



TAMPEREEN
AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

JIRA-ohjelmiston käyttö ohjelmistoprosessin hallinnassa

Ville Nuutinen

Tietojenkäsittelyn koulutusohjelma
toukokuu 2008
Työn ohjaaja: Paula Hietala

TAMPERE 2008



Tekijä(t)	Ville Nuutinen	
Koulutusohjelma(t)	Tietojenkäsittely	
Opinnäytetyön nimi	JIRA-ohjelmiston käyttö ohjelmistoprosessin hallinnassa	
Työn valmistumis- kuukausi ja -vuosi	toukokuu 2008	
Työn ohjaaja	Paula Hietala	Sivumäärä: 28

TIIVISTELMÄ

Universomolla oltiin kehittämässä uutta ohjelmistotuotannon prosessia. Jotta projektien etenemistä prosessin suunnasta voitaisiin seurata, haluttiin selvittää mikä olisi paras sovellus tarkoitukseen. Yksi sovellusehdokkaista oli JIRA-niminen ongelmanseurantaohjelmisto. Opinnäytetyön tavoitteeksi asetettiin sellaisen sovelluksen toteutus, jolla JIRA:n soveltuvuus tehtävään voitaisiin selvittää. Sovelluksen tulisi mahdollistaa satojen tehtävien syöttö JIRA:n tietokantaan. Tehtävät luettaisiin kehysprojektitiedostosta, johon yhdistettäisiin projektikohtaiset tehtävät.

Opinnäytetyössä selvitettiin sovelluksen ohjelmoinnissa käytetyt tekniikat ja niiden perusteet. Työn lähteenä käytettiin pääasiassa sovellusten dokumentaatiota. Tekniikoista etsittiin informaatiota internetistä, sekä alan teoksista.

Opinnäytetyön tuloksena saatiin valmiiksi sovellus, joka lukee kehysprojektin tiedot ja yhdistää nämä projektikohtaisiin tietoihin. Tämän jälkeen sovellus tuottaa tiedoston, jonka JIRA voi lukea tietokantaansa.

Yritys itse hyödyntää opinnäytetyötä ja sovellusta, sekä sen lähdekoodia JIRA:n soveltuvuuden selvittämiseen. Ohjelmaa voidaan myöhemmin muokata tulevaan käyttöön, mikäli yritys päättyy JIRA:n käyttämiseen projektien sekä prosessin toteutumisen hallinnassa ja seurannassa. Yrityksen oma tutkimus parhaasta sovelluksesta prosessin seurantaan ja projektien hallintaan ei valmistunut tämän opinnäytetyön kirjoitustyön aikana.



Author(s)	Ville Nuutinen	
Degree Programme(s)	Business Information Systems	
Title	Controlling software production process with JIRA	
Month and year	May 2008	
Supervisor	Paula Hietala	Pages: 28

ABSTRACT

The company Universomo was developing a new software production process. Tracking the progress of projects according to the process required finding out which application would be most fitting for the task. One of the possibilities was JIRA, an issue tracking software. Objective for the thesis was to produce a tool for studying the feasibility of JIRA. The produced application should enable entering hundreds of tasks into JIRA's database. The tasks would be read from a template project file which would be combined with a project-specific task file.

The techniques used in the project and their basics will be told in the thesis. As an information source for the project, mostly the documentation of the specific languages and applications was used. Information on the techniques was researched online and from books relating to the task at hand.

The outcome is a program that combines information from the template project file with a project-specific file. Finally, the program converts this information into a form which can be input into JIRA.

The company uses this thesis and the created program with its source code to find out if JIRA is the program of choice in this case. The program can later be modified to be used if JIRA is chosen for controlling and observing projects and the realization of the process. The company's own research for finding out the best software for the purpose wasn't finished at the time of writing this thesis.

Sisällysluettelo

1	Johdanto.....	2
2	Opinnäytetyön tausta ja tavoite	6
3	Tekniikat.....	8
3.1	JIRA.....	8
3.2	XML ja skeema	10
3.3	XSLT	13
3.4	Apache Ant.....	15
3.5	JellyScript.....	16
4	Toteutus	19
4.1	JIRA:n asennus.....	19
4.2	Sovelluksen spesifiointi ja ohjelmointi	20
5	Arvio JIRA:n soveltuvuudesta prosessin hallintaan.....	24
6	Jatkokehitys	25
6.1	Mahdollisia muutoksia JIRA:an.....	25
6.2	Muutokset nykyiseen sovellukseen	25
6.3	SOAP:in käyttöönotto	26
6.4	Lisäsovellusten rakentaminen	26
7	Lopuksi	27
	LÄHTEET.....	28

1 Johdanto

Mobiilipeliyritys Universomon johto halusi tehostaa ohjelmistotuotantonsa menetelmiä ja oli kehittämässä uutta ohjelmistotuotantoprosessia. Uuden prosessin tavoitteena oli saada projektien resurssitarpeen ennakointi ja käytettävien resurssien seuranta helpommaksi, sekä seurata projektin etenemistä sille asetettuja aikataulutavoitteita vasten.

Uutta prosessia käyttävien projektien seurantaan tarvittiin työkalu. Tähän tarkoitukseen soveltuvaa sovellusta ei haluttu työmäärästä johtuen toteuttaa puhtaalta pöydältä. Yrityksellä oli ollut jo pitkään käytössä ongelmienseurantaan tarkoitettu JIRA-ohjelmisto, joten sen käyttöä prosessiin liittyvien tehtävien seurantaan haluttiin tutkia.

Käytännössä tämä tarkoitti sitä, että yrityksellä oli tarve sovellukselle, joka mahdollistaisi satojen työtehtävien syöttämisen JIRA:n tietokantaan, jokaista alkavaa projektia kohden. Tämän informaation tuli olla helposti muunnettavissa JIRA:n lukemaan JellyScript-muotoon.

Projektissa selvitin XML, XSLT, Apache Ant ja JellyScript-ohjelmointikielten toiminnan periaatteet. Tutkimustyötä tein sekä alan kirjallisuuden, että internetistä löytyvän dokumentaation parissa. Ohjelmiston toteutuksessa tarvitsin huomattavasti enemmän kielten omia spesifikaatioita valmistajan omilta sivuilta, koska ne olivat kirjallisuutta paremmin ajan tasalla. Itse ohjelmiston toteutin Eclipse-ohjelmointiympäristössä Apache Ant -skriptauskielellä.

Raportissa selvitän projektissa käytettyjen ohjelmointikielten perustoiminnan ja projektissa tarvitut toiminnot. Raportista selviää myös toteutetun sovelluksen rakenne ja toiminta.

2 Opinnäytetyön tausta ja tavoite

Tausta

Opinnäytetyön toimeksiantajana toimi Universomo Oy, joka on vuonna 2002 perustettu, pääosin Tampereella toimiva mobiilipeliyritys. Tällä hetkellä Universomolla on töissä noin 60 henkeä Tampereella, Helsingissä ja San Diegossa. Universomo on osa THQ Wireless:iä.

Universomo oli kehittämässä ohjelmistokehitykselleen uutta prosessia, joka toimisi ohjelmistoprojektien reseptinä. Prosessissa määriteltäisiin missä järjestyksessä ohjelmistoon toteutetaan ominaisuuksia ja milloin projekti on valmis. Projektissa pitää suorittaa prosessissa versiolle määrätty tehtävät hyväksytysti, jotta projekti saavuttaa kyseisen version tilan. Prosessissa voidaan kuitenkin määritellä vain jokaiselle projektille yhteiset tehtävät, näiden lisäksi on selvittävä jokaiselle projektille yksilölliset tehtävät. Tehtävä, jossa vaaditaan ohjelmoimaan ympyröitä piirtävä metodi, ei ole tarpeellinen projektissa jossa piirretään pelkkiä neliöitä. Tällaiset tehtävät on jätettävä prosessin ulkopuolelle. Ne voidaan kuitenkin kerätä projektikohtaiseen tehtävälistaan, josta on hyötyä myöhemmin, kun projektia toteutetaan.

Prosessia edeltäneet menetelmät projektien toteuttamiselle toimivat hyvin, kun yrityksessä oli vielä alle kymmenen henkeä, mutta henkilöstömäärä kasvoi nopeasti ja työtavassa alkoi näkyä heikkouksia, joihin oli puututtava mahdollisimman pian. Osalla henkilöstöstä oli muihin verrattuna huomattavasti enemmän töitä ja resurssitarvetta oli hankala ennakoida.

