioon

Kuna XSLT laaditabel on ise XML dokument, siis laaditabel algab XML'i deklaratsiooniga:

```
<?xml version="1.0>
```

Seejärel on vaja dokument kindlasti nimetada XSLT laaditabeliks. Selleks on juurelemendid `<xsl:stylesheet>` ja `<xsl:transform>`. Nendest võib kasutada ükskõik kumba. Seega korrektne viis XSL laaditabeli deklareerimiseks on vastavalt W3C XSLT soovitusele

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

või

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Viimane rida määrab ametliku X3C XSL soovituse nimeruumi (*namespace*). Kasutades seda nimeruumi peab lisama ka atribuudi `version="1.0"`.

(Internet Explorer 5 korral tuleb kasutada deklareerimiseks vana varianti: `<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">`.)

XSLT laaditabel koosneb reeglite komplektist, mida kutsutakse mallideks (*templates*). Malli alguse defineerib `<xsl:template>` element. Iga `<xsl:template>` element koosneb reeglitest, mida rakendatakse kui määratud XML element sobib. Malli sidumiseks XML elemendiga kasutatakse `match` atribuuti. Samuti saab seda kasutada malli defineerimiseks tervele XML dokumendi harule (`match="/"` viitab kogu dokumendile, ehk mall seostatakse XML dokumendi juurega).

XSLT elementide ja funktsioonide kirjeldused on kättesaadavad aadressil: `http://www.w3schools.com/xsl/default.asp`

1.3 XML põhine rakenduse kasutajaliides

XML'i kasutades on suhteliselt lihtne luua keel kasutajaliideste defineerimiseks. Tänu sellele on võimalik kirjeldada eraldi kasutajaliidese komponendid ja sisu, mille saab XSLT ja lisaprogrammide abil teisendada kasutajaliideseks.

Üldkasutatavat ühtset keelt ei ole veel loodud, seepärast on välja tuldud mitmete erinevate lahendustega nii veebi- kui töölauarakenduste (*desktop applications*) kasutajaliidese kirjeldamiseks.

Kõige laialt levinum on Mozilla XUL, mis ei ole küll veebipõhine, kuid on kõige kauem turul olnud kasutajaliidese kirjeldamise vahend. XUL loodi kuna eraldi kasutajapoolsete liideste ehitamine kasutades Windows GUI'd, Mac GUI'd ja GTK'd (*The Gimp Toolkit*) on üsna kulukas.

Ilmselt on XUL'i idee elujõuline, kuna hetkel arendab Microsoft välja enamvähem samadel põhimõtetel töötavat XAML'i (*eXtensible Application Markup Language*), millega tehakse ka järgmise Windows'i versiooni (Longhorn) [2] graafiline kasutajaliides.

Veebipõhiste rakenduste jaoks ei ole hetkel loodud laialt levinud kasutajaliidese arhitektuuri, kuid on olemas lahendused konkreetsete rakenduste jaoks. Nt. Euroopa Bioinformaatika Instituudi (EBI) [3] loodud Talisman ning Expression Profiler'i kasutajaliides.

[2] http://longhorn.msdn.microsoft.com/lhsdk/core/overviews/about xaml.aspx
[3] http://www.ebi.ac.uk/Information/index.html

Peatükk 2

XUL - XML User-Interface Language

Üks kõige lihtsamaid võimalusi XML-põhise kasutajaliidese loomiseks on XUL. XUL ei ole mõeldud küll veebipõhiste rakenduste loomiseks, kuid väärib sellegipoolest ära märkimist kui kõige levinum ja ilmselt kõige arenenum XML-põhine graafilise kasutajaliidese arhitektuur (terve Mozilla veebibrauseri kasutajaliides on tehtud XUL'is).

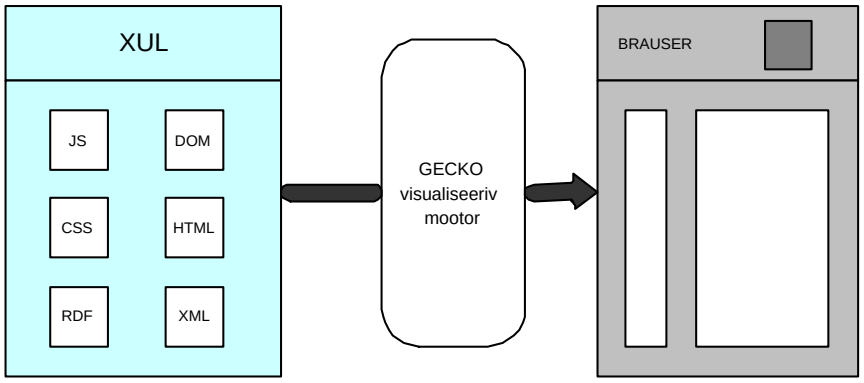
2.1 Mis on XUL?

XUL (*XML User-Interface Language*) on standarditel põhinev tarkvara kasutajaliidese defineerimiskeel. Kasutades koos XUL'i valmis tööriistu ja W3C (*World Wide Web Consortium*) standardeid (*HTML 4.0, DOM 1 ja 2, CSS 1 ja 2, XML Namespaces*) on võimalik kujundada väga erinevaid kasutajaliideseid. Samuti on võimalik kasutada JavaScripti. Tasub meeles pidada, et XUL täiendab ja toetab veebistandardeid - XUL ei dubleeri ega konkureeri veebistandarditega.

XUL töötati välja Netscape'i poolt, kuid hetkel on vastutavaks organisatsiooniks Mozilla Sihtasutus[1].

XML vajab alati jooksutamiseks parserit või mootorit. Selleks on Mozillal Gecko. Gecko on visualiseeriv mootor, mis mõistab XUL'i ja teab, kuidas seda ekraanil pikslites kuvada.

[1] http://www.mozilla.org



Joonis 2.1: XUL'i töövoog

Gecko kujundati struktuurilt kerge, standardeid järgiv ja platvormist sõltumatu. Selle asemel, et ise defineerida, kuidas kasutajaliidese elemente kuvada, sõltub Gecko kõiges muus peale kõige olulisemate käitumuslike ja esituslike aspektide CSS'ist (*Cascading Style Sheets*). Gecko visualiseerib kasutajaliideseks kombinatsiooni struktuurist, mis on defineeritud XUL'is, stiilist, mis on defineeritud CSS'is ja muudest standarditest.

Mõned elemendid, mida on võimalik XUL'is kuvada:

- menüüd: menubar, menupopup, menuitem, menuseparator;
- tööriistaribad: toolbar, toolbaritem, toolbox;
- titlebutton, progressmeter, tri-state checkbox;
- puu: tree, treecaption, treecolgroup, treecol, treehead, treechildren, treeitem, treefoot, treerow, treecell;
- tabeli vaade (tab view): tabcontrol, tabbox, tab, tabpanel, titledbutton.

2.2 XUL'i eesmärk

XUL loomise idee tekkis, kuna:

- W3C standardid pakuvad nüüd piisavalt võimalusi rakenduse kasutajaliidese ehitamiseks;

- XUL'i mootor Gecko on piisavalt kiire, et visualiseerida tarkvararakenduse kasutajaliidest reaalajas;
- XML'i kasutades on lihtne luua keel (XUL) kasutajaliideste defineerimiseks;

2.3 Rakenduse kasutajaliides XUL'is

XUL'is on iga kasutajaliidese element (*widget*) võimalik kirjeldada ühe või enama sildiga ning sama programmikood töötab üle erinevate platvormide - XUL dokument kuvatakse korrektselt nii Microsoft Windows'is, Macintoshis kui ka Linuxis. XUL dokumendi sisu oskavad kuvada Mozilla platvorm ja kõik tooted, mis on sellest tuletatud, nt. AOL Macintoshile, Netscape. Platvorm ei kasuta seejuures lisarakendusi. Selleks on vaja kasutada lihtsalt käsureal Mozilla `-chrome` võtit.

XUL'i siltide komplekt on *de facto* standard, mis on välja kasvanud reaalsete tarkvaraarendusprojektide praktilistest vajadustest. Nagu ka HTML'i puhul, saab XUL dokumente alla laadida üle kogu interneti ja kuvada kohapeal. Vajadusel saab neid installida ka kohalikku masinasse.

XUL'i nimeruumi määrab ära järgnev URL:

```
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul">
```

(Seda nimeruumi viidet kohates teab rakendus, et tegemist on XUL dokumendiga.)

XUL'i dokumendi loomisel tuleb järgida XML põhireegleid.

2.3.1 Lihtsamad kasutajaliidese elemendid

Vaatleme nuppude, tekstiväljade, raadionuppude, jne loomist XUL'is.

Kõigepealt tuleb luua XUL fail, milles on järgnevad read:

```
   <?xml version="1.0"?>
   <?xml-stylesheet href="chrome://global/skin/"
type="text/css"?>
   <window
     id="example-window"
     title="Katsetus"
```

```
  orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  ...
</window>
```

Esimene rida on XML'i deklaratsioon. Teisel real määratakse kasutatav laaditabel. Defineeritud on ka rakenduse aken, milles määratakse akna identifikaator, pealkiri, suund ja nagu juba mainitud - nimeruum. (Punktide asemele defineeritakse kasutajaliidese elemendid.)

