Function:Troilkatt/MySpell

From FunctionWiki

(Redirected from Function:Troilkatt/Private Data)

This document is used for the design description of Spell extended with **private data search**.

To go up.

Contents

Version History

- Before content was merged to main design document: [1] (http://incendio.princeton.edu/functionwiki/index.php?title=Function:Troilkatt/Private_Data&oldid=2662)

Detailed Description

Operational Scenario

We assume the user has converted her microarray data to either the SOFT format used by NCBI GEO or the PCL format used by SMDB.

1. The user goes to the private search web page and clicks on the button that downloads and starts the MySpell application. MySpell will also connect to the SpellMaster running on our servers.
2. Specify the location of the raw data and a location where the processed data is stored. This will do the same processing on the raw data as is done for our public data.
3. MySpell displays the usual Spell search interface, although with a different color scheme.

To do the search the user enters the query genes and the datasets to search for in the usual way. The search will then be executed as follows:

1. The query is sent to MySpell, where the private datasets are removed from the query, which is then forwarded to SpellMaster.
2. MySpell combines the results for the private datasets it calculates, and the results for the public datasets received from SpellMaster.
3. The MySpell client combines the visualizations for the private data received from MySpell, with the public data received from the public web-server.

User Interface

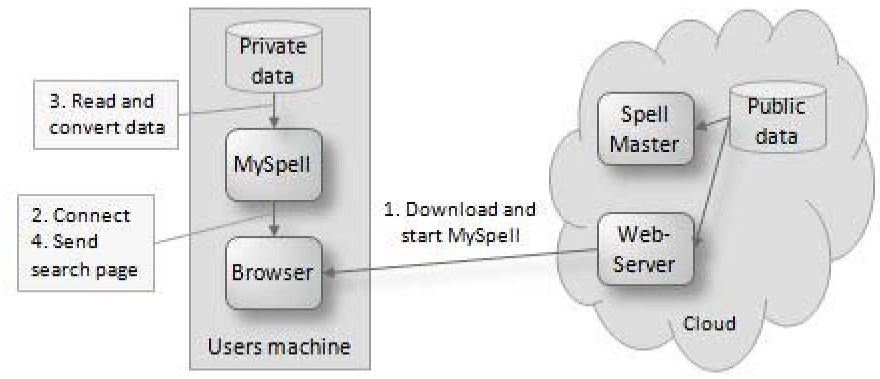
The Spell search interface is extended with the functionality to include private datasets. The result display is changed to show special information for the private datasets.

Architecture and Protocol

The system consist, in addition to the user, of four main components:

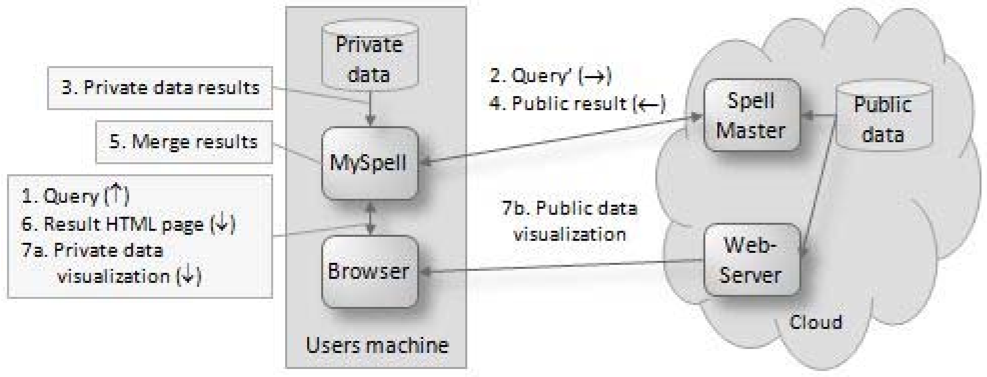
1. The Web-server running on one of our servers.
2. SpellMaster and the SearchWorker's running on our cluster.
3. The MySpell server running on the machine where the user have the data.