Uuden prosessin tarkoituksena oli saada projektien valmistuminen ja resurssien seuranta sekä hallinta selkeämmäksi. Prosessia suunnittelivat pääasiassa yrityksen johtohenkilöstö ja henkilöstöhallinto. Suunnitteluun osallistuivat myös tuottajat ja osa projektien pääohjelmoijista. Ensimmäinen toteuttamiskelpoinen versio uudesta prosessista oli valmiina syksyllä 2007.

Tavoite

Jotta uuden prosessin toteutumista voitaisiin seurata, tarvittiin tarkoitukseen soveltuva ohjelmisto. Koska yrityksessä oli käytetty JIRA-nimistä ongelmanseurantaohjelmistoa (issue tracking software) jo vuosia, oli huomattu sen potentiaali myös projektinseurantakäytössä. JIRA:a käyttämällä pystyttäisiin välttämään uusien ja kalliiden ohjelmien hankinta yritykselle.

Yrityksessä tiedettiin, että prosessinhallintaa varten itse JIRA-ohjelmistoon ei ole tarpeellista tehdä muutoksia, koska tarpeelliset tietotyypit voidaan jo löytää JIRA:sta. JIRA:n tietokanta muodostuu käyttäjistä ja projekteista. Projektit vuorostaan sisältävät versioita, komponentteja ja ongelmia. JIRA:an voidaan määritellä myös tehtäviä ongelmien sijaan, jos uusi tyyppi ”tehtävä” ensin määritellään ongelmatyyppiin joukkoon.

Vaikka JIRA:an ei tarvinnut tehdä muutoksia, jonkinlainen apuohjelma oli tarpeen. Prosessissa määriteltiin kehysprojekti, joka sisälsi kaikille projekteille yhteiset tehtävät. Näihin tehtäviin lisättiin projekteille yksilölliset tehtävät. Yhteensä satojen tehtävien syöttö JIRA:n tietokantaan olisi tuntien, ellei jopa päivien urakka, mutta apuohjelmalla tietojen syötön voisi automatisoida.

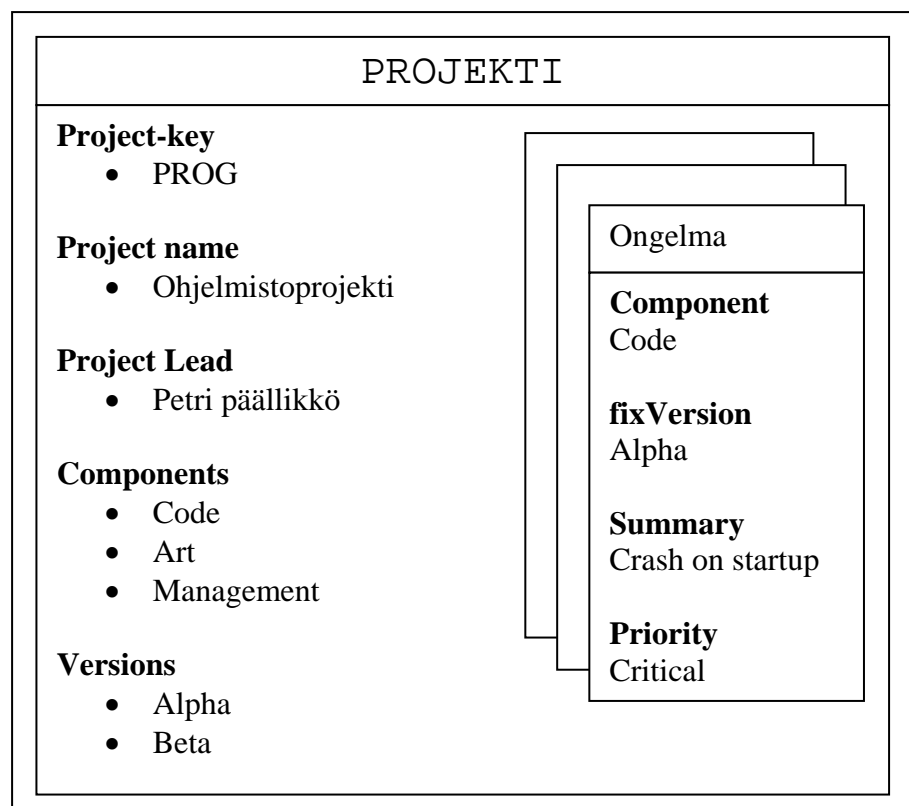
Opinnäytetyön pääasiallinen tavoite oli tuottaa tällainen sovellus. Sovelluksen syötteenä toimisi kehysprojektin sekä projektikohtaisen tehtävälistan tiedot. Sovellus yhdistäisi ne ja tuottaisi ulos JellyScript-tiedoston, jota JIRA pystyisi lukemaan. Koska sovelluksen tarkoituksena on mahdollistaa suurien tehtävämäärien syöttö JIRA:n tietokantaan, sen avulla voitaisiin helpommin testata JIRA:n toimivuus ja soveltuvuus prosessin valvontaan ja projektien hallintaan käyttämättä silti suurta määrää henkilöstöresursseja.

3 Tekniikat

3.1 JIRA

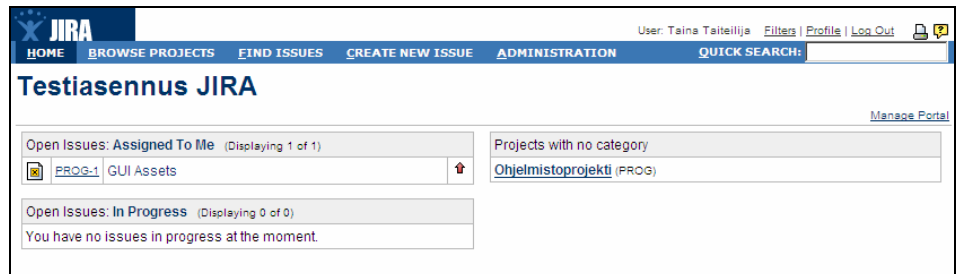
JIRA on Atlassian Inc. –ohjelmistotalon toteuttama ongelmanseurantaohjelmisto. JIRA mahdollistaa monipuolisen ja nopean ongelmien seurannan ja h. JIRA:n tietokantaan voidaan määritellä erilaisia tietoja, kuten käyttäjiä, projekteja, versioita ja ongelmia. Käyttäjät voivat raportoida ongelmia projekteissa ja ongelmamäärittelysten mukaan asiat siirtyvät korjattavaksi projektin tietyille henkilöille, jotka on määritetty oletuskäsittelijöiksi. Ongelmiin kirjoitetaan ongelman kuvaus, vaiheet sen toistamiseen ja mitä testattavan ohjelmiston versiota ongelma koskee. Tällöin käsittelijän on helpompi itse toistaa ongelma ja korjata se.

Kuviosta 1 voi nähdä projektien ja ongelmien rakenteen JIRA:ssa. Projekti sisältää tiedoissaan projektin avaimen, nimen, vastuuhenkilön, komponentit ja versiot, sekä ongelmia ja/tai tehtäviä. Ongelmat ja tehtävät ovat rakenteeltaan samanlaisia: ne sisältävät tiedon komponentista, versiosta jota ongelma tai tehtävä koskee, lyhyen kuvauksen, sekä tärkeysasteen.



Kuvio 1. Projektin ja sen sisältämän ongelman rakenne JIRA:ssa.

Käyttäjälle JIRA näyttää tavalliselta verkkosivustolta (Kuvio 2), eikä JIRA:n käyttö vaadi työasemalta tavallisen selaimen lisäksi mitään erityisteknologioita tai ohjelmistoja. Kuviosta voi nähdä joitakin projektin tietoja. Kun JIRA viittaa projektiin, se yleensä tapahtuu projektin avaimella.



Kuvio 2. Tavanomainen näkymä käyttäjän kirjaututtua JIRA:an.

Seuraava esimerkki ohjelmistovirheen raportoinnista ja korjauksesta havainnollistaa tavanomaista JIRA:n käyttöä.

