Contents

1 Lab Exercise 01. Vector Images: Configurations, Objects and Templates

Course: "Web technologies", Spring 2007.

Due: Monday, March 5th, 2007

Purpose: The purpose of this exercise is to introduce Spring MVC application with a simple component architecture, Velocity templates and RDF data access. It transforms data from an RDF datasource into SVG or raster images via Velocity templates. This application could serve as a command-line or a Web application programmatically generating vector images from simple configuration data and HTTP requests.

Download: Use Anonymous checkout from BerliOS SVN (a Subversion client should be installed prior to this): https://developer.berlios.de/svn/?group_id=7959. See http://subversion.tigris.org/; it is sufficient to use Subclipse (an SVN plugin for Eclipse IDE).

1.1 Background

Vector and raster images each have their advantages; for comparison see Vector vs. Raster graphics. The software of this lab exercise will generate a range of simple images from their descriptions with RDF. (RDF is a general, non-presentational way to describe objects and their relationships. The data model of an RDF is a directed graph with labeled nodes and edges (these labels are URLs). The RDF can be written either as XML (this is used in all the W3C specification documents) or in the so called N3 notation, which is easier to create by hand, especially when compared to the XML syntax for RDF, which is the only format explained in most RDF specifications.

The processing chain by the software is the following:

1. Read RDF data e.g. with Jena library,
2. Initialize components. Component properties may depend on the defaults remembered at the application-wide context, some more specific contexts (e.g. user role, user specific settings, session settings, etc.), current model of image descriptions (all this information is recorded in `description.n3`). For the Web application the property values may depend also on the HTTP request parameters and actions (e.g. rotate, scale, hide, translate, etc.) performed on these components.
3. Transform the components to SVG vector graphics via Velocity templates
4. Optionally, filter the SVG to rasterize the graphics to PNG, which has better browser support.

   The Web application also contains a separate HTML page, which provides links to all the preconfigured images and allows setting their properties and performing actions from Web forms.

1.2 Preparing for the Exercise

Get familiar with running simple Java command-line applications from Eclipse, get an SVN snapshot from the BerliOS project KLUCIS and run it from Eclipse. Run JUnit tests and verify that all of them work properly. Download Spring framework, build and deploy on Tomcat and/or JBoss some of the Spring samle applications (e.g. "countries"). Make sure that you understand Spring's configuration file `countries-servlet.xml` and the `web.xml` configured for Spring's DispatcherServlet.

Install Adobe SVG Viewer 3.x plugin for Internet Explorer. Verify that you can view some SVG files (SVG samples are included with Batik distribution). Read some RDF specifications and familiarize yourselves with N3 syntax for RDF. More detailed instructions how to run the KLUCIS project are given in the Seminar 01.

1.3 Design Problem

1. Propose and implement "abstract" JUnit tests for the contract of `AbstractComponentFactory.localGetComponent(...` so that these tests are run for all subtypes of `ComponentFactory`, but they need not be repeated in all the respective test classes. I.e. implement an approach to test contract of abstract method and to define these tests in one location only (the principle is "Don't repeat yourself" or "DRY"). In particular you must check that after `localGetComponent()` returns the "id" property for the component is properly initialized. Read more about Abstract test cases.

2. Propose and implement JUnit tests to check that warnings are logged in case when there is a repeated property in `KlucisDAO.getCascadedProperty()`. Right now the JUnit tests are not sensitive against the messages being logged, even though the contract of a class includes its behavior regarding the logger. Your tests should check that the log warning text matches certain *regular expression*. A regular expression is a way to describe strings satisfying some pattern (see e.g. `java.util.regex.Pattern`). In particular, check that if the same property is repeated for the same resource, then the following warning message is logged:

   ```
   "Non-unique property '[^']+' for resource '[^']+'"
   ```

   where we do NOT care about the name of the property or the resource. (Even though the testcase may know the property and the resource, asserting them in the log message could make the testcase brittle. In fact, the property may be output either as a full URL, or a QName - e.g. "KLUCIS:hasColor", "klucis:hasColor" and "http://www.webkursi.lv/schema/20061008/klucis#hasColor" - may all be valid ways to write the property. Similarly the resource may be a URL or a QName, or a blank node.