4. The user's web-browser.



The handshake protocol for a search that includes private data is as follows (we assume the user has already pre-processed the data):

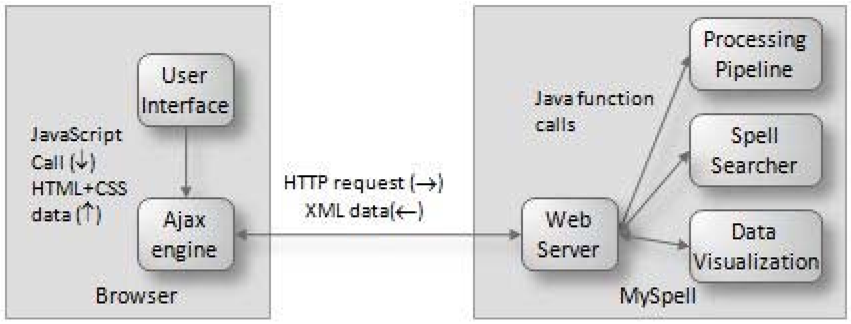
1. The user goes to the private data serarch web-page and clicks on a button that downloads and executes the MySpell Java application.
2. The web-browser automatically connects to the MySpell server.
3. The user specifies the private datasets to be included in the search using the MySpell application user-inter. MySpell converts the data to the same format as used in our public datasets, and saves the converted datafiles locally.
4. The MySpell server provides backend services with the same interface as SpellWeb2. These are used by the MySpell browser GUI which is an extension of the SpellWeb2 GUI.



After registering the datasets, the user enters the query gene names and the datasets to search. The private data search protocol is then executed:

1. The search query is sent to MySpell (instead of the public web server).
2. MySpell parses the query, identifies the private datasets to search for, and forwards the query with the private dataset IDs removed to SpellMaster.
3. MySpell does the search on the private data.
4. SpellMaster does the search on the public data, and sends the raw results to MySpell (this includes the per dataset scores and per gene scores).
5. MySpell combines it's private data results with the results received from SpellMaster, and sends these to the client code running in the web browser.
6. The client code presents the results to the user in the same way as for results returned from the public web server.

Execution Environment



MySpell and the user interface running in the browser are both implemented in Java, and they use remote procedure calls (RPC) for communication. The web-interface is based on the Google Web Toolkit RPC (GWT [2] (http://code.google.com /webtoolkit/) and [3] (http://code.google.com/webtoolkit/doc/latest/DevGuideServerCommunication.html)). The client is implemented in Java using the GWT libraries. This code is the compiled to JavaScript, where Ajax asynchronous RPC's are used to communicate with MySpell. The server is implemented using servlets, which are run in the Jetty container [4] (http://jetty.codehaus.org/jetty/) (for details about the Jetty setup refer to the Jetty Setup section).

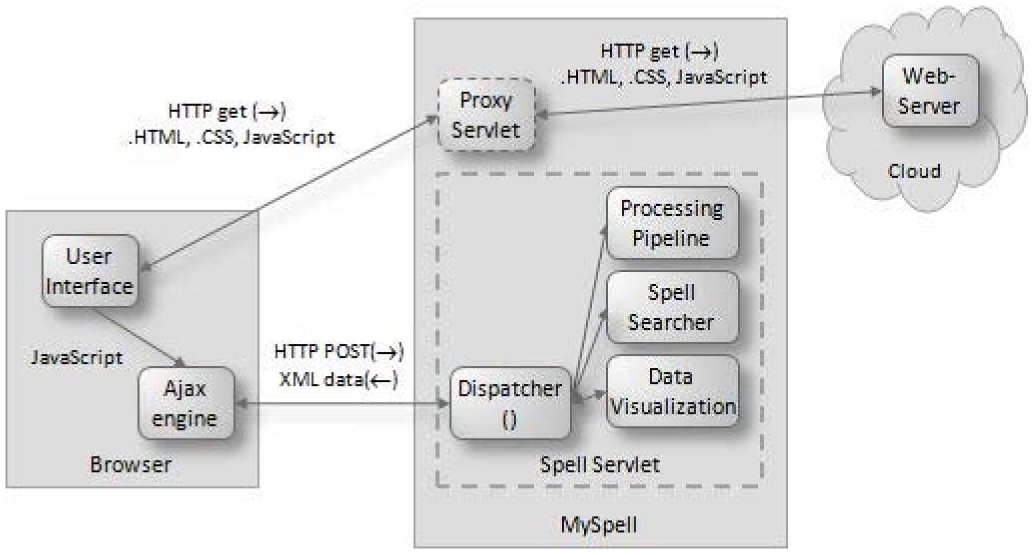
The MySpell GUI is in the browser is implemented as a web-application. The browser clientd runs most of the business logic, which in our case is the MySpell program. The GUI is implemented using Google Web Toolkit which compiles Java code to JavaScript code.

GWT was chosen since it is open source, well documented, shown to be portable across many platforms, and since it uses the Java programming language that the rest of the code is written in. We use Jetty since it is a Web-server implementation that is already used on small devices such as Android mobile phones.