- 1) Testaaja saa testattavakseen ohjelmistoprojektin.
- 2) Testauksessa projektista löytyy ohjelmistovirhe, joka näyttää liittyvän ohjelman koodiin. Ohjelma kaatuu aina johonkin tiettyyn tilanteeseen tullessaan.
- 3) Testaaja raportoi ohjelmistovirheen JIRA:an. Raporttiin määritellään, että ongelma on ohjelmiston koodissa. Valittu tieto ohjaa ongelman automaattisesti ohjelmointipuolen päävastuuhenkilölle.
- 4) Kirjautuessaan sisään ohjelmoija näkee tehtävälistassaan uuden tehtävän. Mikäli ohjelmoija ei itse osaa ratkaista tehtävää, hän voi ohjata sen edelleen toiselle ohjelmoijalle.
- 5) Ohjelmoija korjaa ongelman ja merkitsee sen JIRA:an ratkaistuksi. Tällöin tehtävä palautuu testaajalle tarkistettavaksi.
- 6) Testaaja näkee JIRA:an kirjautuessaan avoimen tehtävän kohdallaan ja tarkistaa vieläkö ongelma toistuu uudella ohjelmistoversiolla. Mikäli ongelma toistuu vielä uudelleen, voi testaaja palauttaa sen ohjelmointipuolelle korjattavaksi. Jos ongelma kuitenkin on korjattu, testaaja merkitsee sen suljetuksi.

3.2 XML ja skeema

XML

XML, eli eXtensible Markup Language (laajennettava merkitäkieli) on niin sanottu metakieli: sillä voidaan määritellä muita kieliä. Yksi esimerkki XML-pohjaisesta kielestä on verkkosivujen tekoon käytetty XHTML, joka on käytännössä sama kuin vanhempi HTML-kieli, mutta sen laatuvaatimukset ovat tarkemmat. XML-dokumentin oikeamuotoisuus (well-formedness) vaatii dokumentilta seuraavia asioita (Extensible Markup Language 2006):

- Se sisältää yhden tai useamman elementin.
- Se sisältää vain yhden juurielementin.
- Kaikki avatut elementit myös suljetaan.
- Kaikki elementtien ominaisuudet (attribuutit) ovat lainausmerkkien sisällä.

Määrittelynsä mukainen XML-tiedostoa tulkitseva ohjelmisto ei saa korjata XML:ssä esiintyviä virheitä, se voi ainoastaan ilmoittaa virheestä (Harold 2000: 165). Tämän vuoksi on tärkeää noudattaa oikeamuotoisen XML-tiedoston määrittelyä. Koodiesimerkistä 1 voi nähdä XML-kielen perussyntaksin. Esimerkissä on yksinkertainen kirjatietokanta.

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

<kirjat>
  <nide nimi="Hyperion"/>
  <nide nimi="Endymion"/>
</kirjat>
```

Koodiesimerkki 1. Yksinkertainen XML-dokumentti.

Jokainen elementin esittely aloitetaan pienempi kuin -merkillä (<) ja lopetetaan suurempi kuin -merkillä (>). Jokainen avattu elementti täytyy lisäksi sulkea kirjoittamalla samankaltainen elementti, jonka toiseksi alkuun on lisätty kauttaviiva (/). Mikäli elementin sisään ei tule tietoa, sen voi sulkea jo aloituselementissä lisäämällä elementin loppuun kauttaviivan. Koodiesimerkissä 1 lapsielementti ”nide” sisältää attribuutin ja attribuutilla on arvo ”nimi”, joka on kirjoitettu lainausmerkkien sisään.

Tavallisesti XML-dokumenttien alkuun on tapana lisätä ilmoitus dokumentin muodosta. Tämä ilmoitus kertoo käyttäjälle ja mahdolliselle sovellukselle, että kyseessä on XML-tiedosto ja mikä merkistö on ollut tiedoston kirjoittamisessa käytössä. Merkistö on hyvä ilmoittaa, mikäli tiedostossa käytetään esimerkiksi skandinaavisia merkkejä. Toisella henkilöllä ei välttämättä ole käytössä sama merkistö ja tällöin erikoismerkit saattavat korvautua omituisilla merkkijonoilla, jos merkistöä ei ole kerrottu dokumentin alussa.

XML-Skeema

XML-Skeema on XML-teknologia, jolla voidaan kuvata XML-dokumentin rakenne. Kuvaustiedostolla voidaan määritellä mitä XML-Skeemaa hyödyntävässä XML-dokumentissa saa olla ja mitkä tiedot ovat pakollisia. Skeemassa voidaan myös pakottaa XML-dokumentin elementeille jokin tietty järjestys. Koodiesimerkki 2:ssa on XML-Skeematiedosto koodiesimerkin 1 kirjatietokannalle.

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

  <xs:element name="kirjat">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nide" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="nide">
    <xs:complexType>
      <xs:attribute name="name" use="required">
        <xs:simpleType>
          <xs:list itemType="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Koodiesimerkki 2. Yksinkertaisen XML-dokumentin XML-Skeema-tiedosto.

Koodiesimerkki 2:sta voi huomata, että jo yksinkertaisenkin XML-tiedoston XML-Skeematiedostosta voi tulla pitkä ja monimutkainen. Koodiesimerkki 2:ssa esitellään kaikki dokumentissa esiintyvät elementit ja määritellään niiden rajoitukset. Komennolla `xs:sequence` määritellään, missä järjestyksessä elementtien täytyy esiintyä toisten elementtien sisällä. Tässä tapauksessa elementtityyppiä on vain yksi: `nide`. `Nide`-elementtejä voi määrittelyn mukaan olla kirjat-elementin sisällä rajaton määrä.

`Nide`-elementin määrittelyssä kuvataan, että sen on pakko sisältää attribuutti `"nimi"`, jonka täytyy olla merkkijonomuodossa. Käytännössä tällainen rajoitus ei rajoita syötettyä dataa mitenkään, koska merkkijono voi sisältää mitä tahansa tekstimuotoista dataa. Mikäli tarkoitus sitä vaatisi, voitaisiin tieto rajoittaa esimerkiksi päivämäärämuotoiseksi tai numeromuotoiseksi. Mikäli aiemmin esitellyssä kirjatietokannassa haluttaisiin ottaa käyttöön koodiesimerkissä 2 esitelty XML-Skeema, täytyisi kirjatietokannan juurielementti muokata koodiesimerkin 3 kaltaiseksi.

```
<kokoelma xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xsi:noNamespaceSchemaLocation="kirjaskeema.xsd">
```

Koodiesimerkki 3. Uusi aloituselementti.

Vaikka XML-Skeeman käyttö voi vaikuttaa monimutkaiselta, hyvin tehty skeematiedosto estää tehokkaasti vääränlaisen tiedon syöttämistä ja voi näin säästää monilta ongelmilta myöhemmissä vaiheissa. Jos XML-tiedosto menee skeemataarkistuksesta (validation) läpi, voi olla varma, että tieto on aina oikeassa muodossa.

3.3 XSLT

XSLT, eli Extensible Stylesheet Language Transformations, on XML-muotoinen kieli, jolla voidaan helposti lukea XML-tiedoston data ja tulostaa se eri muodossa. XSLT-kielessä on komennot XML-elementtien ja attribuuttien lukuun, sekä tiedon erottamiseen niiden sisältä. XSLT-komennoilla voi myös yhdistää XML-tiedostoja.

XSLT:n rakennetta on helpoin selittää esimerkeillä. Käytetään tähän aiemmin luotua yksinkertaista XML-dokumenttia. Koodiesimerkin 5 XSLT-tiedostolla voidaan yhdistää aiempi kirjatietokanta koodiesimerkin 4 tietokantaan.