Nupu lisamine

```
<button
  id="OKnupp"
  label="OK"
  default="true"/>,
```

kus `id` on identifikaator, mida kasutatakse antud elemendile viitamiseks; `label` määrab kirja, mis ilmub nupule; `default` atribuudi väärtus võib olla kas `true` või `false`, `true` puhul on nupp vaikimisi aktiivne (vaikimisi valitud). Reeglina tähendab see seda, et kasutaja saab nupu aktiveerida vajutades Enterit.

Nupule on võimalik lisada ka pilt. Samuti saab nuppu "ära keelata"(*disabled*).

Teksti lisamine

```
<description value="Lühike tekst"/>
```

Sellisel kujul on mõtet lisada üherealist teksti. Pikema teksti puhul võib kasutada järgmist võimalust:

```
<description>
  Siia saab kirjutada pikema teksti.
</description>
```

Mingi elemendiga seotud teksti (juhendi) lisamiseks saab kasutada elementi `label`:

```
<label value="Vajuta siia:" control="open-button"/>
```

```
<button id="open-button" label="Open"/>
```

Pildi lisamine

```
<image src="images/banner.jpg"/>
```

Üldiselt on mugavam pildi URLi määramiseks kasutada CSS'i `list-style-image` võimalusi.

Tekstikasti (*textbox*) lisamine

```
<textbox
  id="tekstikast"
  type="password"
  maxlength="8"/>
```

Atribuut `type="password"` võimaldab peita trükitud sümboleid. `maxlength` määrab maksimaalse sümbolite arvu, mis on antud tekstikastis lubatud. Selleks, et kasti oleks võimalik lisada mitu rida teksti tuleb panna veel `multiline="true"`. Lisaks on võimalik atribuudiga `value` kuvada kastis vaikimisi tekst, atribuut `class` määrab stiili klassi. Samuti saab tekstikasti "ära keelata"(`disabled="true"`). Kindlate mõõtmetega tekstikasti saamiseks on atribuudid `rows` ja `cols`.

Kontrollkastid ja raadionupud

Kontrollkastidel ja raadionuppudel on samad atribuudid, mis tavalistel nuppudel.

```
<checkbox id="case-sensitive" checked="true" label="Case
sensitive"/>
<radio id="punane" label="Punane"/>
<radio id="lilla" selected="true" label="Lilla"/>
<radio id="kollane" label="Kollane"/>
```

Nagu näha kasutatakse raadionuppude puhul `checked` asemel `selected` atribuuti. Raadionuppe saab ka kokku grupeerida kasutades elementi `<radiogroup>`.

Loendid

XUL'is on loetelude loomiseks 2 elementi: `<listbox>` ja `<menulist>`. Esimene on tavalise loendi ja teine ripploendi loomiseks.

Ripploend:

```
<menulist label="Buss">
  <menupopup>
   <menuitem label="Auto"/>
   <menuitem label="Takso"/>
   <menuitem label="Buss" selected="true"/>
   <menuitem label="Rong"/>
  </menupopup>
</menulist>
```

Nagu näha on elemendi `<menulist>` sees omakorda elemendid `<menupopup>` ja `<menuitem>`. Esimene neist on hüpikakna ja erinevate valikute tarbeks. (Loendikasti puhul kasutatakse ühte lisaelementi `<listitem>` valikute loomiseks.) Atribuut `selected` näitab loendi vaikimisi valikut. Redigeeritava loendi saamiseks on atribuut `editable`.

HTML elementide lisamine

Lisaks kõigile XUL elementidele saab XUL dokumendile lisada ka HTML elemente. Selleks tuleb lihtsalt kasutada XHTML nimeruumi ja panna iga sildi algusesse prefiks html. Samuti tuleb muidugi jälgida XUL'i reegleid (sildid peavad olema väikeste tähtedega, atribuutide väärtused jutumärkide vahel). Küll aga on alati kui võimalik soovitatav kasutada siiski XUL silte. Näide:

```
<html:p>
  Kas sa nõustud tingimustega?
  <html:p>
   <html:input id="html" type="checkbox"/>
   <html:div for="html">Jah, nõustun</html:div>
  </html:p>
</html:p>
```

XUL'is on võimalik ka elementide automaatne paigutumine ja suuruse muutmine. Nt. akna suurus kohandub vastavalt elementidele. Samuti on võimalik ise tühikuid paigutada elemendiga `<spacer>`.

Sellega on XUL'i lihtsamad elemendid kirjeldatud, kuid võimalusi on veel palju. Lisainformatsiooni saamiseks on väga hea materjal `http://www.xulplanet.com`.

Peatükk 3

Talisman

Vaatleme platvormi, millega on võimalik luua veebipõhiste rakenduste kasutajaliideseid, kuid see ei ole selle platvormi ainus funktsioon.

Leidub hulgaliselt inimesi, kes vajavad vahendeid keeruliseks analüüsiks ning andmetele ligipääsemiseks, kuid neil puudub kogemus oma süsteemi loomiseks või siis juba olemasolevate keeruliste süsteemide kasutamiseks ilma koolituseta. See käib eriti laboris töötavate bioloogide kohta, kes soovivad analüüsida oma andmeid. Andmete analüüs on sageli põhimõtteliselt keeruline, sisaldades mitmeid kokkusobimatuid allikaid. Teoreetiliselt on see täpselt sellist tüüpi rakendus, milleks *grid*[1] tehnoloogia sobib, aga hetkel puudub piisavalt lihtne võimalus *grid* ressurssideni jõudmiseks. On vaja sellist klienttarkvara, mida oleks lihtne kasutada ja samaaegselt nii võimas, et oleks kasutatav.

3.1 Mis on Talisman?

Talisman on avatud lähtekoodiga kiire rakenduste arendamise platvorm (*rapid application development platform*) veebipõhiste kasutajaliideste kirjeldamiseks.

[1] GRID on termin mida kasutatakse geograafiliselt eri kohtades paiknevate arvutite kaasamisel ühtseks ressursiks nii, et kasutaja ei pea mõtlema, kus tema ülesandeid lahendatakse. Omavahel on võimalik ühendada erinevaid arvuteid ja arvutuskeskusi, mis kõik paralleelselt sama ülesande lahendamisega tegelevad. Lisaks massiivse paralleelsuse saavutamisele lubab selline arhitektuur vähendada ka arvutite kasutamata seismist ja võimaldab lahendada ülesandeid, mida muidu ei saakski lahendada. [10]

Talisman'i põhifookus on suunatud küll bioinformaatika valdkonnale, aga ta sobib siiski peaaegu iga ülesande jaoks, mis nõuab veebipõhist kasutajaliidest.

Talisman loodi algselt EBI's (*European Bioinformatics Institute*) SwissProt grupi poolt, edasine arendus kuni praeguse ajani on toimunud eScience R&D grupi egiidi all ja on rahastatud EPSRC myGrid[2] projekti osana.

3.2 Milleks on vaja Talisman'i?

Talisman loodi selleks, et bioinformaatika ekspertidel oleks võimalus lihtsalt arendada uusi rakendusi keskendudes seejuures üksnes bioloogiale.

Talisman kannab hoolt kõigi rutiinsete liidese kirjutamise aspektide eest, nagu näiteks vormi parameetrite jälgimine, kasutajasessioonid, ressursihaldus jne. ning jätab kasutaja tööks ainult süsteemi käitumise defineerimise. Tegelikult on rakenduse genereerimiseks kasutatav lähtefail informatsiooni sisalduselt üsna sarnane spetsifikatsioonile, mis tavaliselt antakse programeerijale rakenduse tühjalt kohalt ülesehitamiseks, kuigi fail on koostatud kindlat formaalset süntaksi kasutades.

MyGrid projekti osana on Talisman'is tahetud lubada ligipääsu mitmesugustele ressurssidele kaasa arvatud andmebaasid, analüüsprogrammid, *grid*'is salvestatu jne.

Talisman'i rakendusi on suhteliselt lihtne kasutada hoolimata sellest, et need võimaldavad pääseda ligi *grid*'i võimalustele ning teistele sarnastele keerulistele süsteemidele. Tugiinfrastruktuur Talisman'is lubab sellel protsessil olla piisavalt väle, et teha selline lähenemine praktiliseks.

Talisman'i rakendus koosneb ühest või enamast XML definitsiooni failist, mis defineerivad rakenduse käitumise. Definitsiooni süntaks on tahtlikult minimalistlik tuginedes lehe küljendamisel ja informatsiooni esitamisel Talisman'i ekspertsüsteemidele. Enamus toodetud rakendusi sisaldab alla 4 lehekülje XML definitsioone; tavaliselt Javas läheks mitu tuhat rida sama funktsiooni täitmiseks.