An alternative approach is to implement the GUI in the MySpell Java application. However, we choose the web-application approach since we can reuse the GUI code for public search.

Jetty

The GWT-RPC interface used by the browser to communicate with MySpell is implemented as a Java servlet. The servlet is executed in a Jetty container implemented by embedding Jetty into the MySpell code. The alternative is to use a servlet container as Tomcat, but it will increase the code size and complicate the startup (described in the next section). However, the code can still be compiled as a web-application and deployed in another container if needed.



The Jetty container in MySpell has two servlets: SpellSearchServiceImpl and TransparentProxy. The Spell servlet handles all of the GWT-RPC requests, while the proxy forwards all requests for static webpages (.html, .css, and JavaScript files). The container for a request is chosen based on the URL used. All myspell/searchserver/* requests are sent to the Spell servlet and the remaining /* requests are handled by the proxy.

The reason for using a proxy servlet instead of letting MySpell serve the static pages is to keep all the static pages at the public web server (separation of functionality). This allows using for example a single .css file that can be used to make global changes of the web page layout.

Startup approach

To run MySpell on the user's machine we would ideally like a have a browser based solution that does not require downloading and installing any additional code. However, JavaScript and Java applet code run the browser are in a sandbox with restricted access to for example the files system. It would therefore be tricky to use the local file system or has access to sockets. So instead we chose to run MySpell as a separate Java application with full access to the resources on the machine. (i.e. not a sandboxed applet). This requires an approach for starting the Java application in a simple way. For this we use Java Web Start.

Using Java Web Start, MySpell can (ideally) be started by going to a webpage and clicking on a "launch MySpell" button. The browser will then download a JNLP file and run it using the javaws program (javaws is part of the standard Java JVM distribution and most browsers have jnlp files associated with this program). Java Webstart (javaws) will then:

1. Download the MySpell.jar file from the web-server. This is an executable jar file that contains the MySpell code and all necessary libraries (external jar files).
2. Verify that MySpell.jar is signed with a certificate and that the code is untampered. This is necessary in order to run MySpell outside a sandbox. **Note** that we do not have a certificate verified by a "trusted source" so the user will get a warning dialog.
3. Execute MySpell as a java application with full access to the resources on the machine.

To work with Java Webstart MySpell implements a simple GUI with a status bar and an exit button, and a function that downloads the .rpc files necessary to communicate with the client code running in the browser (the files are saved in $cwd/myspell/). The GUI is necessary to show the user that MySpell is running and to provide a way to kill the program. The .rpc file is used by GWT-RPC and it must be on the local file system.

Java Webstart is not an ideal solution and it has several issues:

- If the user does not have Java installed (or the browser correctly set up) a confusing Java download page is displayed.
- Starting MySpell may show several warning dialog boxes in the browser.

An alternative is therefore to keep all files in memory and expand the SpellWeb2 RPC interface to provide MySpell search capability. This would allow compiling the MySpell code to JavaScript.

Security Assumptions

We make a number of assumptions with regards to the security and privacy of the system.

Our main assumption is that the privacy requirements are meet by never transmitting any private data to the cloud. Further we assume that:

- The MySpell server only accepts local connections, and that malicious users cannot make such connections.
- The communication between the web-browser and MySpell is secure.
- The data used by MySpell is stored securely.

Performance Considerations

Private data search requires transferring more data over the network since the weights and scores for all datasets and genes are needed for the final reduce which is done on MySpell. This increase in data volume compared to just sending the weights and scores for the top ranking genes is likely to impact the search time.

Another potential factor search time increase is the data processing on the local host. However, we do not believe this will introduce a signigicant performance decrease since we assume the number of private datasets is relatively low (tens of datasets) and hence the computation time will also be low.

Failure Considerations

Failures can occur if the user-side code crashes, or the SpellMaster or Web-Server crashes (or there is a network outage).

Since there is no state saved in the cloud about private data no additional steps are required during recovery. However, in either case it is necessary to:

1. Re-connect to SpellMaster.
2. Re-execute the query.

Processing Pipeline Extension

The Spell processing pipeline scripts needs to be implemented in Java, such that they can be executed with minimal installation effort.

SpellWeb2 extensions

SpellWeb2 is extended with:

- A page with a one-click solution to download and start MySpell.
- A Java RMI interface used by MySpell which is implemented in Java and therefore cannot use the GWT-RPC interface.

SpellMaster Extension

The SpellMaster is extended with a new search interface where:

- A MySpell client can receive the full results for a search (all gene scores, all dataset weights, and all per-gene dataset weights).