```
<kirjat>
  <nide nimi="Tripodien aika" />
</kirjat>
```

Koodiesimerkki 4. Toinen yksinkertainen XML-dokumentti.

Tässä tilanteessa toimisi seuraavanlainen XSLT-tiedosto:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes" />
  <xsl:param name="file" />
  <xsl:variable name="kanta2" select="document($file)" />
  <xsl:template match="/">
    <kokoelma>
      <kirjat>

        <xsl:for-each select="/kokoelma/kirjat/nide">
          <xsl:element name="nide">
            <xsl:attribute name="nimi"><xsl:value-of
select="@nimi" /></xsl:attribute>
          </xsl:element>
        </xsl:for-each>

        <xsl:for-each
select="$kanta2/kokoelma/kirjat/nide">
          <xsl:element name="nide">
            <xsl:attribute name="nimi"><xsl:value-of
select="@nimi" /></xsl:attribute>
          </xsl:element>
        </xsl:for-each>

      </kirjat>
    </kokoelma>
  </xsl:template>
</xsl:stylesheet>
```

Koodiesimerkki 5. XSLT-tiedosto kahden tietokannan yhdistämiseen.

Koodiesimerkin 5 toisella rivillä ilmaistaan, että kyseessä on XSLT-tiedosto. Xsl:output-elementissä asetetaan tulosteen muoto. Seuraavaksi luetaan XSLT-tiedostolle parametrinä annettu tiedostonimi "file". Tiedosto asetetaan muuttujaan "kanta2".

Jokainen xsl-alkuinen elementti tarkoittaa XSLT-komentoa. Jos elementissä ei ole xsl-alkua, se tulee tulosteeseen sellaisenaan. Tällöin kokoelma- ja kirjat-elementit tulevat tulosteeseen niin kuin ne on XSLT-tiedostoon kirjoitettu.

Esimerkin xsl:for-each-komennolla suoritetaan silmukka jokaista nide-elementtiä kohden. Elementin xsl:for-each sisällä olevat komennot suoritetaan siis kerran jokaista nidettä kohti. Komento xsl:element luo elementin tulosteeseen. Komento xsl:attribute taas luo attribuutin tämän kyseisen elementin sisään.

Jos käytetään aiemmin esiteltyjä tietokantoja ja koodiesimerkin 5 XSLT-muunnosta, saadaan uusi tietokanta, joka sisältää molempien tietokantojen tiedot (Koodiesimerkki 6).

```
<kirjat>
  <nide nimi="Hyperion"/>
  <nide nimi="Endymion"/>
  <nide nimi="Tripodien aika"/>
</kirjat>
```

Koodiesimerkki 6. Yhdistetty tietokanta.

3.4 Apache Ant

Apache Ant (Another Neat Tool), on työkalu ohjelmistojen kääntöprosessin automatisointiin ja se on tarkoitettu Java-ohjelmien kääntöön käyttöjärjestelmästä riippumatta. Apache Ant:in kääntöprosessi kuvaillaan XML-tiedostossa, jota kutsutaan kääntötiedostoksi (buildfile). Kääntötiedostoon kirjoitetaan käännössä suoritettavat vaiheet. Apache Ant:in kääntötiedosto sisältää yhden tai useamman projektin (projects), kohteita (targets) ja tehtäviä (tasks). Projektit sisältävät kohteita, jotka vuorostaan sisältävät tehtäviä (Niemeyer & Poteet 2003: 22-26). Kääntötiedostoa ajettaessa valitaan mikä projekti ja mitä kohteita sen sisältä suoritetaan.

Apache Ant:in saa useiden ohjelmointiympäristöjen mukana valmiina. Esimerkiksi Eclipse, NetBeans ja IntelliJ IDEA sisältävät Apache Ant:in. (Apache Ant User Manual 2008) Apache Ant:ia tarvittiin projektissa XML-tiedostojen XML-Skeema-tarkistukseen (Koodiesimerkki 7) ja XSLT-komentojen suorittamiseen (Koodiesimerkki 8).

```
<xmlvalidate failonerror="true">
  <fileset dir="." includes="kirjat.xml"/>
  <attribute name="http://xml.org/sax/features/validation"
value="true"/>
  <attribute
name="http://apache.org/xml/features/validation/schema"
value="true"/>
  <attribute name="http://xml.org/sax/features/namespaces"
value="true"/>
</xmlvalidate>
```

Koodiesimerkki 7. Apache Antin käyttö XML-Skeema-tarkistukseen.

Koodiesimerkissä 7 on Ant-tehtävä (task), jolla tarkistetaan projekti.xml-tiedosto siinä ilmoitettua schema-tiedostoa vasten. Failonerror-parametrissa määritellään, että build.xml-tiedoston suorittaminen lopetetaan, mikäli validointi epäonnistuu.

```
<xslt style="yhdistä.xslt" in="kirjat.xml"
out="kaikkikirjat.xml" force="true">
  <param name="file" expression="toisetkirjat.xml"/>
</xslt>
```

Koodiesimerkki 8. Apache Antin käyttö XSLT-koodin ajamiseen.

Koodiesimerkissä 8 on tehtävä, jolla suoritetaan xslt-koodia ”yhdistä.xslt”-tiedostosta. XSLT-muunnokseen annetaan muutettavaksi tiedosto ”kirjat.xml” ja lopputulos otetaan ”kaikkikirjat.xml”-tiedostoon. Lisäksi XSLT-koodille välitetään yksi lisätiedosto: ”toisetkirjat.xml”. Tähän voidaan viitata XSLT-koodissa muuttujana ”file”, kuten XSLT-esimerkissä tehtiinkin.

3.5 JellyScript

JellyScript on Apachen kehittämä työkalu, jolla voidaan muuntaa XML-kieltä ajettavaksi koodiksi. (Apache Commons, Jelly: Executable XML 2007). JIRA:n kanssa JellyScript-kieltä voi käyttää erilaisten tietojen syöttämiseen ja muokkaamiseen. JellyScript toimii siis ikäänkuin se olisi JIRA:n komentorivi. Kommentojen rakenne on helpointa nähdä koodiesimerkki 9:stä.

```
<JiraJelly xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:jira="jelly:com.atlassian.jira.jelly.enterprise.JiraTagLib">

  <jira:CreateProject key="PROG" name="Ohjelmistoprojekti"
lead="projektijohtaja">

    <jira:AddComponent name="Code" description="Code
responsibilities" componentLead="koodari"/>
    <jira:AddComponent name="Art" description="Art
responsibilities" componentLead="graafikko"/>
    <jira:AddVersion name="Alpha"/>
    <jira:AddVersion name="Beta"/>

  </jira:CreateProject>

  <jira:CreateIssue project-key="PROG" issueType="Task"
summary="GUI Assets" priority="Major" components="Art"
assignee="graafikko" reporter="reportteri"
duplicateSummary="ignore" fixVersions="Alpha"/>

</JiraJelly>
```

Koodiesimerkki 9. Yksinkertainen JellyScript-tiedosto

Koodiesimerkissä 9 on komento

```
<JiraJelly xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:jira="jelly:com.atlassian.jira.jelly.enterprise.Jira
TagLib">
</JiraJelly>
```

Tällä komennolla ilmaistaan JIRAlle, että syötetty koodi on JellyScript-muotoista. Yksinkertaisuuden säilyttämiseksi kaikki tämän XML-tagin parametrit kannattaa antaa tässä muodossa. Mikäli XML:ää tuntee paremmin, voi xmlns-parametriä muuttaa, jos jira-nimiavaruus on jostain syystä käytetty johonkin toiseen tarkoitukseen.

Koodiesimerkin 9 komennolla

```
<jira:CreateProject key="PROG" name="Ohjelmistoprojekti"
lead="projektijohtajannimi">
```

luodaan JIRA:n tietokantaan uusi projekti, jonka koodinimi on ”PROG”, nimi ”Ohjelmistoprojekti” ja projektin ylin vastuhenkilö on ”projektijohtajannimi”. Kaikki lainausmerkkien sisällä oleva tieto on vapaasti muokattavissa.

Esimerkissä *key*-parametrin sisällä on nelikirjaiminen ”koodinimi” projektille ja vastaavasti *name*-parametrin sisällä on projektin nimi täydessä pituudessaan. Key-parametriä käytetään ongelmaraporteissa ja tehtävien luonnissa, joten sitä ei kannata asettaa täysin abstraktiksi numero- tai kirjainsarjaksi, vaan se on hyvä olla jonkinlainen lyhennys projektin nimestä. Key-parametrin koko ei kuitenkaan ole rajoitettu neljään kirjaimeen.

Lead-parametrin sisällä oleva tieto on oltava siinä muodossa, millä nimellä haluttu käyttäjä kirjautuu sisään JIRAan. Tietokantaan tallennettua projektia tarkasteltaessa lead-kohdassa näkyy kuitenkin henkilön koko nimi kirjautumistunnuksen sijaan.