3. Implement a separate controller, which for all requests matching certain URL pattern outputs the PNG image obtained from the SVG through rasterization/transcoding. For example, the pattern could be `/klucis/main/PngImage/*`, so that, for example "bilde_15" can be requested as http://localhost:8080/klucisDemo/kluci
   You will probably need Apache Batik library to do this. Classes `SaveAsJPEG` and `SaveAsPNG` show sample code to do offline transformations for SVG, but your approach (e.g. filter) should do this server-side. Each transcoding/rasterization to PNG typically takes about 1-2 seconds to complete. Do not worry about the performance. The PNG filter could be configured similarly to the XSLTFilter:

   ```
   <filter-mapping>
     <filter-name>PngFilter</filter-name>
     <url-pattern>/klucis/main/PngImage/*</url-pattern>
   </filter-mapping>
   ```

4. You will notice that the factory code in the package `lv.webkursi.klucis.component.geom2d` is very predictable. Propose a method to generate this code from annotations (e.g. `RectangleFactory` code should be generated from annotations for `Rectangle` properties). You may use either Java 1.5 annotations or XDoclet approach or anything else to do that. You can add one extra precompilation step to the Ant buildfile, which would perform the code generation before everything is compiled and before the JAR is archived.

5. Currently there is a possibility to do recursive render of Velocity views (i.e. call merge of one Velocity view inside another). The approach works both for command-line and for Web application. There is one drawback though - sometimes components have many properties, and all these are added individually. For example in `Rectangle.lifecycleEvent()` there are many quite predictable lines:

```
addObject("_offsetX", offsetX);
addObject("_offsetY", offsetY);
addObject("_showRectangle", showRectangle);
addObject("_width", getCoreWidth());
addObject("_height", getCoreHeight());
if (!label.equals("")) {
        addObject("_label", label);
}
addObject("_content", content);
addObject("_rotate", -rotate);
```

This enables the Velocity template to refer to all these variables as `$_width` etc. The alternative, which you should implement, is to add just the reference to the underlying component (e.g. for the key "_component"), and all the properties could be just extracted by "$_component.getXxx()" methods. So, in `Rectangle.lifecycleEvent()` we would write just one `addObject()` command, e.g.:

```
addObject("_component", this);
```

And in the Velocity template we could write something like `$_component.getWidth()` instead of `$_width`.

6. Implement a few more components to enable drawing the flags (similar to the ones `17a.gif`, ... `17e.gif`). These images should be displayed in response to the following URLs - http://localhost:8080/klucisDemo/01/main/l etc.

7. Currently, if several versions of the same shape is being output, it is repeated. Implement an optimized version, which would output each component once and use SVG refid mechanism, if necessary. (Compare house1.svg and house2.svg from Seminar 01 Demo).

1.4 Resources

Do not worry, if there are too many frameworks and libraries used by this project. Most of the libraries are used in a simple and straightforward way, and the usage samples are provided.

1.4.1 What to Install

J2SDK, Ant (or optionally Maven), Eclipse (possibly with Subclipse, Velocity and XMLBuddy plugins), Editplus/Scite plain text editors, Firefox or Adobe SVG plugin for MS Internet Explorer. See Installation instructions.

1.4.2 Spring

Spring framework is used to set up a Web application, which uses Spring's pre-built MVC (Model-View-Controller) solution along with Spring's other Dependency Injection mechanisms. A step-by-step tutorial is available - see [Spring_Ris]. It is recommended that you configure Spring starting with some minimal configuration and then gradually add more things to it as they become necessary rather than start from some huge application, which contains all kinds of things, which do not make immediate sense.

1.4.3 CVSDude

To collaborate more effectively, your team might want to set up version control system for your team. One possibility is CVSDude - it has limited space, so it is better NOT to check in JAR files, but only your own code.

1.4.4 Provided Code

A prototype Spring MVC application is already in place - it has fully configured `web.xml` and Spring's configuration files for each of the two dispatcher servlets. It already contains most of the code for the less obvious Java libraries doing the Velocity configuration, RDF data access, and SVG rasterization.

1.5 Mechanics

There are 2 Spring configuration files. One Spring config file is for the command-line version of the program `trunk/bat/context.xml`. Another is for the Web version of the program - `trunk/klucis_demo/src/main/webapp/WE`
Both files contain configurations of the beans participating in the mini framework.