Kui kasutaja pöördub nende failide poole läbi veebibrauseri, siis definitsioonifailid kompileeritakse serveri poolseteks andmestruktuurideks, mida esitatakse kui komplekti dünaamiliselt genereeritud veebilehti lubades kasutajal tegeleda nende andmestruktuuridega. Interaktsioonid ja kuvatavad andmed on mõlemad täielikult defineeritud algse XML definitsioonifaili poolt.

[2] http://www.mygrid.org.uk/

Talisman pakub erinevaid algandmetüüpe ja toiminguid, näiteks tekstiväljad, pildid, regulaaravalisel baseeruvad tekstioperatsioonid ja muud sarnased üldised funktsioonid. Süsteemi tõeline võimsus tuleneb selle laiendatavusest, kuid selleks, et pääseda Talisman'ist ligi teistele rakendustele, on vaja ühendavat koodi (*bridging code*). Samas, kui see kood on korra juba kirjutatud, siis süsteem saab kopereeruda kõige muuga Talisman'i sees.

Talisman'i käitusfaasikeskkond teostab erinevaid funktsioone. Kõige elementaarsem on ette antud rakenduse jooksutamine, kuid on veel funktsioone, mida kõik aplikatsioonid võivad ära kasutada. Hetkel on kõige levinum päritolu jälgimine.

3.3 Näiterakendus Talisman'is

Talisman on süsteem kiireks veebipõhiste rakenduste ehitamiseks. Kasutades ülemaailmset võrgendit (*world wide web*) on võimalik saavutada kohe oma lehe suurt levikut ja platvormist sõltumatust. Leviku all mõeldakse võimalust pääseda loodud süsteemile ligi erinevatest geograafilistest kohtadest. Platvormist sõltumatus kasutab ära fakti, et enamikel süsteemidel on piisavate võimalustega veebibrauser Talismani põhiliste omaduste jooksutamiseks.

Kuidas luua lihtsat vormi kasutades Talismani? Kuna Talisman baseerub laiendataval pluginate süsteemil, siis rakenduse funktsionaalsus sõltub kättesaadavatest pluginatest. Kusjuures pluginate arv kasvab ilmselt oluliselt, kui Talisman on levinud erinevatesse valdkondadesse. Praegused pluginad hõlmavad endas XML valideerimise ja relatsioonilise andmebaasi ligipääsu, juurdepääsu emailile.

Üldiselt on iga Talismani poolt genereeritud veebileht toodetud ühest Talisman'i lehekülje definitsiooni failist (*Page Definition file*). See peab olema paigaldatud Talismani serverisse, milleks peab muidugi arvutis olema töötav Talismani instants.

Lehe definitsiooni fail ühe tekstiväljaga

Loodav fail tuleb paigutada Talismani installi juures olevasse kataloogi '`realpages`' ning laiendiks peab olema `.xml`. Nimi võiks seega olla näiteks vorm.xml. Failile ligi pääsemiseks tuleb veebibrauseris osutada sellele: `[talisman'i\_asukoht]/pages/vorm`

Järgnevas XML failis kirjeldatakse üherealine tekstiväli (tekstikast):

```
<talisman>
  <page name="Lihtne vorm">
   <field name="TextInput" type="Text" value="A
text input box">
    <description>This is a text field</description>
    <short>Example text field</short>
   </field>
  </page>
</talisman>
```

Esitatud XML failis loodi Talisman'i leht ühe teksti sisestamise väljaga.

Faili alguses määratletakse ära, et tegemist on Talisman'i failiga ning deklareeritakse lehe algus ja antakse sellele nimi (Lihtne vorm).

Enamik Talisman'i lehti on kokku pandud väga erinevat tüüpi väljadest. Igal väljal on defineeritud `<field>` silt ja `name`, `type` ja `value` atribuudid. Need ütlevad Talisman'ile vastavalt kuidas identifitseerida välja, mis liiki andmeid see sisaldab (antud juhul tekst) ja mis on hetkel välja väärtus. Lisaks on väljadel `<description>` ja `<short>` sildid (*tags*), mis annavad vastavalt välja täieliku kirjelduse ja nime, mida kasutatakse selle kuvamiseks. Hea tava on alati väljadele lisada ka kirjeldus (`description`), et igaüks kes loeb antud lehe faili saaks aru, mida see teeb.

Tekstiala (*textarea*)

Järgmiseks lisatakse mitmerealine tekstiväli. Uue välja lisamiseks tuleb lihtsalt, selle definitsioon lisada `vorm.xml`'is sinna, kus eelnev väli lõppes.

```
<field name="TextAreaInput" type="TextBox"
columns="80" rows="10"  value="Suurem tekstiväli">
 <description>See on tekstiala</description>
 <short>Tekstiala näide</short>
</field>
```

Lisatud on tekstiala (*textarea*) väli. Nagu näha, saab definitsiooni parema loetavuse saavutamiseks poolitada, kuna lehekülje fail koosneb XML'ist ja XML ignoreerib tühikuid ja reavahetusi. Brauserist vaadates on lehel nüüd kaks tekstivälja. Loodud lehed ei tee veel tegelikult midagi, kuid Talisman'iga saab lisada lehele ka interaktiivset käitumist

Päästikprotsessid (*triggers*)

Staatilised lehed on toredad, kuid neid saab lihtsalt luua ka tekstieditori abiga, selleks ei ole vaja nii keerulist süsteemi nagu Talisman. Õnneks õigustab Talisman oma eksistentsi võimega lisada interaktiivset käitumist genereeritavatele lehtedele.

Talisman'is mitte lihtsalt ei defineerita nupp (nagu XUL'is) vaid lisatakse sellele kohe tegevus ka külge.

Selles osas lisame oma lehele nupu, mis muudab sellel klikkides tekstiala välja suurust.

Ühesõnaga, lisaks väljadele on Talisman'is veel nn. päästikprotsessid (*triggers*). Need on tavaliselt nupud veebilehel, mida kasutaja saab klikkida, et Talisman teeks midagi.

Talisman'i päästikprotsessid sisaldavad tegevuste nimekirja ja iga tegevus on juhend Talisman'ile mingi lihtsa operatsiooni sooritamiseks, näiteks andmete sissetoomine andmebaasist, välja kustutamine või parameetrite muutmine mõnel teisel lehe komponendil. Kui päästikprotsessil on rohkem kui üks tegevus, siis Talisman teeb iga tegevuse selles järjekorras nagu nad on kirja pandud. Hetkel ei ole võimalik Talisman'ile öelda, et ta teeks kahte tegevust paralleelselt, teist tegevust saab alustada alles siis kui esimene on lõpetatud. Seega teine tegevus saab toetuda esimese tulemustele.

Päästikprotsessi lisamine:

```
<trigger name="ChangeSize" visible="true">
 <description>Pane tekstikasti pikkuseks 20 rida
</description>
 <action name="ChangeSizeAction" class="SetParameter">
  <actionparameter name="target">TextAreaInput</actionparameter>
  <actionparameter name="parameter">rows</actionparameter>
  <actionparameter name="value">20</actionparameter>
 </action>
</trigger>
```

Silt `<trigger>` annab Talisman'ile teada, et defineeritakse nupp, mida saab näidata genereeritud veebilehel. Parameeter `name` töötab samamoodi nagu väljade puhulgi - annab päästikprotsessile nime, millega saab viidata sellele teistest komponentidest (nagu antud päästikprotsess viitab tekstiala väljale). Atribuut '`visible`' saab olla kas `true`, kuvades päästikprotsessi nupuna või `false` vastasel korral. Silt `<description>` on üldine atribuut läbi

Talisman'i; see võib olla peaaegu igal komponendil ning töötab samamoodi nagu välja juures kirjeldati.

Sildi `<trigger>` sees on üksik `<action>` silt. Neid võib olla päästikprotsessi kohta ka rohkem kui üks. Atribuut `name` on sama nii päästikprotsessi de kui ka väljade puhul. Kriitiline parameeter on siin `class` atribuut, kuna see määrab täpselt ära milline kood laetakse ja käivitatakse tegevuse (*action*) poolt. Talisman'i kodulehel on link 'JavaDoc', mille all on java klass `org.embl.ebi.escience.talisman.action.SetParameter`. Klikkides sellel on näha, et sellele eraldiseisvale klassile peab olema saadetud 3 parameetrit. JavaDoc'is on seletus, mida see klass teeb ja mida parameetrid tähendavad.

Tegevus seatakse 0 või enama (antud näites 3) `<actionparameter>` elemendi abil, mis sisaldavad `<name>` atribuuti (vastab parameetri nimele tegevuse klassi JavaDoc'is) ja ümbritseb tekst, milles on parameetri väärtus sellele nimele. Ülevalpool esitatud näites on parameeter nimega `target` ja tema väärtus on `TextAreaInput`. Parameetris `target` on välja objekti nimi, kuna see liidab tegevuse (parameetri sättimine või midagi muud) ja reaalse efekti (teksikasti suurendamine). `uk.ac.ebi.talisman.action.SetParameter` klass lihtsalt otsib komponenti (välja, päästikprotsessi, jne) nõutud nimega (`target`) ja paneb antud atribuudi (`parameter`) antud väärtusele (`value`). Esitatud näite puhul *action* leiab välja nimega `TextAreaInput` ja paneb sellele `rows` atribuudi väärtuseks 20. Nüüd kui leht uuesti joonistada (nt. nuppu vajutades), siis tekstiala on suurem.