Koodiesimerkin 9 komennolla

```
<jira:AddComponent name="Code" description="Code
responsibilities" componentLead="koodari"/>
```

luodaan projektille yksi komponentti. Komponentilla tarkoitetaan osastoa tai osaamisaluetta. Esimerkin komennolla luodaan ”Code”-niminen komponentti. Tälle komponentille voidaan antaa ohjelmointiin liittyviä tehtäviä. Komponentille annetaan kuvaus *description*-parametrissa. Komponenttien nimet ja kuvaukset ovat vapaasti valittavissa, JIRA ei rajoita niitä millään tavalla.

ComponentLead-parametrillä ilmaistaan kuka on kyseisen komponentin vastuuhenkilö. Jos kyseessä on ohjelmointikomponentti, voi tähän asettaa esimerkiksi pääohjelmoijan kirjautumistunnuksen. Tieto on vapaaehtoinen,

Koodiesimerkissä 9 on komento

```
<jira:AddVersion name="Alpha"/>
```

Tällä komennolla lisätään projektiin eri versioita. Näihin versioihin voidaan lisätä tehtäviä seuraavaksi esiteltävällä *CreateIssue*-komennolla.

Koodiesimerkissä 9 on komento

```
<jira>CreateIssue project-key="PROG" issueType="Task"
summary="GUI Assets" priority="Major" components="Art"
assignee="graafikonnimi" reporter="automaatti"
duplicateSummary="ignore" fixVersions="Alpha"/>
```

Koodiesimerkin 9 *CreateIssue*-komennolla luodaan projektiin tehtäviä. Tämä komento vaatii huomattavasti enemmän parametrejä, kuin yksikään aiemmin mainituista komennoista.

Project-key-parametrille ilmaistaan mihin projektiin luotava tehtävä liittyy. JIRA:n JellyScript-dokumentaation (Jelly Tags 2008) mukaan

CreateIssue-komennon voi antaa myös CreateProject tagien sisällä, jolloin project-key -parametriä ei tarvitse antaa.

IssueType-parametriin voidaan antaa mikä tahansa tyyppi, kunhan tyyppi on ensin määritelty JIRAn hallintatyökalujen avulla. Task-tyyppi löytyy tavallisesti suoraan JIRASTA, eikä sitä tarvitse erikseen määritellä. Mikäli JIRA:n tietokantaan on itse määritellyt uusia tehtävätyyppejä, niitä voi käyttää tässä.

Summary-parametrille (yhteenvedo) annetaan lyhyt kuvaus tehtävästä. Mikäli halutaan, voidaan komentoon lisätä parametri ”description”, johon voidaan syöttää yhteenvedoa tarkempi kuvaus tehtävästä. Yhteenvedo on kuitenkin hyvä kirjoittaa selkeäksi, sillä se näytetään, kun tehtäviin tehdään hakuja JIRAn sisällä. Yrityksen tapauksessa jokainen tehtävä oli määritelty toisaalla, joten tehtäviin tarvittiin vain lyhyt yhteenvedo.

Priority-parametriin määritellään miten korkealla tehtävä on prioriteeteissa. *Components*-parametriin määritellään mitä komponenttia tehtävä koskee. *Assignee*-parametriin vastaavasti määritellään, kuka ensisijaisesti lähtee tehtävää tekemään.

Reporter-parametrissa määritellään mitä tehtävän tiedoissa näkyy reporter-kohdassa, eli kuka tehtävän on tietokantaan luonut. Parametri vaatii käyttäjän, jolla on oikeudet Create Issue-komennon käyttöön. Lisäksi, mikäli reporter-parametriin asetettu käyttäjä ei ole sama joka on kirjautunut sisään, täytyy kirjautuneella käyttäjällä olla lisäksi Modify Reporter-oikeudet asetettuna käyttäjäprofiilissaan.

DuplicateSummary-parametrissa valitaan pitäisikö JIRAn hylätä CreateIssue-komento, mikäli samanlaisella summary-parametrilla on jo ilmoitettu yksi tehtävä. Parametrin asettaminen ”ignore”-valinnalle mahdollistaa saman tehtävän asettamisen useampaan versioon samassa projektissa. Tämä on tärkeää, koska ohjelmistoa halutaan testata joka versiossa ja tehtävän nimi on usein sama ”Testing.” Lopulta *fixVersions*-parametrissa asetetaan mihin versioon luotava tehtävä liitetään. Kun JIRASSA tarkastellaan version tehtäviä, kaikki tällä parametrilla asetetut tehtävät näkyvät aina kyseisen version yhteydessä.

4 Toteutus

4.1 JIRA:n asennus

Ohjelmiston testausta varten tarvittiin erillinen asennus JIRAssa. Erillisen instanssin etuna oli se, että komentoja voitiin testata vapaasti ilman riskiä tietokannan tuhoutumisesta tai vahingoittumisesta. Erillinen instanssi serveristä perustettiin työpisteelleni. Asennusta varten ladattiin ohjelmiston kokeiluversio Atllassianin sivuilta. Atllassian myönsi 30 päivän kokeilulisenssin pyynnöstä.

JIRAn asennus oli yksinkertainen, kun noudatti ruudulla olevia ohjeita. Asennuksen jälkeen JIRA pyysi kokeilulisenssiä, joka kopioitiin sille osoitettuun kenttään. Tämän jälkeen JIRAlle kerrottiin vielä järjestelmänvalvojan tunnus ja salasana, sekä mahdollinen virheilmoituksissa käytettävä sähköpostipalvelin.

Tietokannan varmuuskopiointi ja siirto testiympäristöön

Projektissa tuotiin yrityksen käytössä olevalta JIRA-palvelimelta tietokanta tähän erilliseen asennukseen. Tietokannan sai tulostettua xml-tiedostoon siirtymällä JIRA:n hallinnointisivustolle yläreunan ”administration”-linkin kautta hallinnointioikeudet omaavalla käyttäjätunnuksella. Tietokannan vietiin ja tuontiin liittyvät linkit löytyivät ”Import & Export”-otsikon alta sivupalkista. ”Backup Data to XML”-toiminnolla tietokannan sai tallennettua tiedostoon. Sen alta löytyvä ”Restore Data from XML”-toimintoa taas käytettiin lukemaan tietokanta testiympäristön JIRA-asennukseen.

JellyScriptauksen mahdollistaminen JIRAssa

Koska Jellyskriptauksella on mahdollista rikkoa monta asiaa JIRA-tietokannassa, se on asennuksen jälkeen asetettu oletuksena pois päältä. Jotta se toimisi JIRAssa, se täytyi asettaa päälle konfiguraatietiedostoista tai vaihtoehtoisesti käynnistysparametreillä. JIRA:n käynnistystiedostosta löytyi seuraavan kaltainen rivi:

```
set JAVA_OPTS=-server %JAVA_OPTS%
```

Tälle riville lisättiin jellyn käytön mahdollistava parametri, jolloin rivi näytti seuraavalta:

```
set JAVA_OPTS=-server -Djira.jelly.on=true %JAVA_OPTS%
```

Palvelimen uudelleenkäynnistyksen jälkeen oli Jelly-koodin syöttö mahdollista Administration sivun ”Jelly Runner”-linkin alta.