MySpell

The MySpell server consists of three parts:

1. The data processing pipeline implemented in Java.
2. The SearchWorker code that allows doing a Spell search in the private datasets.
3. The SpellMaster code that allows merging results returned from SearchWorker's (that is the private and public data).
4. A modified SpellWeb2 server code to merge visualizations for private and public datasets.
5. A modified search page that allows specifying private datasets to include in a search.

Programming Interface

Startup Protocol

During startup MySpell executes the following protocol:

1. Connect to a web-server running at a predefined host.
2. Download a configuration file, filelist.txt, that contains the resource files to download.
3. Download each file listed in filelist.txt. Presently there is only one file which is the serialization metadata file (<md5 sum>.rpc) that is necessary to send objects between MySpell and the client code running in the browser.
4. Open the MySpell.html page in the browser window that executed JavaScript code that connects to MySpell. The HTML, image, and JavaScript files are all downloaded from the public web-server.

MySpell - SpellMaster protocol

MySpell forwards public data search requests to SpellMaster using the *MySpell search* message in the DistrubtedSpell protocol.

MySpell - SpellWeb2 RMI protocol

See also: SpellWeb2 Java RMI Application Interface (TODO: link to Javadoc)

MySpell receives gene metadata, dataset metadata, and expression values from the SpellWeb2 web server using a Java RMI interface exported by SpellWeb2. This interface is identical to the GWT-RPC interface used by SpellWeb2 browser client code.

Browser - MySpell RPC interface

See also: MySpell RPC Application Prorgamming Interface (TODO: link to Javadoc)

The client code running in the browser communicated with MySpell using the GWT-RPC protocol. MySpell exports the SpellWeb2 API extended with functions to add, remove, and receive information about private datasets.

Browser GUI

The browser GUI extends the SpellWeb2 GUI with a dialog box that allows to select which private datasets to include in a search. In addition the download MySpell page is removed and a different CSS file is used.

MySpell GUI

MySpell implements a setup GUI that allows the user to setup the processing pipeline and to add private datasets to be included in the search. The GUI is designed using the Jigloo plugin for Eclipse. The GUI automatically initializes the organisms list by requesting the list from the public web server, and sets the output directory to the platform specific temp directory. It also opens the add datafiles dialog box when started. Once the user has selected the private datasets all further user interaction can be done using the browser GUI, including the (TODO: addition and) removal of private datasets. The browser GUI and MySpell GUI are synchronizes such that changes, such as the removal of private dataset, in one GUI are reflected in the other.

The MySpell GUI implements the following functions:

- Select the organism used in the private dataset experiments.
- Add private dataset(s).
- Remove private dataset(s).
- Open the browser GUI.
- Select the processing steps to be done on added private datasets.
- Select the output directory.
- Select output file properties.

Code Overview

MySpell is split into five projects/directories:

1. DistributedSpell: contains inherited distributed Spell code.
2. SpellWeb2: contains inherited web server and GUI code.
3. MySpell: contains the code for the MySpell application and the browser GUI.
4. MySpellJettyServer: contains the code for launching MySpell as an embedded Jetty container
5. MySpellJNLPServer: contains code for launching MySpell using JNLP. The reason for splitting MySpell into two parts is to simplify deployment as a jar file.

The MySpell (GWT) project has the following packages and files:

- edu.princeton.function.myspell.client: user interface
  - MySpell.java: entry point and GUI setup code. This file contains code copied from SpellWeb2.java since it provided difficult to subclass that file.
  - MySpellException.java: exception throw for MySpell specific exceptions that are sent to the GUI code.
  - MySpellService.java and MySpellServiceAsync.java: RPC specification (subclass of the SpellWeb2Service files).
  - PrivateDatset.java: private dataset metadata.
  - PublicSearchView.java: search interface (spellweb.SearchView extended with private dataset selection).
  - SelectDatasetDialog.java: dialog box for selecting private datasets to include in a search.
- edu.princeton.function.myspell.client.MySpell.gwt.xml: MySpell GWT configuration file.
- edu.princeton.function.myspell.pipeline: data processing pipeline.
  - OsPath.java: utility functions for directory and file management. It implements functions provided by the Python os.path module.
  - Pipeline.java: data processing pipeline.
  - PipelineException.java: data processing exceptions.
- edu.princeton.function.myspell.search: private data search and data merge code.
  - MySpellSchedulerThread.java: thread that handles a search for private and public datasets, and merges the results.
  - MySpellWorker.java: SearchWorker module that creates and runs a MySpellSchedulerThread (this class is likely to be removed since it is unnecessary and confusing)
- edu.princeton.function.myspell.server: webserver code.
  - MySpellBackend.java: private dataset management and code to forward requests to the public SpellWeb2 server.
  - MySpellServiceImpl.java: wrapper class that exports a GWT-RPC interface and calls the functions implemented by MySpellBackend.
- war/MySpell.css: MySpell style defintions.
- war/MySpell.html: HTML file that contains the GUI Javascript code.
- war/WEB-INF/web.xml: configuration file.