Valikuloendid (*selectionlists*) ja kontrollitud sõnastikud (*controlled vocabularies*)

Talisman'is on võimalik kasutada ka mitmeid keerulisemaid vormielemente, mis ei vasta otse HTML vormile. Kõige üldisem mitte eriti lihtne andmetüüp on kontrollitud sõnastik (*controlled vocabulary*), millest lihtsaim variant on ilmselt `true` ja `false`. Selleks, et sellist tüüpi välju kasutada on vaja esiteks defineerida `<selectionlist>`, mis sisaldab soovitud sõnastikku. See lubab defineerida terminite nimekirju, mida kuvatakse ja lisaks igale terminile erinevat väärtust. Viimane on kasulik juhul, kui andmed on andmebaasis salvestatud kui `T`,`F`,`P` jne. kuid vaja on kuvada `True`,`False`,`Partial` nii, et süsteemi kasutajad ei pea teadma, mida need ühetähekoodid tähendavad. Teiseks on vaja luua väli tüübile, mis pääseb ligi loodud valikuloendi elemendile ja pakub sealt valikut kasutajale.

Järgnev kood lisab `true/false` valiku kasti lehele.

```
<field name="SelectionInput" type="Select"
selection="TFSelectionList" value="F">
 <description>See on valikuloendi väli.</description>
 <short>Valikuloendi näide</short>
</field>
<selectionlist name="TFSelectionList">
 <listelement value="T">True</listelement>
 <listelement value="F">False</listelement>
</selectionlist>
```

Element `<selectionlist>` defineerib leksika ja `type` atribuut osutab sobiva tüübi (antud juhul `select`) väljale selles. Atribuut `value <listelement>` sildil on valikuline, kui seda ei ole, siis võetakse väärtuseks loendi elemendi sisu, seega `<listelement value="T">T</listelement>` on täpselt võrdne reaga `<listelement>T</listelement>`.

Valikuloend võib olla jagatud mitme erineva välja vahel. Sellisel juhul tuleb lihtsalt dubleerida välja deklaratsiooni kuigi arvatavasti on iga välja jaoks erinev nimi ja nad kõik kasutavad sama leksikat.

Graafilised vormi osad, liugurid (*sliders*)

Talisman'is on hulk funktsionaalsust, mis on mõeldud peamiselt kohalikus võrgus kasutamiseks. Need komponendid kasutavad Java-aplette ning nõuavad seega Java Runtime Environment'i (JRE) installimist Sun Microsystems'ist[3].

Kasutades aplettide eeliseid saab Talisman lehtedele lisada keerulisi graafilisi komponente.

Tasub teada, et need keerulised komponendid ei pruugi alati töötada (näiteks vanemate veebibrauseritega, aga ka uutega, kui puudub JRE Java plugin).

Järgnev näide lisab lehe alumisse äärde konfigureeritava graafilise liuguri:

```
<field name="SliderInput" type="Slider" min="50" max="150"
major="20" minor="4" labels="true" value="70">
 <description>See on apletti kasutav liuguri väli.</description>
 <short>Liugur</short>
</field>
```

[3] http://java.sun.com

Atribuudid `min`, `max` ja `value` määravad vastavalt liuguri minimaalse, maksimaalse ja hetkeväärtuse.

Atribuudid `major` ja `minor` määrab väikeste ja suurte kriipsude vahe liuguril ning `labels` näitab, kas kuvatakse ka numbriline skaala.

Need olid Talisman'is loodava kasutajaliidese põhilised osad. Rohkem infot leiab Talisman'i projekti koduleheküljelt:

`http://www.ebi.ac.uk/collab/mygrid/service1/talisman/index.html`

Peatükk 4

EP ja tema kasutajaliidese arhitektuur

4.1 Mis on Expression Profiler?

Expression Profiler (EP)[1] loodi algselt 2001 aastal Jaak Vilo poolt. Hetkel tegelevad selle arendamisega peamiselt Patrick Kemmeren ja Misha Kapushesky. Viimaselt pärineb ka EP kõige uuema versiooni - EP:NG[2] - idee.

Järgnevalt esitan Misha Kapushesky definitsiooni EP kohta: *Open, extensible web-based collaborative platform for microarray gene expression, sequence and PPI data analysis, exposing distinct chainable components for clustering, pattern discovery, statistics (thru R), machine-learning algorithms and visualization. Expression Profiler: Next Generation (EP:NG) is a web-based environment for the analysis of, mainly, two types of data: gene expression and sequences. The system is designed to be extensible to other data types - currently protein-protein interaction (PPI) data support is being added.*[1]

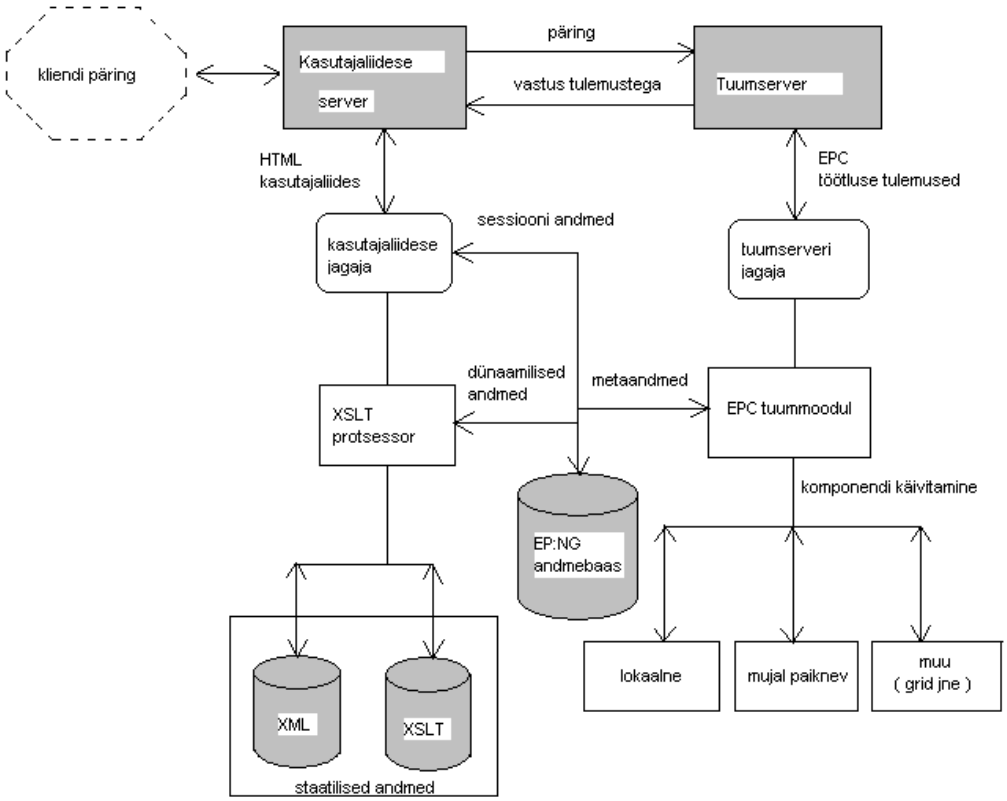
4.2 EP:NG kasutajaliidese arhitektuur

EP:NG on mõeldud kasutamiseks Apache veebiserveris.

Tuumkood on kirjutatud C/C++'is või R'is (keel ka keskkond statistilisteks arvutusteks ja graafikute loomiseks) ning EP:NG erinevad osad on ühendatud perli siduva koodiga.

[1] http://ep.ebi.ac.uk/EP/

[2] http://ep-sf.sourceforge.net



Joonis 1.1: Süsteemi arhitektuur

Kasutajaliides luuakse XML/XSLT transformatsiooni teel.

Põhiolemuselt on EP:NG kolme kihiline raamistik. Kasutajaliidese kirjeldamise aspektist on kõige olulisem esitluskiht (*presentation layer*), kuid ülevaatlikkuse huvides on oluline ka äriloogika kihi (*business layer*) ja andmekihi (*data layer*) olemuse selgitamine.

EP:NG koosneb üksikutest funktsionaalsetest komponentidest (*EPC - Expression Profiler Component*), millest igaüks on kirjeldatud selle sisendite, väljundite ja käivituste abil ühe XML faili poolt.