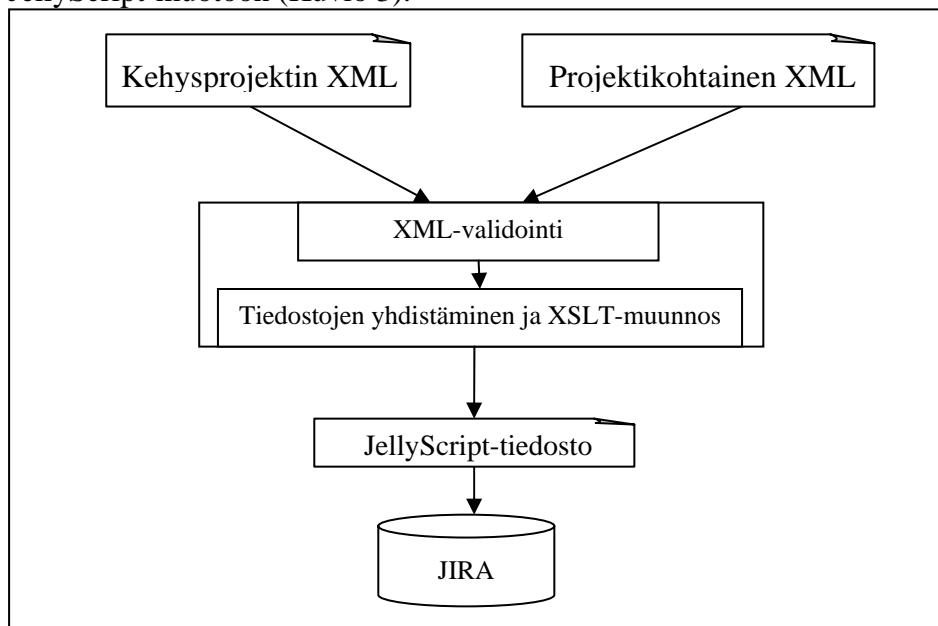
JellyScriptauksen testaus

Jellyn testaamista varten tehtiin JellyScriptillä JIRA:n tietokantaan muutama henkilö ja yksi projekti. Tähän projektiin lisättiin joitakin tehtäviä komentojen syntaksin testaamiseksi.

JellyScriptiä testatessa huomattiin, että vaikka JIRA:n dokumentaatio (Jelly Tags 2008) kirjoittaa mahdollisuudesta syöttää projektin tehtäviä CreateProject-elementin sisällä, on se turvallisinta tehdä kyseisen elementin ulkopuolella. Jos CreateIssue-komento ajetaan CreateProjectin sisällä, fixVersions-parametriin annettu tieto ei välttämättä tartu tehtävän tietoihin ja se voi jäädä tyhjäksi. Ellei versioinformaatiota tallenneta jokaiseen tehtävään, tehtävät jäävät Unassigned-tilaan. Tällöin niiden tekeminen oikean version yhteydessä muuttuu käytännössä mahdottomaksi. Kun elementti on suljettu, on projekti ja kaikki siihen liittyvä informaatio varmasti tallentunut. Kun tallentumisesta voidaan olla varma, voidaan tietoihin viitata CreateIssue-elementissä.

4.2 Sovelluksen spesifiointi ja ohjelmointi

Sovelluksen spesifiointi Projektissa käytettiin kahta XML-tiedostoa, jotka yhdistettiin yhdeksi projektimäärittelytiedostoksi. Kehysprojekti sisältäisi jokaiselle projektille kuuluvat yleiset tehtävät ja projektikohtainen tiedosto sisältäisi jokaiselle projektille yksilölliset tehtävät. Kehysprojekti on eräänlainen kehys maalaukselle, eli projektille. Maalaus olisi joka kerta erilainen, mutta kehys pysyisi suurimmalta osalta samana. Toki kehystäkin hienosäädettäisiin ja parannettaisiin ajan kuluessa, mutta sen yleismuoto pysyisi aina samana. Tarvittiin sovellus, joka yhdistää nämä kaksi tiedostoa ja muuntaa nämä yhdistetyt tiedot JIRA:n tukemaan JellyScript-muotoon (Kuvio 3).



Kuvio 3. Sovelluksen toiminta.

Jotta ohjelmisto olisi tulevaisuudessa helposti ja nopeasti muokattavissa, päätettiin sovellus toteuttaa XML-tekniikoita hyödyntävillä kielillä. Koska JellyScript on XML-kieli, päätettiin, että myös kehysprojektin ja projekteille yksilölliset tiedot syötetään XML-tiedostoon.

Sekä kehysprojektin, että projektikohtaisen XML-tiedoston tietorakenteiden määrittely kuului osaksi projektia. Rakenteessa pyrittiin mahdollisimman yksinkertaiseen tapaan syöttää tieto. Päätettiin, että syöttöön käytetään ainoastaan XML-elementtien parametreja, kuten voi nähdä koodiesimerkeistä 10 ja 11.

Kehysprojektin tiedostossa (Koodiesimerkki 10) määritellään kaikille projekteille yhteisiä tehtäviä. Käytännössä tämä tarkoittaa tehtäviä, jotka on toteutettava jokaiselle projektille, riippumatta siitä millainen projekti on kyseessä. Kehysprojektin tiedostossa myös määritellään kaikki projektin komponentit. Komponenteilla käsitetään projektin osaamisen eri alueet, kuten ohjelmointi, grafiikka jne.

```
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="
templateschema.xsd">
  <components>
    <component name="Code" description="Programming"/>
    <component name="Art" description="Graphics"/>
  </components>
  <versions>
    <version name="Alpha">
      <goal component="Code" name="Menu functionality"/>
      <goal component="Art" name="GUI Assets"/>
    </version>
  </versions>
</project>
```

Koodiesimerkki 10. Yksinkertainen kehysprojekti.

Projektikohtaisessa XML-tiedostossa (Koodiesimerkki 11) taas määritellään tehtäviä, jotka koskevat vain kyseistä projektia. Tiedostossa myös määritellään ketkä ovat komponenttien vastuuhenkilöitä.

```
<project codename="PROG" working-
title="Ohjelmistoprojekti" project-lead="projektijohtaja"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="projectschema.xsd">
  <components>
    <component name="Code" lead="koodari"/>
    <component name="Art" lead="graafikko"/>
  </components>
  <versions>
    <version name="Alpha">
      <goal component="Art" name="Draw a ninja"/>
      <goal component="Code" name="Code player movement"/>
    </version>
  </versions>
</project>
```

Koodiesimerkki 11. Projektikohtaisen XML-tiedoston esimerkki.

Kehys- ja projekti-kohtaiset tiedostot validoidaan niille asetettua XML-Skeemaa vasten. Mikäli validointi läpäistään hyväksytysti, ajetaan tiedostot yhdistävä XSLT-muunnostiedosto. Tämä XSLT-muunnos tuottaa yhden XML-tiedoston, joka sisältää molempien tiedostojen tiedot. Yhdistetty XML-tiedosto ajetaan vielä toisen XSLT-muunnoksen läpi, joka muuntaa sen JIRA:n käyttämään JellyScript-muotoon. JellyScript-muodossa olevan tiedoston sisältö kopioidaan JIRA:n ”Jelly Runner”-sivulle ja ajetaan. JIRA suorittaa JellyScriptissä annetut komennot ja luo tietokantaansa projektin ja tehtävät.

Sovelluksen toteutus

Ohjelmisto toteutettiin ajettavaksi Eclipse-ohjelmistoympäristössä, koska Eclipse sisältää tuen Apache Antille. Ant vuorostaan mahdollistaa XML-Skeeman tarkistamisen ja XSLT-muunnoksen teon ilman erillisiä sovelluksia. Projekti ajettiin suorittamalla Apache Ant -skripti, jolle annettiin parametrinä projekti-kohtaisen tiedoston nimi. Eclipse mahdollistaa Ant-skriptin suorittamisen suoraan ohjelmointiympäristöstä, eikä sen vuoksi tarvitse avata komentokehotetta. Kun projekti ajettiin, Eclipsen konsoliin tulostui sovellukseen ohjelmoidut ilmoitukset erilaisten prosessien onnistumisesta (Kuvio 4).

```
Buildfile: C:\Thesis\jira.createproject\build.xml
default:
[echo] *** VALIDATING XML FILES ***
[xmlvalidate] 1 file(s) have been successfully validated.
[xmlvalidate] 1 file(s) have been successfully validated.
[echo] *** VALIDATION COMPLETED ***
[echo] *** BEGINNING MERGE OF FILES ***
[xml] Processing C:\Thesis\jira.createproject\scripts\templateproject.xml to C:\Thesis\jira.createproject\output\merged.xml
[xml] Loading stylesheet C:\Thesis\jira.createproject\scripts\merge.xslt
[echo] *** MERGE COMPLETED SUCCESSFULLY ***
[echo] *** BEGINNING JELLYSCRIPT GENERATION ***
[xml] Processing C:\Thesis\jira.createproject\output\merged.xml to C:\Thesis\jira.createproject\output\unijelly.xml
[xml] Loading stylesheet C:\Thesis\jira.createproject\scripts\makejelly.xslt
[echo] *** FINISHED JELLYSCRIPT GENERATION SUCCESSFULLY ***
[echo] *** COPY EVERYTHING FROM OUTPUT\UNIJELLY.XML TO JIRA ***
BUILD SUCCESSFUL
Total time: 717 milliseconds
```

Kuvio 4. Sovelluksen onnistunut ajo.