1.6 Expectations

1.6.1 Deliverables

- Your application (including all source code and tests), which passes all the provided JUnit and Selenium tests.
- Provide brief description of your aproach (some explanation regarding the analysis and design) for Factory class generation (Design Problem #4), the picture 17_N generation (Design Problem #6) and using the SVG refid (Design Problem #7). This presentation should contain the full names and e-mails of all your team members (i.e. 1..3 people) on its first slide. Write this presentation to a MS PowerPoint-compatible file `presentation.ppt` and place this file under `klucisTrunk/docs`.
- Before submitting your project, please run "ant submit" to create the file `lab01.zip`, which you can send as attachment to the instructor's e-mail, namely **kalvis.apsitis** at the domain **gmail.com**. The Ant target "submit" was missing in the original KLUCIS distribution. Please see "Errata" section below for how to resolve the situation.

1.6.2 Prerequisite knowledge

Familiarity with SVG vector graphics, Spring configurations and RDF in N3 notation.

1.6.3 What is learned during this exercise

How to set up Java/Eclipse/Maven/Subversion environment, how to develop simple Spring MVC applications, how to use Velocity templates to create simple markup (HTML and SVG), how to access RDF data from Java code and how to generate bitmap images on the fly.

1.6.4 Guidelines for Evaluation

- 20 grade points total - 2 for each of the design/implementation problems (i.e. 14 for all 7 design/implementation problems). Your team can also get up to 6 grade points for the write-up of the presentation `presentation.apt`.

1.7 Bibliography

[W3C_SVG_03] World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C, Ed. Jon Ferraiolo et al., 2003-01-14. http://www.w3.org/TR/SVG11/.

[Eis_SVG_02] Eisenberg, J.David. *SVG Essentials*. O'Reilly, 2002.

[Ris_Spring_03] Risberg, Thomas. *Developing a Spring Framework MVC application step-by-step*. 2003-06-30. http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-by-step.html.

[Ber_N3_05] Berners-Lee, Tim. *Primer: Getting into RDF & Semantic Web using N3*. W3C, 2005-08-16. http://www.w3.org/2000/10/swap/Primer.html.

[Velocity] The Apache Software Foundation. *Velocity User Guide*. Jakarta.Apache.Org, 2005. http://jakarta.apache.org/velocity/doc guide.html.

[Jena] Hewlett-Packard Development Company, LP. *Semantic Web Framework Jena*. Jena.Sourceforge.Net, 2006-06-15. http://jena.sourceforge.net/documentation.html.

[Batik] The Apache Software Foundation. *Batik SVG Toolkit*. Xmlgraphics.Apache.Org, 2005. http://xmlgraphics.apache.org/batik/

[APT] Shafie, Hussein et al. *APT User Guide*. Www.Pixware.Fr, 2005-06-13. http://www.xmlmind.com/_aptconvert/docs/userguide

1.8 Errata

The missing "ant submit" target: The buildfile in the project root directory - `klucisTrunk/build.xml` now contains the "submit" target. Please make sure that your submission of Lab01 is produced by something like this (please read the subsection "Mechanics" above). The Ant code looks like this:

```
<project name="klucisTrunk" basedir="." default="clean">
 ...
 <target name="submit" description="Create a ZIP file ready for submission">
  <ant antfile="build.xml" dir="klucis_core" target="clean"/>
  <ant antfile="build.xml" dir="klucis_demo" target="clean"/>
  <delete dir="target/submit" failonerror="false"/>
  <mkdir dir="target/submit"/>
  <copy todir="target/submit">
    <fileset dir="." includes="**/*"
        excludes="target/**,bin/**,.*,**/*.jar,**/*.zip,**/*.log">
    </fileset>
  </copy>
  <zip destfile="target/lab01.zip">
    <fileset dir="target/submit"/>
  </zip>
 </target>
...
</project>
```

i.e. it executes "clean" target in both subprojects (klucis_core and klucis_demo), it creates an empty staging directory "target/submit"; it copies everything under klucisCore to this directory, except for dependent files, any JARs, ZIPs and logfiles. Then it creates a new zip file `lab01.zip` in the `target` directory. To execute this task, either copy-paste the above target to your `klucisTrunk/build.xml` or perform the Subversion update on this `build.xml` file from the BerliOS repository.

This `lab01.zip` can be submitted by any member of your team. Please do not forget to tell me, what are ALL members from your team (otherwise I won't know who should get the grade!).

2 Lab Exercise 02. Quiz App: MVC Pattern and Data Access

Course: "Web technologies", Spring 2007.

Due: Monday, April 16th, 2007

Code Freeze: The Lab02 description and its sample Java code in BerliOS won't change after Monday 26th, 23:59 (i.e. at that point all the requirements become finalized). Before this date you can suggest

all kinds of improvements; after that date only the errors will be changed (and documented under "Errata").