The MySpellJettyServer has the following packages and files:

- com.cloudgarden.layout: files produced by Jigloo.
- edu.princeton.function.myspell.jetty:
  - MySpellJettyServer.java: implementation of an integrated Jetty server for MySpell that can be run as a standard Java application.
- edu.princeton.function.myspell.pipeline:
  - PipelineGUI.java: the MySpell setup GUI.

The MySpellJNLPServer has the following packages and files:

- edu.princeton.function.myspell.jnlp:
  - MySpellJettyServerJnlp.java: implementation of an integrated Jetty server for MySpell that can be started using Java webstart.

Deployment

This section describes how to compile, configure and run the MySpell code.

MySpell consists of a server run as a Java application, and a client run in the users web browser. Before starting MySpell, the DistributedSpell search engine, and the SpellWeb2 web server should be running.

Note that the deployment requires configuring and compiling three projects before deployment: MySpell, MySpellJettyServer, and MySpellJNLPServer.

Quick Setup

To download, compile, and run MySpell are:

1. Download and install Eclipse Java IDE, the Google Web Toolkit plugin, and the Jigloo GUI builder plugin.
2. Download and install the Jetty web server.
3. Download, compile, and deploy DistributedSpell.
4. Download, compile, and deploy SpellWeb2.
5. Download the MySpell module from the CVS server at cvs.cs.princeton.edu.
6. Create a new Google Web Application project in Eclipse with the downloaded code.
7. Include the DistributedSpell and SpellWeb2 projects, and the gwt-user.jar external archive in the build path.
8. Download MySpellJettyServer project and create a new Java project.
9. Include DistributedSpell and SpellWeb2 projects, and the following libraries in the build path: gnu-getopt.jar, gwt-user.jar, jetty-server-<version>.jar, jetty-servlet-<version>.jar, and jetty-util-<version>.jar, jetty-http-<version>.jar, jetty-io-<version>.jar, jetty-security-<version>.jar, jetty-continuation-<version>.jar
10. Download MySpellJnlpServer project and create a new Java project.
11. Include as class folders: DistributedSpell/bin, SpellWeb2/war/classes, MySpell/war/classes, MySpellJettyServer/bin.
12. Include the following libraries in the build path: gnu-getopt.jar, gwt-user.jar, jetty-server-<version>.jar, jetty-servlet-<version>.jar, and jetty-util-<version>.jar, jetty-http-<version>.jar, jetty-io-<version>.jar, jetty-security-<version>.jar, jetty-continuation-<version>.jar
13. Configure MySpell by setting the following variables in MySpellJettyServerJnlp.java: WEB_SERVER, WEB_PORT, WEB_SERVER_STATIC_DIR, RMI_SERVER, and RMI_PORT.
14. Do a GWT compile on the MySpell project.
15. Run the MySpell/scripts/createFilelist.sh script.
16. Export a "Runnable JAR File" with the launch configuration "MySpellJettyServerJnlp - MySpellJnlpServer" and output MySpell/output/MySpell.jar".
17. Generate a keypair for signing the jar file:

```
keytool -genkeypair -dname "CN=Olga Troyanskaya, OU=Lewis-Sigler Institute, O=Princeton University, L=Princeton
```

1. Sign the exported jar file by running the MySpell/scripts/signJar.sh script:
2. Run the MySpell/scripts/copyFiles.sh script to copy the jar files and the client side binaries to the SpellWeb2 directory
3. Start the SpellWeb2 server and start MySpell from the "Private Search" page.

Source Code and IDE Setup

Install Eclipse with the Google Web Toolkit plugin, and the Jigloo GUI builder plugin.

Refer to the SpellWeb2 design document for detailed instructions for how to download and setup the MySpell GWT project, and the MySpellJettyServer Java project.

Debugging

To debug MySpell it is simplest to use the MySpellJettyServer class that can be started without configuring Java web start. First start SpellWeb2 using the runStandalone.sh script, then run MySpellJettyServer is run as an ordinary Java application using the arguments: "-r hostname:port -w hostname:port" to specify respectively the hostname and Java RMI port of SpellWeb2, and the hostname and port of the SpellWeb2 web interface.