Koska sekä kehysprojektin, että projektin oma XML-tiedosto oli jo validoitu XML-Skeemalla, ei yhdistetylle tai JellyScript-tiedoston validoinnille ollut tarvetta. Kun sisään menevä informaatio oli oikeassa muodossa, voitiin olla aina varmoja siitä, että ulos tuleva informaatio on myös oikeassa muodossa. Tässä vaiheessa kuitenkin huomattiin, että vaikka JIRA:n dokumentaatiossa ei ole erityistä mainintaa tietojen syöttöjärjestyksestä, sillä kuitenkin oli väliä. Jos CreateProject-elementin sisällä esiteltiin myös CreateIssue-komennot, ei näihin luotuihin tehtäviin tarttunut mukaan fixVersions-attribuutin tieto. Jotta uuden prosessin toteutumisen seuranta olisi mahdollista, tarvittiin fixVersionsin tieto tehtävien mukaan. CreateIssue-komennot oli siis turvallisinta ajaa vasta sen jälkeen, kun CreateProject on suljettu.

Sovelluksen testaus

Sovelluksen tulosteita testattiin JIRA:n testiasennuksella työasemallani. Testauksessa huomattiin, että virhetilanteessa JIRA ei lopeta koodin suorittamista. Käytännössä tämä tarkoittaa sitä, että mikäli CreateIssue-komennon syntaksissa on virhe, JIRA yrittää silti suorittaa kaikki myöhemmätkin (virheelliset) CreateIssue-komennot. Virheilmoitusten määrä paisui jättimäiseksi yksinkertaisestakin virheestä. Kymmenien

sivujen pituisesta rivinvaihdottomasta virhetulosteesta oli välillä vaikea löytää vika syntaksissa. Parhaaksi keinoksi ongelmien ratkaisuun huomattiin keino, jossa verrataan käsinkirjoitettua, toimivaksi testattua, CreateIssue-komentoa virheelliseen automaattisesti luotuun komentoon. Tällöin pystyi näkemään jo yhden kirjaimen eroja melko helposti.

Kuviosta 5 voi nähdä miltä onnistuneesti luotu projekti näyttää projektistauksessa. Kuviossa 6 näkyy yksittäinen tehtävä. Tehtävälisauksessa näytetään vain vähän tietoja tehtävästä, joten Summary-kohdasta on hyvä tehdä selkeä ja kuvaava.

Administration ?					
Below is the list of all 1 projects for this installation of JIRA.					
Add Project					
Name	Key	URL	Project Lead	Default Assignee	Operations
Ohjelmistoprojekti	PROG	No URL	Petri Päälikkö	Project Lead	View Edit Delete

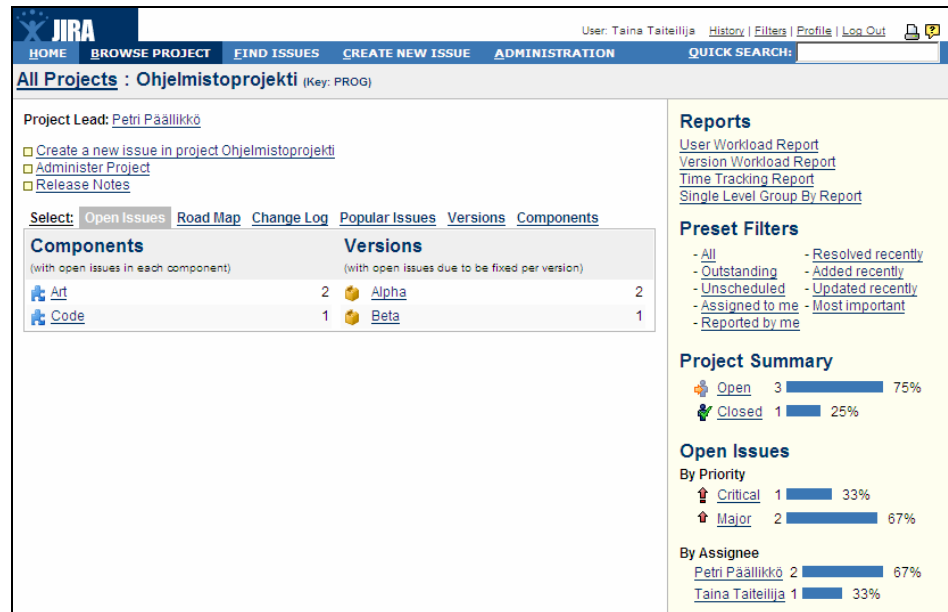
Kuvio 5. Onnistunut projektin luonti näkyy JIRA:n projektistauksessa.

T	Key	Summary	Components	Assignee	Pr	Status	Res	Updated	Due
	PROG-1	GUI Assets	Art	Taina Taiteilija		Open	UNRESOLVED	12.04.08	

Kuvio 6. Luotu tehtävä näkyy tehtävälisauksessa.

5 Arvio JIRA:n soveltuvuudesta prosessin hallintaan

Sovellusta rakennettaessa huomattiin, että tekniseltä kannalta JIRA:n käyttö projektien seurannassa on mahdollista. Projektiselaimesta (Kuvio 7) näkyy selkeästi montako tehtävää kullakin osa-alueella on ja paljonko niistä on tehty ja paljonko tekemättä.



Kuvio 7. Projektiselaimen näkymä projektista.

Kuviosta 7 voi nähdä miltä pieni esimerkkiprojekti näyttää projektiselaimessa. Yhteenvedosta (Project Summary) näkee miten paljon tehtävistä on tehty valmiiksi. Avoimista tehtävistä (Open Issues) näkee millaisilla prioriteeteilla tekemättömät tehtävät ovat ja kuinka moni tehtävistä on osoitettu kullekin vastuuhenkilölle. Vaikka projektiselaimen raportti on visuaalinen ja selkeä, siitä ei välttämättä saada tarpeeksi informaatiota. Tarkempien tietojen koostamiseen pitää kuitenkin rakentaa erillinen pieni lisäsovellus.

JIRA on yrityksen ohjelmoijille ennestään tuttu työkalu ongelmien seurantaan. Selkeänä etuna JIRA:n käyttöönotossa on myös huomattava rahansäästö. Ohjelmisto on jo yrityksellä, eikä rahaa uusien sovellusten ostamiseen tarvitse käyttää. JIRA on kuitenkin yrityksen muulle henkilöstölle vieras ja koulutusta jouduttaisiin varmasti järjestämään, mutta tämä olisi tarpeen vaikka yritykseen hankittaisiin erillinen sovellus projektinhallintaa varten.

6 Jatkokehitys

6.1 Mahdollisia muutoksia JIRA:an

Itse JIRA:an ei tarvitse tehdä muutoksia, jotta sitä voidaan hyödyntää uuden prosessin apuna projektien seurannassa. Mikäli kuitenkin halutaan, voidaan JIRA:an luoda useita erilaisia skeemoja. Näillä skeemamäärittelyillä voidaan muun muassa rajoittaa käyttäjien oikeuksia projektin eri osa-alueilla. Kaikille käyttäjille ei esimerkiksi ole tarpeellista, tai kannattavaa, sallia tehtävien poistoa.

6.2 Muutokset nykyiseen sovellukseen

Projektissa olisi voinut käyttää myös DTD-nimistä XML-kuvaustyyppiä. DTD toimii kuten XML-Skeema, mutta on huomattavasti yksinkertaisempi ja rajoittuneempi (XML DTDs Vs XML Schema 2002). DTD ei kuitenkaan mahdollistanut tarvittavan tiukkaa määrittelyä syötettäville XML-tiedostoille, joten sen käyttö ei käytännössä ollut mahdollista. Mikäli syötettävä tieto saisi olla vapaammassa muodossa, olisi DTD huomattavasti helpompi luoda ja ylläpitää, kuin XML-Skeema.

