gee_fu v1
An extensible Ruby on Rails web-service database for GFF data.

Dan MacLean
dan.maclean@tsl.ac.uk

Contents

1 What is gee_fu?

gee_fu is a Ruby on Rails [1] based RESTful web-service application that serves genome feature data on request. It is ideally suited to serving large amounts of data such as those from high-throughput sequencing experiments. gee_fu can be used in conjunction with web-service viewports such as AnnoJ [2] to create very fast, data-rich, attractive, RDBMS agnostic genome browsers that can be easily extended into fuller custom web-applications using the powerful Rails framework.

1.1 Web services

Wikipedia [3] describes web services as follows

> Web services today are frequently just Application Programming Interfaces (API) or web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. In common usage the term refers to clients and servers that communicate over the Hypertext Transfer Protocol (HTTP) protocol used on the web. Such services tend to fall into one of two camps: Big Web Services and RESTful Web Services.

The use of web services means that data sources and application components can be distributed across the web. By maintaining standard data exchange formats between individual application components, such as the software that renders the view or the database that holds and serves the data individual components can be implemented according to which provides the better solution for the problem at hand.

1.2 How gee_fu fits into the web service world

gee_fu is an application centered around a database schema that holds GFF3 [4] formatted data. It has been designed with the needs of researchers wanting to view high throughput sequencing data in a genome browser. In the jargon of web applications it is the Model and Controller of the Model View Controller framework [5]. On request from a web service it will search the database on the basis of the parameters of the request and return a result. As an instance of a Rails application gee_fu can be used with any type of RDBMS currently supported by Rails [1] without modification. At this stage in development gee_fu is capable

of receiving and handling requests from AnnoJ [2], a web service based viewing engine for genomic data. It returns json [6] data which AnnoJ is able to render. We anticipate being able to serve up data in formats suitable for different applications as development progresses and we become aware of other rendering engines and web services that request data. Already gee_fu and AnnoJ make a powerful, fast and easily set up genome browser.

1.3 Obtaining gee_fu

The current version of gee_fu can be obtained from `http://github.com/danmaclean/gee_fu/`

2 Ruby, Ruby-on-Rails and the extensibilty of gee_fu

Ruby is, in its own words

> A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

If you aren't familiar with it, it is really worth a look. Ruby-on-Rails is

> a full stack, Web application framework optimized for sustainable programming productivity, allows writing sound code by favoring convention over configuration.

Rails is a powerful and straightforward way of creating powerful database centric web applications, it is easily extensible too. That's why Rails was chosen as the basis for gee_fu. With Rails at it's core gee_fu is a great tool for building a genome browser and custom web-tools that can interact with the database.

3 Setting up gee_fu

3.1 Pre-requisites

gee_fu requires the following software and packages to be available before installation

You will also require a RDBMS such as mySQL, ORACLE, PostGreSQL

Software	Version
Ruby	1.8.7 or later
Rails	2.3.4 or later
BioRuby	1.3.1 or later

3.2 Installation

Copy the gee_fu folder to a place on your hard disk from which you wish to work, we will call this place RAILS_ROOT (because this is what Rails calls it). A good place would be the cgi-bin directory on your system. It need not be in the cgi-bin while you build and test and have a only a few users (although your sys-admin may require this) since Rails comes with its own server for development. See the Rails documentation for further information when it comes to deploying the browser you have built into a production environment. Once gee_fu folder is copied, thats it installed and it is ready to configure.

3.2.1 A note for sys-admins

gee_fu is just an instance of a Rails application and has no special requirements other than BioRuby. See the Rails website [1] for full documentation.

3.3 Configuration

In this section I will describe the different stages in setting up gee_fu, though most of this material will be reference for now.

3.3.1 Configuring the database

You will need an RDBMS and an account that can create and select at least. These accounts are set up as in all Rails apps by editing the RAILS_ROOT/config/database.yml file. You do not need to create the database. The top of the file looks like this (for a mysql database), its YAML format

```
development:
        adapter: mysql
        database: db_name
        username: my_user
        password: 'my_pass'
```

```
  host: localhost
  timeout: 5000
```

change the adapter, database, username and password to suit whatever you want. See the Rails documentation for a fuller discussion of this file.

3.3.2 Configuring AnnoJ

The AnnoJ view requires a JSON format configuration file, to tell it about the genome and the service etc. JSON is a data-serialisation format that is fairly easy to write but there are a lot of brackets and it can be a pain to keep up. YAML is easier so edit the file RAILS_ROOT/config/config.yml. The most important are the tracks stanzas, this is where you tell AnnoJ the tracks that should be available in your browser. Looks like there are 3 track types: ModelsTrack, for aggregated transcript style gene models; RepeatsTrack, for strandless block objects like repeats; ReadsTrack, for high throughput sequencing reads that will be be shown at various zoom-levels. Here are the different configuration options for the rest

```
-id:                # a numeric unique id for the track

 name:      # free text name

 type:              # one of ModelsTrack, RepeatsTrack, ReadsTrack

 path:              # the place in the track selection panel
                    # where this track will appear

 data: /features/lane/x
                    # x = the experiment id from the gee_fu database

 showControls:      # either true to see the individual lane
                           # control button or just absent

 height: 80         # start up height of the
                    # track in pixels

 minHeight: 20  # height of track
                         # when minimised
```

```
maxHeight: 60      # height of track when maximised

single:            # true if the track is strandless
                   # otherwise absent
```

The active stanza lists the tracks that will be shown when the page loads, use the id of the track. The genomes bit is the url of the service that provides information about the genome, use genomes/x, where x is the id of the genome in the genomes table of gee_fu. Almost always 1. The bookmarks and stylesheets stanzas seem redundant (perhaps they are for unimplemented features of AnnoJ), leave them as-is. Location is where to focus AnnoJ on start up. Bases and pixels refer to the zoom level at startup. Its Bases:pixels. Admin is the name and contact details that appear in the information panel in AnnoJ in the information panel for the whole genome.
Once the config.yml file is made, prepare RAILS_ROOT/config/genome.yml, which allows you to describe the genome service. The three stanzas are fairly self-explanatory, the keys format, access and server seem to do nothing, leave them as unspecified, public and unspecified, respectively. The genome: assemblies stanza requires -size and id for each reference sequence in your genome. This can be easy to add if you only have a few chromosomes, but if you have hundreds of draft contigs then its a problem. If you have a fasta file of your assembly. There is a utility built into gee_fu, a rake task, that can do this for you. This can be done by executing rake add:assembly_to_yml from RAILS_ROOT.