Build Path Configuration

MySpell inherits from the DistributedSpell and SpellWeb2 projects, so these must added to the "Required projects in the build path" in the Java Build Path configuration in Eclipse. If MySpell is configured as a Google Web Application in Eclipse all required libraries should automatically be configured. These are: App Engine SDK, GWT SDK, JRE System Library, and JUnit4 for unit testing.

MySpellJettyServer throws exception classes from SpellWeb2 and DistributedSPELL so these projects must be added to the build path. In addition the following external libraries must be added: gnu-getopt.jar, gwt-user.jar, jetty-continuation-<version>.jar, , jetty-http-<version>.jar, jetty-io-<version>.jar, jetty-security-<version>.jar, jetty-server-<version>.jar, jetty-servlet-<version>.jar, and jetty-util-<version>.jar. The GNU getopt library is included with most Linux distributions, gwt-user.jar is from GWT, and the jetty jar files are from the Jetty web server.

MySpellJNLPServer should be configured to minimize the size of the generated jar file. So instead of including projects, include as class folders: DistributedSpell/bin, SpellWeb2/war/classes, MySpell/war/classes, MySpellJettyServer/bin. Then include the following libraries in the build path: gnu-getopt.jar, gwt-user.jar, jetty-continuation-<version>.jar, , jetty-http-<version>.jar, jetty-io-<version>.jar, jetty-security-<version>.jar, jetty-server-<version>.jar, jetty-servlet-<version>.jar, and jetty-util-<version>.jar.

The server side code should run on all JVM versions 1.6 or later. For the development we have used Google App Engine Java SDK 1.3.4, and Google Web Toolkit SDK 2.0.3.

Configuration

MySpell is designed to be run without the user having to do any configuration. It must therefore know at a minimum a server to contact for the configuration information. We have chosen to add this information at compile time for ease of implementation. The following constants must therefore be set in MySpellJettyServerJnlp.java before compiling the code:

1. The address and port of the public web server serving static content (WEB_SERVER and WEB_PORT).
2. The static content directory on the web server (WEB_SERVER_STATIC_DIR).
3. The address and port of SpellWeb2 Java RMI server (RMI_SERVER and RMI_PORT).
4. The address and port of the public server in the codebase field in MySpell/output/startMySpell2.jnlp

The web server is used to download the GWT object serialization meta-data file (.rpc), and MySpell forwards public data requests to the RMI server.

Compiling

Make sure that the MySpellJettyServerJnlp variables are set to the web-server host specific settings as described in the previous section.

MySpell is packaged as a jar file and the compilation is therefore a five step process:

1. In Eclipse, do a GWT compile on the MySpell project. This will create a war/ directory as described in the SpellWeb2 design document.
2. Eclipse will also compile the MySpellJettyServer and MySpellJnlpServer projects automatically and output the binaries in respectively MySpellJettyServer/bin and MySpellJnlpServer/bin.
3. In Eclipse, export a "Runnable JAR File"

1. Set launch configuration to "MySpellJettyServerJnlp - MySpellJnlpServer"
2. Set export destination to "MySpell/output/MySpell.jar"
3. Select "Extract required libraries into generated JAR"
4. (there may be some warnings about missing rpc files and duplicate entries, but these can be ignored).

JNLP Setup

Java web startup uses the MySpell/output/startMySpell.jnlp configuration file to start MySpell. In that file the codebase should be set to point to the web server serving static content.

Java web startup also requires the MySpell.jar file to be signed. To sign the document using the MySpell key run he following command in MySpell/output:

```
jarsigner -keystore myspell-keystore -storepass q3bls445 MySpell.jar myspell
```

This will sign the jar file using a key generated by running the following command in MySpell/output/:

```
keytool -genkeypair -dname "CN=Olga Troyanskaya, OU=Lewis-Sigler Institute, O=Princeton University, L=Princeton
```

Deployment

MySpell is deployed by copying the client side code and meta-data files to a directory served by a web server. The location of this directory must match the mySpellPrefix argument specified for SpellWeb2 (either in the web.xml file or as a command line argument). For example if SpellWeb2 is run using SpellWeb2JettyServer with the command line argument "-s /<workspace>/SpellWeb2/war/myspell" then the files should be copied to the SpellWeb2/war/myspell directory. If SpellWeb2 is run using a web container and mySpellPrefix is set to "myspell/" in the web.xml file, then the files should be copied to the SpellWeb2/war/myspell directory before SpellWeb2 deployment.