4.2.1 Lühiülevaade

Klient taotleb komponenti sisenedes EP:NG'sse. Seejärel kasutajaliidese jagaja (*UI dispatcher*) visualiseerib sellele komponendile kasutajaliidese. Klient annab visualiseeritud kasutajaliidese kaudu sisse parameetrid, millest kasutajaliidese jagaja ehitab komponendi XML päringu (*EPC query XML*), valideerib sisse antud parameetrid ning saadab taotluse tuumserverile (*CORE server*) taas komponendi XML päringuna. Tuumserver täidab komponendi ja annab tulemuseks XML vastuse. Kasutajaliidese jagaja visualiseerib järgmise komponendi graafilise kasutajaliidese andes vastu võetud XML vastuse parameetrina edasi XSLT protsessorile.

4.2.2 Graafilise kasutajaliidese (*GUI*) server

Kasutajaliidese jagajat jooksutab graafilise kasutajaliidese server, millel on järgnevad funktsioonid:

- graafilise kasutajaliidese visualiseerimine komponendi kirjeldusfailidest;

  Graafiline kasutajaliides on kõik tavaline HTML, mis visualiseeritakse komponendi XML kirjeldusfailidest XSLT laaditabelite (*stylesheets*) abil - saades kliendilt päringu mingile komponendi kasutajaliidesele otsitakse välja selle komponendiga seotud XML fail ning esitatakse tulemus HTML'is.

- sessiooni seisundi hoidmine EP:NG kasutaja interaktsioonile;

  Nimelt on kasutajaliidese serveris olemas teenus, mis hoiab alles sessiooni seisundite informatsiooni erinevate programmimoodulite vahel. Ühesõnaga, kui klient ühendub EP:NG veebilehega, siis lastakse tema tegevuste jälgimiseks läbi sessiooni välja unikaalne sessiooni identifikaator (*SID*).

- komponentide töötluse lähetamine tuumserverile.

  Kasutajaliidese jagaja võtab kliendi antud (kas läbi visualiseeritud HTML vormide või veebiteenuste (*Web Service*) liidese kaudu) sisendi ja loob XML sõnumi kodeerides komponendi sisendparameetrid. Sõnum saadetakse tuumserverisse, kus seda töödeldakse erinevate moodulite poolt ning tulemuseks saadakse XML sõnum. Kasutajaliidese jagaja programm algatab tulemuste kätte saamisel kas järgmise komponendi

töötluse või annab tulemused vaikimisi seatud järgmisele komponendile ning visualiseerib kasutajaliidese.

- Veebiteenuste liidese esitamine EP:NG-st väljapoole.

  Nimelt on komponente võimalik käivitada ka programselt väljastades XML sõnumeid graafilise kasutajaliidese serverile, mis peaks need saatma otse tuumserverile ilma kasutajaliidest väljastamata.

4.2.3 Tuumserver (*CORE server*)

Tuumserver teenindab äriloogika kihti ning vastutab komponendi parameetrite parsimise ja nende lähetamise eest sobivatele moodulitele, mis töötlevad komponenti. Põhi sisenemis punkt EP:NG tuumserverisse `cgi-bin/ep_core.pl` seab üles tuumserveri jagaja (*CORE dispatcher*) ja kutsub välja selle `EP::EPC::Core::Dispatcher::dispatch_request` meetodi, mis annab komponendi protsessi töötlemise üle individuaalsele EPC tuummoodulile `try/catch/otherwise` plokis.

4.2.4 Kolmanda osapoole tööriistad (*third-party tools*)

Kuigi osade komponentide täidetavad osad on ise perli moodulid (nt. Data Selection komponent) võivad mõned olla eraldiseisvad teiste programmide osad või tööriistad. Üldine abinõu nende liidestamiseks on integratsiooni mallide (*templates*) kaudu. Need on individuaalselt kohaldatud `Text::Template` failid (asuvad `cgi-lib/tmpl` kataloogis), mis on parsitud komponendi perli tuuma poolt, mis täidab nõutud parameetrite kohad mingis kolmanda osapoole formaadis ja annab lõpetatud malli välisele tööriistale täitmiseks.

4.2.5 Komponentide aheldamine (*Component chaining*)

Komponendi töötlus algatatakse, kui komponendi XML päring näitab, et seda tuleb teha. Komponendi XML päring võib ka näidata, et mitu komponenti on järjest töötluses. Seda nimetatakse komponentide aheldamiseks (*chaining*). Komponentide käivitamise järjekorra saab paika panna kasutades `target_sequence` elementi.

Iga jada komponendi puhul kutsutakse välja `ep:process_component`, tulemused püütakse kinni `node_set`'i ja antakse argumendina edasi järgmisele jada komponendile.

Seega ühe komponendi väljundeid saab kasutada kui teise sisendeid. Komponendi spetsifikatsioonid (komponendi XML failid) on vastutavad sobivate sisendite ja väljundite andmise eest.

XSLT saab visualiseerida mitte ainult komponentide jadasid, vaid ilmselt igat tüüpi komponendi päringuid defineerivaid XML struktuure. Seega saab terveid komponentide jadade puid visualiseerida, mis tähendab, et kasutaja poolt vaadatuna saab terveid tegevuste puid uuesti täita, lihtsalt andes uuesti sisse vajalikud komponendi XML päringud.

4.2.6 Staatiline XML

Staatiline XML (kasutajaliides, mis reeglina ei muutu) on kirjutatud komponendi XML kirjeldusfailidesse `static/xml'is`. Kogu kasutajaliides, mis ei nõua esitamiseks mingite andmete genereerimist või üle vaatamist peaks olema kirjeldatud nendes XML failides. See tähendab et, kui komponendil on vaja esitada 1000 elemendi valik 100-s rippvalikukastis (*drop-down box*), siis need kõik peaks olema kirjutatud XML faili.

Varasem kujundusele lähenemine toetas andmebaasi põhist staatilist kujundust. See tõi aga kaasa mitmeid raskusi Expression Profiler'i komponendi arendajale - kohmakas API ja raske komponentidede haldus - ühesõnaga nüüd ollakse veendumusel, et parim on hoida kõik staatiline kasutajaliides XML failides.

4.2.7 Dünaamilised andmed

Liidese dünaamilised (nt. sõltuvad andmebaasist või genereeritud andmetest) elemendid edastatakse kasutajaliidese jagajale staatilistes XML dokumentides olevate `dynamic_` (või `XXX`) elementide abil.

Vaatame erinevaid dünaamilisi andmetüüpe:

- **dünaamilised sisendid** (*dynamic inputs*);

  Peaaegu igas komponendis on olemas read:

  ```
  <dynamic_input type="hidden" name="session_id"
  call="get_session_id">

  </dynamic_input>
  ```

  See annab peidetud sisendivälja ja paneb hetke sessiooni identifikaatori sinna.

- **dünaamilised valikud** (*dynamic selects*);

  Need on dünaamiliselt genereeritud rippvalikukastid.

- **dünaamilised loendid** (dünaamiliste sisendite loendid) (*dynamic lists*);

  Need on kõige üldisemalt kasutatavad. Nad annavad dünaamiliselt dünaamiliste loendite sisendid. Põhimõte on selles, et väline väljakutse genereerib ühe dünaamilise loendi ja iga selle ühiku jaoks tehakse väljakutse, mis ise saab vastu anda palju ühikuid.

4.2.8 Expression Profiler'i XML'i visualiseerimine XSLT'ga

Nagu juba öeldud, komponendi XML failide visualiseerimisel kasutajasõbralikuks liideseks kasutatakse XSLT laaditabeleid.

Elementaarse kasutajaliidese visualiseerimiseks on lähtelaaditabel `basic_ui.xsl`. Tavaliselt saab laaditabel EPC XML lähtekoodi vähemalt kahelt komponendilt korraga - hetkel taotletud komponendilt ja selle vaikimisi järgmiseks seatud komponendilt. Lisaks taotleb laaditabel XML päringut kasutajaliidese jagajalt `ep:get_epc_query` XSLT laiendfunktsiooni kaudu. Hetkel aktiivse komponendi (nt. see millele kasutaja taotles töötlust) komponendi XML päringus on märgitud `active` atribuudi väärtuseks 1.

XSLT protsessor seega visualiseerib kõik taotletud komponendid, kutsudes välja laiendfunktsioone terve selle protsessi käigus. Kui komponendi XML päring näitab, et seal on aktiivne komponent, siis kutsutakse välja `ep:process_component` funktsioon ja väljund, mis on välja otsitud `ep:component_output` funktsiooni kaudu antakse edasi XSLT mallidele aktiivse komponendi vaikimisi seatud järgmise komponendi visualiseerimiseks. Kui aktiivset komponenti, siis kuvatakse komponendi XML lähtekoodi esimene komponent.

Plaanis on tekitada selline disain, et erinevaid laaditabelite komplekte saaks kasutada komponentide visualiseerimiseks erinevatele kasutajagruppidele või klientidele. Veelgi enam, laaditabelite sees peaks eksisteerima parameetreid visualiseerimise häälestamiseks.