Koska projektissa haluttiin toteuttaa mahdollisimman yksinkertainen sovellus, sen käytettävyyteen ei panostettu. Mikäli JIRA:a kuitenkin aletaan käyttää projektinhallinnassa ja prosessin seurannassa, on sovelluksen käytettävyyteen mahdollista tehdä huomattavia parannuksia.

Sovelluksessa ei tällä hetkellä ole minkäänlaista graafista käyttöliittymää. Ellei sovellukseen haluta toteuttaa erillistä editoria, sen käyttöliittymä tarkoittaisi lähinnä ikkunaa, joka kysyy kahden syötetiedoston paikan ja tämän jälkeen käyttäjä saa valita, mihin valmis JellyScript-tiedosto tallennetaan. Vaihtoehtoisesti ohjelma voisi näyttää valmiin JellyScript-koodin ikkunassaan, josta se olisi helppo kopioida JIRA:n Jelly Runner:iin.

Mikäli sovellukseen haluttaisiin panostaa huomattavasti enemmän, voisi siihen tehdä oman tekstinkäsittelyosan. Tekstitiedostojen muokkaus ja XML:n validointi olisi tällöin mahdollista sovelluksen sisältä ja JellyScriptin tuottaminen ei vaatisi eri sovelluksen käynnistämistä. Tämä panostus tuskin olisi kovinkaan hyödyllistä, sillä valmiita sovelluksia XML-dokumenttien validointiin ja Scheman rakentamiseen on saatavilla lukuisia.

Tällä hetkellä kehysprojektin määrittelyä päivitetään yrityksen intranetissä. Jos nykyisen kaltaista sovellusta päädytään käyttämään, joudutaan kehysprojektin spesifikaatiota päivittämään sekä intranetissä,

että sovelluksen sisällä. Ratkaisuna tähän olisi käyttää toisenlaista sovellusta, joka osaisi hakea tiedon yrityksen intranetistä, eikä kehysprojektia tällöin tarvitsisi pitää ajan tasalla kahdessa paikassa.

6.3 SOAP:in käyttöönotto

Projektin toteuttamisen jälkeen kävi ilmi, että JIRA tukee tietojen syötössä myös protokollaa nimeltä SOAP (Simple Object Access Protocol). Tämä protokolla mahdollistaa XML-muotoisten viestien vaihdon tietokoneverkkojen ylitse. SOAP:ssa on mahdollista vaihtaa tietoa HTTP-protokollan yli. Tällöin palomuurien ja muiden tietoturvaratkaisujen ei pitäisi tuottaa hankaluuksia (McLaughlin 2001: 360-361). SOAPia käyttämällä olisi mahdollista toteuttaa sovellus, joka osaisi hakea kehysprojektin määrittelyt intranetistä. Tällöin sovellukselle tarvitsisi syöttää vain projektikohtaiset tiedot ja kehysprojektia tarvitsisi päivittää vain intranetissä. Näin säästyttäisiin siis huomattavalta ylimääräiseltä työltä. Javalla toteutettava SOAP-ohjelmisto olisi tuskin nykyistä sovellusta paljonkaan vaikeampi toteuttaa, vaikka itse protokollan toiminnan tutkimiseen voisi mennä aikaa.

6.4 Lisäsovellusten rakentaminen

Joskus JIRA:sta valmiiksi ulos saatavat raportit ja näkymät eivät ole prosessin ja projektin seurantaan sopivia. Tilanteessa, jossa täytyy antaa tietoa projektista ulkopuolisille, tarvitaan usein täysin erilainen raportti. Tällöin voidaan SOAP-protokollaa hyödyntämällä rakentaa erilaisia sovelluksia avuksi. Koska SOAP-protokollan avulla voidaan kysyä JIRA:lta kaikkia tietoja, joita normaalistikin selaamalla voi nähdä, voisi esimerkiksi intranetiin rakentaa serverillä ajettavan ohjelman (servlet), joka hakee projektin tilatiedot SOAP:lla ja tulostaa ne verkkosivulle. Tällöin käyttäjä ei pääse näkemään mitään ylimääräistä tai salaista tietoa.

Esimerkiksi projektin markkinoinnista vastaava henkilö ei tarvitse tietoa projektia toteuttavista henkilöistä ja sen näyttäminen voi jopa hidastaa oikean tiedon löytymistä. Erikseen rakennettu sovellus hakisi tarvittavat tiedot, kuten eri versioiden keskeneräisten tehtävien määrän. Näistä tiedoista voisi rakentaa esimerkiksi taulukon tai kuvaajan, käyttäjän tarpeiden mukaan.

7 Lopuksi

Projektissa onnistuttiin ratkaisemaan annettu tehtävä. Yritys sai toimivan sovelluksen, joka pystyi tekemään määritellyt tehtävät. Sovellus jäi kuitenkin hieman karuksi käyttöliittymän puutteen vuoksi. Tämän vuoksi sovellusta pystyy käyttämään ja muokkaamaan vain henkilö, joka ymmärtää siinä käytettyjä kieliä. Ohjelmistoon kirjoitettiin myös lyhyt dokumentaatio, joka opastaa kuinka ohjelma ajetaan.

Projektin aikataulu sisälsi ajan tarvittavien tietojen opiskeluun ja ohjelman toteuttamiseen. Varattu aika ei kuitenkaan riittänyt, vaan se venyi melkein kaksinkertaiseksi. Tämä johtui enimmäkseen epätarkoista ja osittain toimimattomista ohjeista dokumentaatioissa, erityisesti JellyScript-kielen kohdalla. Asioita jouduttiin kokeilemaan paljon, eivätkä JIRA:n epäselvät, kymmenien sivujen mittaiset, virhetulosteet auttaneet osaltaan.

Projektin aikana opin paljon käytettyjen kielten toiminnasta. Erityisesti Apache Ant ja JellyScript-kielten opituista taidoista on varmasti hyötyä tulevaisuudessa. Jos aloittaisin tekemään projektia nyt, tekisin osan asioista varmasti eri tavalla. Syötettävien tietojen rakennetta voisi miettiä hieman tarkemmin ja XSLT-muunnosta voisi varmasti optimoida lukuisilla tavoilla.

Karuudesta huolimatta sovelluksella voidaan selvittää JIRA:n soveltuvuus prosessin tarpeisiin. Mikäli huomataan, että JIRA on sopivin ohjelmisto prosessin hallintaan, voidaan opinnäyteprojektissa valmistunutta tiedon syöttöön käytettyä sovellusta helposti muokata. Pienellä panostuksella sovelluksesta saisi paremmin käytettävän ja sovelluksessa voitaisiin käyttää SOAP-protokollaa. Tällöin ohjelmistoa ei tarvitsisi ajaa Eclipse-ohjelmointiympäristöstä tai komentoriviltä. Yrityksen oma tutkimus sopivimmasta sovelluksesta ei ehtinyt valmistua tämän opinnäytetyön kirjoitustyön aikana.

LÄHTEET

- Extensible Markup Language 2006. W3C Organization. [online]
[viitattu 29.4.2008]
<http://www.w3.org/TR/2006/REC-xml-20060816/#sec-well-formed>
TAI: <http://www.w3.org/TR/2006/REC-xml-20060816/> → 2.1 Well-Formed XML Documents
- Harold, Elliotte R. 2000. XML: Tehokäyttäjän opas. Jyväskylä: Satku - Kauppakaari.
- Niemeyer, Glenn & Poteet, Jeremy 2003. Extreme Programming with Ant. Building and Deploying Java Applications with JSP, EJB, XSLT, XDoclet, and JUnit. USA: Sams Publishing.
- Apache Ant User Manual 2008. Apache Organization [online]
[viitattu 29.4.2008]
<http://ant.apache.org/manual/> → Introduction
- Apache Commons 2007. Jelly: Executable XML. Apache Organization [online]
[viitattu 29.4.2008]
<http://commons.apache.org/jelly/>
- Jelly Tags 2008. Atlassian. [online]
[viitattu 29.4.2008]
<http://www.atlassian.com/software/jira/docs/latest/jelly.html>
- XML DTDs Vs XML Schema 2002. Sitepoint. [online]
[viitattu 31.5.2008]
<http://www.sitepoint.com/article/xml-dtds-xml-schema>
- McLaughlin, Brett 2001. Java & XML: Tehokäyttäjän opas. Helsinki: Satku - Kauppakaari