The files to copy are:

1. From MySpell/output: MySpell.jar, startMySpell.jnlp, pu.gif, myspell-tutorial.JPG, and filelist.txt. The last file is created by running the MySpell/scripts/createFilelist.sh
2. From MySpell/war: MySpell.html and MySpell.css
3. From MySpell/war: the myspell/ directory

All these files are copied to the SpellWeb2/myspell directory by running the MySpell/scripts/copyFiles.sh script (note that createFilelist.sh must have been run in adcvance).

Execution

MySpell is executed by clicking the link to jnlp file in the MySpell view in SpellWeb2. The browser will then execute Java web start which will download and execute the jar file.

Alternatively MySpell can be debugged by:

1. Executing the jar file can be executed directly: java -jar MySpell.jar (requires the web-server to have been configured correctly).
2. Starting MySpellJnlpServer/MySpellJnlpServer.java from Eclipse (requires the web-server to have been configured correctly).
3. Running MySpellJettyServer/MySpellJettyServer.java in Eclipse.

Testing

For the testing of MySpell we can reuse the code and methodolgy for the DistributedSpell tests and SpellWeb2 tests.

Status

- Unit tests:
  - All pass.
- Integration tests:
  - Search result correctness: all pass.
  - MySpell startup and GUI:
    - Linux (Ubuntu 10.04) + Chrome 7.0.5 + Open JDK 1.6 64-bits:
    - Linux (Ubuntu 10.04) + Firefox 3.6.10 + Sun SE 1.6 64-bits:
    - Windows 7 + Internet Explorer 8 64-bits + Sun SE 1.6:
    - OS X + Safari + Mac Sun
- System tests:

Unit Testing

We use EclEMMA [5] (http://www.eclemma.org/) (EMMA [6] (http://emma.sourceforge.net/userguide_single/userguide.html) for Eclipse) for code coverage testing, and the JUnit [7] (http://www.junit.org/) [8] (http://junit.sourceforge.net/doc/cookbook/cookbook.htm) framework for unit testing (JUnit is built in Eclipse). For additional details about GWT application testing, refer to the GWT Testing guide [9] (http://code.google.com/webtoolkit/doc/latest/DevGuideTesting.html) .

The unit test code is in MySpell/test. The files that ends with Test are the unit tests.

The following are not unit tested:

- The search functions as these are tested as part of the integration tests.
- The MySpell GUIs, since most functions require triggering user events.

Integration Testing

Methodology

The correctness of MySpell is compared against the result of the old sequential Spell implementation. A correct result is defined to be a result where the gene weights and datasets weights are maximum 0.05 different than in the sequential version. Exact match is usually not possible due to floating point inaccuracies. For the tests we use the yeast compendium used in the DistributedSpell tests, but we have split the datasets in each compendia into "private" and "public" datasets and then do the following search:

1. Search on only the datasets in the private set
2. Search on only the datasets in the public set
3. Search on datasets in both sets

The gold standards, queries, and reference results are all in the MySpell/test/data directory. The tests are implemented by the MySpellClient.java application, and run using the runIntegerationTests.py script.

The MySpell Startup and GUI tests should be run on different platforms with a mix of OS and browsers.

Search result correctness

1. Run the runIntegrationTests.py script in the MySpell/scripts/ directory. This test will run the following tests with the yeast compendium, and then with the human compendium:
   1. Private data search.
   2. Public data search.

    3. Private and public data search.
    4. Refined search using private, public, and private combined with public data

Note! The script automatically starts all necessary servers, but it assumes that the user knows when all servers have been initialized. This is usually done a few seconds after the cpu utilization of the servers drop to zero in top.

Note! Some of the test may fail due to numeric inaccuracies for very low scoring genes (rank > 5000). In this case a manual check of the error file is required.

MySpell startup and GUI

Before running this test, MySpell must have been deployed on a web-container as described in the deployment section.