Praegune EP:NG väljalase hõlmab ühte laaditabelite põhikomplekti, mis on suunatud algaja kasutajatasemele. Sellel laaditabelite komplektil on parameeter `user_level`, mis saab olla vastavalt kas 0,1 või 2 - algajate, keskmiste

ja edasijõudnute tase. Vaikeväärtus on 0. Laaditabelid suhtlevad kasutajaliidese jagajaga XSLT laiendfunktsioonide kaudu. Need funktsioonid on registreeritud ep namespace'is ja kutsutakse välja kui xsl funktsioonid:

```
   <xsl:variable name="this_component_output"
select="ep:component_output(string($this_component_id))"/>
```

4.3 Expression Profiler'i komponendi kasutajaliidese loomine

Iga komponendi kirjeldamiseks luuakse eraldi XML fail. Need transformeeritakse XSLT laaditabelite abil HTML'iks. XSLT laaditabelid on aluseks XML failide loomisel.

4.3.1 XML faili loomine

Vaatleme XML faili loomist Data Upload komponendi näitel. Kogu Data_Upload.epc.xml fail on toodud Lisas 1.

XML'i deklaratsioon

XML fail peaks algama XML'i deklaratsiooniga:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Esimesed 5 märki näitavad parserile, mis failiga üldse tegu on ning `encoding` määrab ära, millist tekstikodeeringut dokumendis kasutatakse.

Juurelement

Igal XML dokumendil on juurelement. EP komponentide puhul on selleks elemendiks `component`.

```
<component xmlns:xinclude="http://www.w3.org/2001/XInclude"
id="1" title="Data Upload" user_level="0" display="1">
```

Nimeruumi viide `xmlns:xinclude="http://www.w3.org/2001/XInclude"` võimaldab dokumendis `<xinclude />` siltide ja mehhanismi kasutamist. `id` on komponendi identifikaator, `title` on komponendi nimi, `user_level` näitab kasutajataset (nendest oli varem juttu) ning `display` komponendi kuvatavust.

Üldine komponendi info

Hea tava on ka faili algusesse panna informatsioon selle kohta, mis dokumendiga tegu, selle autorid jne.

Näide:

```
   <about>
     <Version>1.0</Version>
     <Author>Patrick Kemmeren (email: patrick (at) ebi.ac.uk)</Author>
     <Author>Misha Kapushesky (email: ostolop (at) ebi.ac.uk)</Author>
     <Description>
           <p align="justify">On this screen you can upload your
own data - either via a file, or from a URL, or by pasting it
directly into provided text boxes.</p>
     </Description>
   </about>
```

4.3.2 Põhiline kasutajaliides - `basic_ui.xsl`

Failis `basic_ui.xsl` on mall, milles esineb atribuut `match="/"`. See tähendab, et iga komponendi kasutajaliidese kuvamist alustatakse sellest mallist. Ühesõnaga iga Expression Profiler'i komponendi puhul joonistatakse kõigepealt menüü (luuakse sama faili mallis `render_ebc_menu`) ning menüüriba paremale serva pannakse EP kasutaja nimi ja aeg minutites sessiooni aegumiseni. Samuti luuakse `basic_ui.xsl`'is HTML'i tabel komponendi enda kasutajaliidese kuvamiseks - kui töötlust nõudvaid komponente on mitu, siis kutsutakse välja kõigepealt mall `process_component_chain`, vastasel juhul kutsutakse välja `render_component` ehk teostatakse komponendi visualiseerimine.

4.3.3 Komponendi tiitliriba

Igal komponendil on tiitliriba. See luuakse mallis `component_title_bar`. Konkreetne komponendi pealkiri saadakse XML failist `component` elemendist, atribuudilt `title`. Pealkiri kuvatakse ekraanil.

4.3.4 Sektsioonid

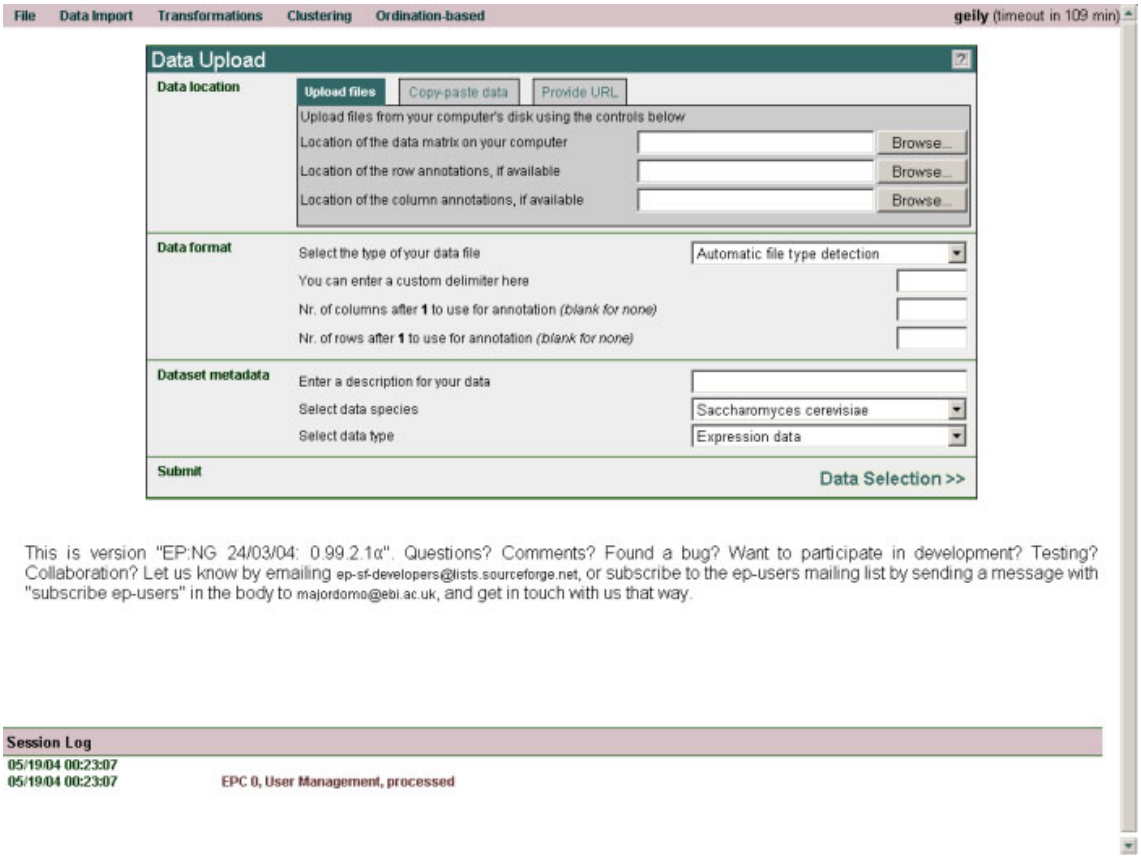
Komponendi kasutajaliides on jagatud sektsioonideks. Data Upload komponendi puhul on need Data location, Data format, Dataset metadata ja Submit. Sektsiooni kirjeldab ära element `section`, millel võib esineda järgmisi atribuute:

`title` - sektsiooni nimi, mis kuvatakse ka ekraani

`id` - sektsiooni identifikaator;

`default_active_subsec` - määrab id järgi alamsektsiooni, mis vaikimisi aktiivne on;

Sektsiooni visualiseerimisega tegeleb `section` mall, mis kutsub välja ka alamsektsioonide sakkide (*tabs*) joonistamise.



Joonis 1.2: Data Upload komponendi kasutajaliides

4.3.5 Alamsektsioonid

Sektsioonil võivad omakorda olla alamsektsioonid (`subsection`), millel vaadeldavas näites on atribuutideks `title` ja `id`. Alamsektsiooni nimi kuvatakse ka sakil. Erinevad alamsektsioonid avanevad kui klikkida vastavatel

sakkidel. Data Upload komponendil on esimese sektsiooni alamsektsioonideks Upload files, Copy-paste data ja Provide URL. Esimene neist on vaikimisi aktiivne (määratud elemendis `section` atribuudiga `default_active_subsec`).

4.3.6 Tekstiväljad

`input` tekstiväli

`input` on tavaline üherealine tekstiväli. Atribuutideks võivad olla `type`, `name`, `size`, `value`. `name` on selle `input`'i nimi, `size` tekstivälja pikkus, `value` sellele `input`'ile seatud väärtus. `type` on tüüp, mille väärtused võivad olla näiteks:

`hidden` - nähtamatu tekstiväli;
`text` - tavaline tekstiväli;
`file` - tekstiväli failide spetsifitseerimiseks (browse nupuga);

Data Upload näide:

```
<input type="file" name="src_file" size="30">
 <shorthelp>Location of the data matrix on your computer</shorthelp>
 <context_help id="3">
  <xinclude:include href="Data_Upload.doc.xml#xpointer
  (/epc_doc/ctxt_topic[@id=3])" />
 </context_help>
 <longhelp />
 </input>
```

Antud lõigus kirjeldatakse üherealine tekstiväli, mille pikkuseks on 30. Kuna type="file", siis tekstivälja juurde tekitatakse ka 'Browse' nupp. Eelpool esitatud Data Upload komponendi ekraanivaates on näha see väli kohas, kus on kiri 'Location of the data matrix on your computer'.