Purpose: The purpose of this exercise is to learn more about Spring and Hibernate to make data access to relational databases.

2.1 Special Thanks

A few Master program students have contributed to this exercise. Thanks to Anete Ozola for providing her application Mtest.lv written in RubyRails. Vladimirs Potapovs suggested the Spring's solution for an internationalization filter. (Prior to that a more complicated custom solution was used to support UTF-8 encoding for Web forms.) Here is the solution (see also the file `edu_demo/webapp/WEB-INF/web.xml`):

```
...
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

2.2 Background

A simple multiple-choice testing application illustrates how a domain model can be mapped to a relational database, and how a Spring MVC application may be used to be the Web layer for that database application. The domain model (the design of lv.webkursi.mtest.lab02.domain) is available as an image in PNG and SVG formats.

2.3 Preparing for the Exercise

In this exercise you will create a Java analogue to an existing application done in Ruby/Rails. Here are instructions how to run the RubyRails prototype application Mtest.lv:

```
RubyRails instal\={a}cijas instrukciju sk. www.rubyonrails.com.
Aptuvena solju seciiba:

(1) Instaleet Ruby (piemeeram ruby186-25.exe no
http://rubyforge.org/frs/?group_id=167 ).

(2) Ja atrodaties aiz PROXY servera,
tad DOS lodzinjaa uzstaadiet pagaidu mainiigo:
set HTTP_PROXY=http://your.proxyserver.url:8080
Preteejaa gadiijumaa izlaidiet sho instrukciju.

(3) Izpildiet DOS komandu:
gem install rails --include-dependencies

P\={e}c Ruby un Rails instal\={e}\v{s}anas
var sagatavot un izpild\={\i}t mtest.lv aplik\={a}ciju:

(4) Palaist BAT failu initialize-db-and-run-webrick.bat.
\v{S}is .bat faili\c{n}\v{s}:
-izveido localhost datub\={a}zi 'mtest' (mysql login/parole root/root)
-izveido lietot\={a}ju Admin (mtest login/PAROLE admin/admin)
-ieraksta datub\={a}z\={e} p\={a}ris testa modu\c{l}us
-start\={e} Webrick (http://localhost:3000)
```

2.4 Design Problem

1. Consider the RubyRails application as a sample and add functionality to create new modules ("Testa moduļi"), add questions ("Jautājumi") to the modules, create assignments ("Testa sagataves") and take tests ("Veikt testus"). (The database schema is somewhat different for the Ruby application, e.g. it allows some enum fields - "status" and "visibility" for test modules, etc. You may assume for the sake of simplicity that everything is public and visible, i.e. you do not need to implement these enum fields, unless you want to.).

2. Create a utility program (a Java console application with its main() method), which can import questions annotated as XML (see `csdd_fragmenti.xml`) into the MySQL database "portaledu".

3. Create functionality to upload images for the tests. Each image belongs to a certain testmodule and has a unique name within that module. Image size does not exceed 64K, so you can use the regular MySQL BLOBs to store the image data. Currently Image objects are partly mapped to the database using Hibernate (see ContentItem.hbm.xml mapping), but the binary BLOB is not mapped. You may choose to implement the access to BLOB via normal JDBC. There is a command-line utility lv.webkursi.mtest.lab02.dao.BlobDemo, which inserts into a database table some JPEG images. You may use this as a code sample (notice that it is easy to get a JDBC connection from a Hibernate session object).

4. Whenever the application displays a list of something (e.g. "/person/listall", "/module/listall", etc.), implement a paging functionality - results should be displayed in pages - 20 results per page.

5. Implement reasonable security restrictions - links/buttons like "Edit" and "Delete" should not allow the non-admin users to update data in table Person and QuestionType. Each user sees only his/her own testmodules, but can take assignments made by anyone else.

6. Have a correct and efficient process of obtaining/releasing database connections. Namely, use connection pool instead of "drivermanager" (see commented section in file `mtest-servlet.xml`). Also, call the "close()" method on various instances of `CommonDao`, when the beans, which need them become garbage or are unloaded by the Web container. (This way your application will not crash or run out of memory as often as it does now.)

2.5 Resources

2.5.1 Provided Code

Check out the code from the following BerliOS repository (e.g. with Subclipse): https://kalvis@svn.berlios.de/svnroot/repos/klucis/p
It should not ask for any passwords, if you access that repository in read-only mode.

2.6 Mechanics

You will mostly edit files under the packages `lv.webkursi.mtest.lab02.*` under the "edu_core", and also all kinds of files under the "edu_demo". There should be very few dependencies from "lv.webkursi.mtest.lab02.*" packages to any "lv.webkursi.mtest.core" (the latter deals with RDF data processing and is not needed for this exercise).

- The JUnit tests can be run by executing all test methods under `portalEdu\edu_core\src\test\java`.
- The Selenium tests for lab02 can be executed from the URL http://localhost:9080/eduDemo/tests/. Select the link "Run MTest tests".
- In case you need to modify something under the "lv.webkursi.mtest.core.*" packages, also execute the other Selenium test suite - pick the link "Click here to run PortalEdu tests" in the above Web address.

2.7 Expectations

In this exercise a special attention will be paid to the completeness to your test suites. Make sure that all the DAO methods you need are tested (i.e. you would need tests for the new DAO methods regarding Assignments, Sessions and Answers, which persist the domain objects from the package lv.webkursi.mtest.lab02.domain), and also JUnit tests for various other objects, e.g. validators.

Your Selenium tests would need to cover all kinds of normal behaviors and also abnormal ones (invalid form submissions, someone attempting to do something without proper authorization, e.g. a non-admin user doing admin functionality, etc.)

2.7.1 Deliverables

As before, you are requested to write a presentation lab02.ppt, which you can include under "portalEduTrunk/doc" directory. This presentation should contain short description of all your main activities and design solutions taken for this exercise and also the full names of all participants in your team. **Having a PPT presentation is mandatory**; submissions, which do not contain it will be rejected.

When you are finished doing all the things described under "Design Problem", please run the Ant task "ant submit" in the root directory of "portalEduTrunk". It should create a file `lab02.not_a_zip`, which one of your team can send as a regular e-mail attachment to the "kalvis.apsitis" account at "gmail.com". (The file actually IS a ZIP file, but it has a different extension to bypass checking for executable BAT files, which may be contained in this attachment.)

2.7.2 Prerequisite knowledge

It is recommended that you understand Hibernate and Spring MVC before attempting this exercise.

2.7.3 What is learned during this exercise

Design and configuration for a typical Spring/Hibernate database application; also its development steps and testing.

2.7.4 Guidelines for evaluation

Each of the 6 exercises in "Design problems" is worth 3 points. Points may be reduced, if your code is not properly object-oriented (e.g. if you write the command-line utility as a huge main() method), also, if the code is not well readable and does not properly handle resources (like database connections, uploaded image files, etc.), is potentially insecure or iniefficient. Also, if the JUnit/Selenium tests are incomplete. The remaining 2 points will be given for the PPT presentation.

2.8 Bibliography

- Good resources to learn Hibernate and Spring is their standard documentation (one can unpack the distributions of `spring-framework-2.0.2-with-dependencies.zip` and `hibernate-3.2.2.ga.zip` to a local directory.
- For advanced Spring features the textbook `Apress - Expert Spring Mvc And Web Flow - Feb 2006` is recommended.

2.9 Errata

2.9.1 Remarks regarding Unix and similar environments

Mārtiņš Barinskis atzīmē sekojošas izmaiņas, ja darbina Mtest.lv (t.i. Ruby/Rails aplikāciju) uz Unix-veida platformas.

- Lai palaistu ROR aplikāciju, MTest direktorijā ir jāraksta sekojoša komandrinda: "ruby script/server". Tad tiek izmesta kļūda "Permission denied", kas bija saistīta ar FastCGI moduli. Problēma atrisinājās ar sekojošu komandrindu:

  ```
  chmod 755 public/dispatch.fcgi
  ```

  acīmredzot, kaut kas bija noticis ar šī faila piekļuves tiesībām.

- Vajadzēja mazliet pamainīt arī `simple_import.rb` scriptu, lai tas izpildītos. Sākotnēji Ruby žēlojās par to, ka neatrod active_record moduli. Lai to laistu ārpus ROR ietvara, vajag iekļaut papildus moduli - 'rubygems'. Man viss aizgāja ar sekojošu `simple_import.rb` skripta sākuma daļu:

  ```
  #!/usr/local/bin/ruby
  require 'rubygems'
  require 'active_record'
  require 'rexml/document'
  require 'logger'
  ```