1. Start SpellWeb2 with the yeast-public compendia.
2. Open the link http://<hostname>:9007/?locale=sce#private_search in a web browser.
3. JAR startup test:
    1. Expand the "Alternatives to Java Web Start" panel, and right-click and save the MySpell.jar file.
    2. Goto the folder where the MySpell.jar file was saved to.
    3. Start MySpell.jar using the command "java -jar MySpell.jar", verify that MySpell is started, and close the application.
4. Default settings test:
    1. Verify that MySpell was started and that the organism combo box has one element: "yeast".
    2. Verify that the processing steps is set to "Copy" (SINCE PIPELINE IS NOT IMPLEMENTED IN JAVA)
    3. Verify that "Output directory" is set to:
        1. "/tmp/myspell" on Linux.
        2. "C:\Windows\Temp\myspell" on Windows.
        3. "/tmp/myspell" on OS X.
    4. Verify that in "Output file settings" both, "Delete output files on removal" and "Delete output files on exit" are deselected.
5. In the browser right-click on the tutorial link and select to open the link in a new tab, then verify that the tutorial picture is displayed. **Tip:** for debugging purposes it is a good idea to configure Java to open a console for MySpell. In Linux: execute the ControlPanel command, then in the Advanced tab select "Java console: show console". On Windows the Java control panel is the Windows Control Panel, while on Mac it is in "Application | Java | Java Settings".
    1. Close the MySpell application.
6. Java Web Start test:
    1. In the browser, press the download and run link to start MySpell using Java Web Start.
    2. Chose to open the startMySpell2.jnlp file with Java Web Start.
    3. Wait until the jar file is downloaded, then choose "Run" in the security warning dialog box (The application's signature cannot be verified...).
7. Add dataset tests:
    1. Click on the "Add" button to add the CarmelHarel01 dataset from the private-yeast compendium. Then verify that the dataset was added to the list of private datasets, and that there is a copy of the file in the output directory.
    2. Click Add, then select jones03 and Pitkanen04 and click Ok. Verify that the these datasets are added to the list and that there are copies in the output directory.
    3. Select the jones03 dataset in the list and click Remove, then verify that the dataset is removed from the list, but that the file is not deleted.
    4. Add the jones03 dataset again.
    5. Enable "Delete output files on removal", then select the jones03 dataset and click remove. Verify that the dataset is removed from the list and the file in the output directory is deleted.
    6. Add the jones03 dataset again.
    7. Click "Add" and add select a non-dataset file, then verify that an error is displayed.
    8. Add the jones03 dataset, then verify that an error message is displayed stating that the dataset is already added.

    9. Enable "Delete output files on exit", close MySpell, and verify that the files were deleted.
8. MySpell backend initialization test:
    1. Click the Search button and verify that MySpell is opened in a browser.
    2. Open the "Manage Private Datasets" disclosure panel on the MySearch page and verify that there are no datasets in the panel.
    3. Add the CarmelHarel01, Jones03, Pitkanen04, and Spellman98 private datasets in the MySpell window, and verify that the four datasets appear on the "Manage Private Datasets" panel.
    4. Goto the "Show expression levels" tab, enter CTR9, click list, and verify that expression values for both the private and public datasets are shown.
    5. Delete the Jones03 dataset in MySpell, and verify that it is removed from the "Manage Private Datasets" panel.
    6. Delete the Pitkanen04 dataset in "Manage Private Datasets" panel, and verify that it is removed from MySpell.
    7. Add the Pitkanen04 private datasets in the MySpell window, and verify that the it appears on the "Manage Private Datasets" panel.
    8. Add the Jones03 private datasets in the MySpell window, and verify that the it appears on the "Manage Private Datasets" panel.
    9. Enable "Delete output files on removal" in MySpell, then remove all datasets in "Manage Private Datasets", and verify that all files are deleted.
    10. Close MySpell.

Browser GUI and data integration

To test the browser GUI three sets of tests are run: with only public datasets, with only private datasets, and with public and private datasets mixed.

The first test is with only public datasets:

1. Start SpellWeb2 using the yeast-all configuration file.
2. Start MySpell.
3. Open the MySpell web page in browser.
4. Do the yeast tests in SpellWeb2 integration tests.
    1. Note that the navigation differs since there is no "MySpell" page and that "New Search" is replaced by "My Search", and there is only one organism (yeast).

The second test is with only private datasets:

1. Start SpellWeb2 using the yeast-null configuration file.
2. Start MySpell.
3. Add the yeast-micro datasets as private datasets in MySpell.
4. Do the micro tests in SpellWeb2 integration tests.

The third test combines private and public datasets:

1. Start SpellWeb2 using the yeast-public configuration file.
2. Start MySpell
3. Add the yeast-private datasets in MySpell.
4. Do the yeast tests in SpellWeb2 integration tests.

Pipeline

TODO:

System Testing

Regression Testing

Eclipse can be configured to run the Unit tests at compile time (invoke JUnit in the build process).

Stress Tests

Evaluation

- Performance and scalability: the usual way.
- Security: not to be evaluated.

Performance

Previous Work

Future Directions