`textarea` tekstiväli

`textarea` on mitmerealine tekstiväli, mille atribuutideks võivad olla `name` (selle tekstivälja nimi) ning `rows` ja `cols`, mis määravad tekstiala suuruse.

Data Upload näide:

```
<textarea name="src" rows="5" cols="50">
 <shorthelp>Paste the data matrix here</shorthelp>
```

```
  <longhelp />
</textarea>
```

Luuakse 5 x 50 mõõtmetega tekstiala, mis ekraanivaates on näha esimese sektsiooni teises alamsektsioonis.

`select` tekstiväli

`select` on valikutega tekstiväli, kus valikud seatakse <option /> sildiga. Data Upload komponendi kasutajaliidese teises sektsioonis kasutatakse seda tüüpi tekstivälja andmefaili tüübi määramiseks:

```
<select name="filetype">
  <shorthelp>Select the type of your data file</shorthelp>
  <option value="AUTO">Automatic file type detection</option>
  <option value="XLS">Microsoft Excel spreadsheet</option>
  <option value="SPACE">Single space delimited data</option>
  <option value="TAB">Tab-delimited data</option>
  <option value="VWS">Any-length whitespace delimited data</option>
  <option value="CUSTOM">Custom delimiter (specify below)</option>
</select>
```

Elemendist `shorthelp` räägitakse lähemalt järgmises peatükis.

4.3.7 Juhendid

Tekstikastide juurde on vaja ka juhendit selle kohta, mida sinna sisestada. Selleks otstarbeks on elementide juurde võimalik lisada abistav tekst sildiga `<shorthelp />`. Nt. Data Upload komponendi kasutajaliidese esimeses sektsioonis, esimese tekstivälja juurde tekitati tekst järgmiselt:

```
  <shorthelp> Upload files from your computer's
disk using controls below
  </shorthelp>
```

4.3.8 Spikker

Spikker kuvatakse, kui hiirega kasutajaliidese osadest üle minna. Selle informatsioon võetakse vastava komponendi dokumendi failist ehk XML failis

võiks see välja näha järgmiselt:

```
<context_help id="1">
<xinclude:include href="Data_Upload.doc.xml#xpointer
(/epc_doc/ctxt_topic[@id=1])">
</context_help>
```

`context_help` on element ning tema atribuudiks `id`, mis määrab ka dokumendifailis koha, kust õige spikker võtta. Spikri visualiseerimine toimub mallis `context_help`, kus oli määratud, et võetakse lihtsalt elemendi sisu ja kopeeritakse väljundisse (`<xsl:copy-of select="." />`).

`xinclude` võtab antud dokumendist `xpointer`'iga viidatud lõigu ning paneb teise, kust vastav spikker ka kuvatakse.

4.3.9 `<action>` element

Iga komponendi XML definitsioonifailil on `action` element. See kirjeldab millisel kujul lehel modifitseeritud andmeid edasi töödeldakse. `success_test` kirjeldab lehekülje eduka töötlemise tingimused. `output` siltides kirjeldatakse järgmiseks edastatavate andmete XML kuju ning võimalikud veasituatsioonid juhul kui vastav sisend andmetest puudub.

Kokkuvõte

Kuni viimase ajani olid veebirakendustes segamini nii programmi kui ka kasutajaliidese loogika. Sellisel kujul on aga üsna keeruline rakendust hallata. Praeguseks pakuvad aga W3C standardid piisavalt võimalusi rakenduse kasutajaliidese ehitamiseks. XML võimaldab kirjeldada eraldi kasutajaliidese komponendid ja sisu, mille saab XSLT ja lisaprogrammide abil teisendada kasutajaliideseks.

Antud bakalaureusetöö kirjeldabki erinevaid võimalusi XML põhise kasutajaliidese loomiseks.

Kõige tuntum neist on Mozilla XUL (*XML User-Interface Language*). XUL ei ole mõeldud veebipõhiste rakenduste kasutajaliideste loomiseks, sellepärast on ta ka küllaltki lihtsa arhitektuuriga. Samas, kasutades koos XUL'i valmis tööriistu, W3C (*World Wide Web Consortium*) standardeid (HTML 4.0, DOM 1&2, CSS 1&2, XML Namespaces) ja JavaScripti on võimalik kujundada väga erinevaid kasutajaliideseid. Ka Mozilla veebibrauseri kasutajaliides on tehtud XUL'is.

Algselt EBI's (European Bioinformatic Institute) loodud Talisman sobib veebipõhiste kasutajaliideste kirjeldamiseks. Kuid lisaks on Talisman'is MyGrid projekti osana tahetud lubada ligipääsu mitmesugustele ressurssidele kaasa arvatud andmebaasid, analüüsprogrammid, grid'is salvestatu jne.

Expression Profiler (EP) on EBI loodud veebipõhine platvorm geneetika alase informatsiooni töötlemiseks.

EP:NG'le on tehtud XML põhine kasutajaliides, ehk iga EP komponent on kirjeldatud ühes XML failis, mis vajaduse korral transformeeritakse XSLT laaditabelite abil kasutajaliideseks.

Põhiprobleemiks praegu see, et puudub standardne keel, mida saaks kasutada nii töölaua- (*desktop*) kui veebipõhiste rakenduste kasutajaliidese loomiseks. Samuti oleks vaja nn. kiire arenduse keskkonda, kus saaks graafiliselt kasutajaliidese kujundada.

The XML-based user interface architecture

Geily Niinemets

Abstract

Until quite recently most of web applications used to have mixed program logic and user interface logic. Application maintenance for such architectures is often difficult. Modern W3C *(World Wide Web Consortium)* standards offer sufficient capabilities to create application user interface. XML provides means to describe user interface components and contents separately, which can be combined into working user interface using XSLT and some utility code.

This bachelor's work describes different currently existing possibilities for XML-based user interface design.

The most commonly known method is Mozilla XUL (*XML User-Interface language)* XUL is not designed for web application user interfaces, thus its architecture is relatively simple. When in conjunction to XUL other toolkits, W3C standards (HTML4.0, DOM level 1 and 2, CSS 1 & 2, XML-NS) and scripting languages like Javascript, are used, its possible to create very different user interfaces. The user interface of popular Mozilla web browser is built with XUL.

Primarily developed by EBI (*European Bioinformatic Institute*), Talisman is suitable for describing web-based applications user interfaces. Additionally, Talisman as a part of MyGrid project provides built-in means for accessing different resources like databases, analysis packages, grid storage etc.

Expression Profiler (EP) is open, extensible web-based collaborative platform for microarray gene expression, sequence and PPI data analysis, expo-

sing distinct chainable components for clustering, pattern discovery, statistics (thru R), machine-learning algorithms and visualization.

EP:NG does have an XML-based user interface, where every component is described in a XML-document, which is transformed into user interface by corresponding XSLT templates.

Currently the basic problem with XML-based UI development is lack of a common standardized language that would work in both web applications and desktop. For the solutions that exist, they lack RAD (rapid application development) tools that exist for other mature UI development toolkits like Qt, GTK or Microsoft MFC.

Sõnastik

andmekiht - data layer
dünaamiline loend - dynamic list
dünaamiline sisend - dynamic input
dünaamiline valik - dynamic select
EP:NG - Expression Profiler: Next Generation
EPC - Expression Profiler Component
esitluskiht - presentation layer
faili eraldajate tüübid - file delimiter types
graafilise kasutajaliidese server - GUI server
kasutajaliidese jagaja - UI dispatcher
kolmanda osapoole tööriistad - third-party tools
komponendi XML päring - EPC query XML
laaditabel - stylesheet
lehekülje definitsioonifail - Page Definition File
lähtepuu - source tree
mall - template
nimeruum - namespace
pesastatud - nested
päästikprotsess - trigger
rippvalikukast - drop-down box
sakk - tab
silt - tag
tekstiala väli - textarea field
tulemipuu - result tree
tuumserver - CORE server
tuumserveri jagaja - CORE dispatcher
töölaua rakendus - desktop application
valikuloend - selectionlist

vidin - widget
äriloogika kiht - business layer
ühendav kood - bridging code
ülemaailmne võrgend - world wide web
XUL - XML User-interface Language

Kirjandus

[1] EP:NG projekti koduleheküIg.

http://ep-sf.sourceforge.net/, viimati külastatud 19.05.2004.

[2] XSLT Tutorial.

http://www.w3schools.com/xsl/default.asp, viimati külastatud 19.05.2004.

[3] Elliote Rusty Harold, W. Scott Means. XML in a Nutshell, 2nd Edition. O'Reilly, 2002.

[4] Thomas M. Oinn. Talisman - rapid application development for the grid. Bioinformatics, Vol. 19 Suppl. 1 2003, pages i212 - i214.

[5] Talisman'i projekti koduleheküIg.

http://www.ebi.ac.uk/collab/mygrid/service1/talisman/index.html, viimati külastatud 19.05.2004.

[6] Nigel McFarlane. An Introductory Tour of Mozilla's XUL.

http://www.informit.com/articles/article.asp?p=102016, viimati külastatud 19.05.2004.

[7] Eric Krock. A Taste of XUL.

http://www.mozilla.org/docs/codestock99/xul/xulfiles/v3_document.htm, viimati külastatud 19.05.2004.

[8] Ian T. Oeschger. XUL Genealogy: What Does XUL Have To Do With XML?

http://www.mozilla.org/docs/xul/xulnotes/xulnote_xml.html, viimati külastatud 19.05.2004.

[9] Neal Deakin. XUL Tutorial.

`http://www.xulplanet.com/tutorials/xultu`, viimati külastatud 19.05.2004.

[10] Estonian Grid. http://grid.eenet.ee, viimati külastatud 21.05.2004.

Lisa 1 - EP:NG komponendi XML

Data_Upload.epc.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<component xmlns:xinclude="http://www.w3.org/2001/XInclude" id="1"
 title="Data Upload" user_level="0" display="1">
 <about>
  <Version>1.0</Version>
  <Author>Patrick Kemmeren (email: patrick (at) ebi.ac.uk)</Author>
  <Author>Misha Kapushesky (email: ostolop (at) ebi.ac.uk)</Author>
  <Description>
   <p align="justify">On this screen you can upload your
   own data - either via a file, or from a URL, or by pasting it
   directly into provided text boxes.</p>
  </Description>
 </about>
 <context_help id="0">
  <xinclude:include href="Data_Upload.doc.xml#xpointer
  (/epc_doc/ctxt_topic[@id=0])" />
 </context_help>
 <section title="Data location" id="0" default_active_subsec="0" output="0">
  <context_help id="1">
   <xinclude:include href="Data_Upload.doc.xml#xpointer
   (/epc_doc/ctxt_topic[@id=1])" />
  </context_help>
  <subsection title="Upload files" id="0">
   <context_help id="2">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=2])" />
   </context_help>
   <shorthelp>Upload files from your computer's disk using
   the controls below</shorthelp>
   <input type="file" name="src_file" size="30">
    <shorthelp>Location of the data matrix on your computer</shorthelp>
    <context_help id="3">
```

```
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=3])" />
   </context_help>
   <longhelp />
  </input>
  <input type="file" name="src_file_row_annot" size="30">
   <shorthelp>Location of the row annotations, if available</shorthelp>
   <context_help id="5">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
   <longhelp />
  </input>
  <input type="file" name="src_file_column_annot" size="30">
   <shorthelp>Location of the column annotations, if available</shorthelp>
   <context_help id="5">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
  </input>
  <input type="hidden" name="data_src" value="file" />
 </subsection>
 <subsection title="Copy-paste data" id="1">
  <context_help id="4">
   <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=4])" />
  </context_help>
  <shorthelp>Copy and paste the data from another program
  into the areas below</shorthelp>
  <textarea name="src" rows="5" cols="50">
   <shorthelp>Paste the data matrix here</shorthelp>
   <longhelp />
  </textarea>
  <textarea name="src_row_annot" rows="5" cols="50">
   <shorthelp>Paste the row annotations here</shorthelp>
   <context_help id="5">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
   <longhelp />
  </textarea>
  <textarea name="src_column_annot" rows="5" cols="50">
   <shorthelp>Paste the column annotations here</shorthelp>
   <context_help id="5">
```

```
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
   <longhelp />
  </textarea>
  <input type="hidden" name="data_src" value="data" />
 </subsection>
 <subsection title="Provide URL" id="2">
  <shorthelp>Let EP retrieve the data from the URLs you
  provide below</shorthelp>
  <context_help id="6">
   <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=6])" />
  </context_help>
  <input type="text" name="src_url" size="50">
   <shorthelp>The data matrix URL</shorthelp>
   <longhelp />
  </input>
  <input type="text" name="src_url_row_annot" size="50">
   <shorthelp>The row annotations URL</shorthelp>
   <context_help id="5">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
   <longhelp />
  </input>
  <input type="text" name="src_url_column_annot" size="50">
   <shorthelp>The column annotations URL</shorthelp>
   <context_help id="5">
    <xinclude:include href="Data_Upload.doc.xml#xpointer
     (/epc_doc/ctxt_topic[@id=5])" />
   </context_help>
   <longhelp />
  </input>
  <input type="hidden" name="data_src" value="url" />
 </subsection>
</section>
<section title="Data format" id="1" output="0">
 <context_help id="7">
  <xinclude:include href="Data_Upload.doc.xml#xpointer
  (/epc_doc/ctxt_topic[@id=7])" />
 </context_help>
 <select name="filetype">
  <context_help id="8">
```

```
   <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=8])" />
  </context_help>
  <shorthelp>Select the type of your data file</shorthelp>
  <option value="AUTO">Automatic file type detection</option>
  <option value="XLS">Microsoft Excel spreadsheet</option>
  <option value="SPACE">Single space delimited data</option>
  <option value="TAB">Tab-delimited data</option>
  <option value="VWS">Any-length whitespace delimited data</option>
  <option value="CUSTOM">Custom delimiter (specify below)</option>
 </select>
 <input type="text" name="cust_delim" size="5">
  <context_help id="8">
   <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=8])" />
  </context_help>
  <shorthelp>You can enter a custom delimiter here</shorthelp>
 </input>
 <input type="text" name="nr_annot_columns" size="5">
  <shorthelp>
   Nr. of columns after
   <b>1</b>
   to use for annotation
   <i>(blank for none)</i>
  </shorthelp>
 </input>
 <input type="text" name="nr_annot_rows" size="5">
  <shorthelp>
   Nr. of rows after
   <b>1</b>
   to use for annotation
   <i>(blank for none)</i>
  </shorthelp>
 </input>
</section>
<section title="Dataset metadata" id="2" output="0">
 <context_help id="10">
  <xinclude:include href="Data_Upload.doc.xml#xpointer
  (/epc_doc/ctxt_topic[@id=10])" />
 </context_help>
 <input type="text" name="dataset_description" size="20"
  style="width: 250px">
  <shorthelp>Enter a description for your data</shorthelp>
 </input>
```

```
 <dynamic_select name="dataset_species" call="list_species"
  required="1">
  <shorthelp>Select data species</shorthelp>
 </dynamic_select>
 <dynamic_select name="dataset_type" call="list_dataset_types"
  required="1">
  <context_help id="11">
   <xinclude:include href="Data_Upload.doc.xml#xpointer
    (/epc_doc/ctxt_topic[@id=11])" />
  </context_help>
  <shorthelp>Select data type</shorthelp>
 </dynamic_select>
</section>
<section title="Submit" id="4" output="0">
 <context_help id="12">
  <xinclude:include href="Data_Upload.doc.xml#xpointer
  (/epc_doc/ctxt_topic[@id=12])" />
 </context_help>
 <dynamic_input type="hidden" name="session_id"
  call="get_session_id" />
 <dynamic_input type="hidden" name="username"
  call="get_username" />
 <dynamic_input type="hidden" name="folder_dst"
  call="get_username" />
 <input type="hidden" name="src_name" value="" />
 <input type="submit" name="submitted" value="Proceed" />
</section>
<action>
 <!--  target to the next component - data selection  -->
 <xinclude:include href="epc_targets.xml#xpointer
  (/epc_target//component[@id=1]/*)" />
 <target id="0" comp_id="2">
  <activate_subsection section_id="0" subsection_id="1" />
  <activate_subsection section_id="2" subsection_id="0" />
 </target>
 <minimize_on_success>1</minimize_on_success>
 <success_test match_regex="uploaded successfully:
  (.*) rows and (.*) columns" />
 <!--
    the output name should correspond to whatever input names
    may be expected in the target component(s)
 -->
 <output name="session_id" label="session id"
  match_regex="<session_id>(.+?)</session_id>" display="---">
  <match_errors regex="(No session id found)" err_code="2001" />
```

```
    <match_errors regex="(Invalid session id)" err_code="2002" />
    <match_errors regex="(Session has expired)" err_code="2003" />
   </output>
   <output name="filetype" label="Uploaded file data type"
    match_regex="<filetype>(.+?)</filetype>">
    <match_errors regex="(No data supplied)" err_code="2005" />
    <match_errors regex="(Could not identify file type)" err_code="2006" />
   </output>
   <output name="DATANAME" label="Name of selected dataset"
    match_regex="<dataset_name>(.+?)</dataset_name>">
    <match_errors regex="(Dataset name already exists)" err_code="2004" />
   </output>
   <output name="row_count" label="Selected row count"
    match_regex="<rows>(.+?)</rows>" />
   <output name="column_count" label="Selected column count"
    match_regex="<columns>(.+?)</columns>" />
   <output name="dataset_src" label="Uploaded dataset ID"
    match_regex="<dataset_id>(.+?)</dataset_id>" display="---" />
  </action>
</component>
```