3.3.3 Creating and populating the database

Once these config files are prepared, the rest of the process is fairly easy. A series of rake tasks help you to populate the database.

Creating the database Execute rake db:create from RAILS_ROOT
this makes the database according to the information you put into database.yml. Any problems here can probably be traced to the database not having the correct user or privileges.

Load the schema Execute rake db:schema:load from RAILS_ROOT

Add the config to the database	Execute rake add:config_to_db from RAILS_ROOT
Generate the AnnoJ config.js	Execute rake create:config from RAILS_ROOT
Add GFF data to the database	Execute rake add:features from RAILS_ROOT

3.3.4 rake add:features

The add:features task requires that you set some options
desc=”a description of the experiment”
gff=”./path/to/gff/file”
practice=true (optional flag.. allows you to do a dry run to test the gff file, doesnt commit data to the database, should be omitted to load the data
exp=”a concise name for the experiment”
trackinfo=”a yaml file like part of config.yml that describes this track. Optional.”
The experiment will be identified to the user and rake tasks with the value of exp. You can make rake tell you what experiments you have if needed, but pick carefully.

4 GFF and gee_fu

gee_fu supports the gff3 specification. In particular it enforces the feature type and requires that it is a member of the SONG SOFA ontology, either SO:xxxxx or the reserved term. The complex group attribute is serialised into JSON format and added. As gee_fu is designed with Next Generation Sequencing in mind, each experiment is expected to represent a separate sequencing experiment (or mulitple replicates thereof). All gff files that are loaded as one particular experiment will be shown in one track. It is therefore a good idea to split the features for different experiments into different gff files. The record will be marked with the experiment id to which it belongs. It is perfectly possible to add data to a track later if you need to. Suitable feature types for sequence reads are SO:0001423, dye_terminator_read; SO:0001424, pyrosequenced_read; SO:0001425 ligation_based_read; SO:0001426, polymerase_synthesis_read; SO:0000150,

read. The BioRuby GFF parser that gee_fu uses also enforces escaping of the following characters in gff lines, '% = & ,'. These must be escaped with the HTML codes. Here is some Perl that will carry out this substitution.

```
$gff =~ s/%/%25/g;
$gff =~ s/;/%3B/g;
$gff =~ s/=/%3D/g;
$gff =~ s/&/%26/g;
$gff =~ s/,/%2C/g;
```

The gff3 format has no specific place to add sequence or quality information for reads. There is a free text Note attribute in the group field though, which can serve this purpose. gee_fu uses two HTML like tags to identify the sequence and quality. Enclose the sequence between <sequence> and </sequence> tags and quality scores between <quality> and </quality> tags. The tags dont have to be used as pairs. When the GFF is added to the database, the tags and information is parsed into separate fields from the rest of the Note attribute and no longer remains within it. If you use the console to access data, there are methods for getting sequence and quality directly, without having to go through Note.

4.1 GFF and AnnoJ

Although gee_fu handles all the feature types of GFF, AnnoJ's support for complicated aggregated features like transcripts is fairly basic. Some features currently clash and fail to render. For an easy life with model tracks in AnnoJ use gene for the parent feature, five_prime_utr, three_prime_utr and CDS ONLY in the gff file that specifies the models. If you want to add other features, such as polypeptides or ESTs add them into tracks of their own.

4.2 A hack for showing reference sequence data in its own track with AnnoJ

AnnoJ has no specific reference sequence track. If you require one, here is a workaround. You will need to create a new experiment/track, distinct from the

5 Setting up gee_fu for use with AnnoJ

In this section, you will see which built in Rails tasks you can use and in which order you should use them to populate the database and make an AnnoJ instance.

5.1 Rake Tasks

Rake tasks are Rails built in way of automating certain processes during development of the database. from RAILS_ROOT you can execute $rake --tasks$ to get a full list. The majority in there are the standard Rails ones, but here are the ones added specific to gee_fu.

These tasks are all you need to populate the database and to get it to respond

add:assembly_to_yml	add assembly information to genome.yml from fasta file
add:config_to_db	load the config information into the database from the yaml file
add:features	add features in an experiment (track) to the database
remove:features	remove features in an experiment (track) from the database
list:experiments	lists the experiments added to the database already
create:config	creates the AnnoJ config.js file based on the information in config.yml
create:song_list	turns a DagEdit style file into YAML and puts it in the lib folder as song.y

to AnnoJ requests.

5.2 Brief Walkthrough

1. Edit the database.yml file
2. Edit the config.yml file (do this every time you want to add a new track)
3. Execute rake db:create (this will tell the RDBMS to create your db)
4. Execute rake db:schema:load (this will load the schema into the db)
5. Execute rake add:config_to_db (this will add the config from the config file to the table in the db)
6. Execute rake create:config (this makes the AnnoJ config.js file, rerun whenever you add a new track to the config.yml file)
7. Execute add:features, (this will add a features track/experiment to the database and enter the gff data)

6 Console

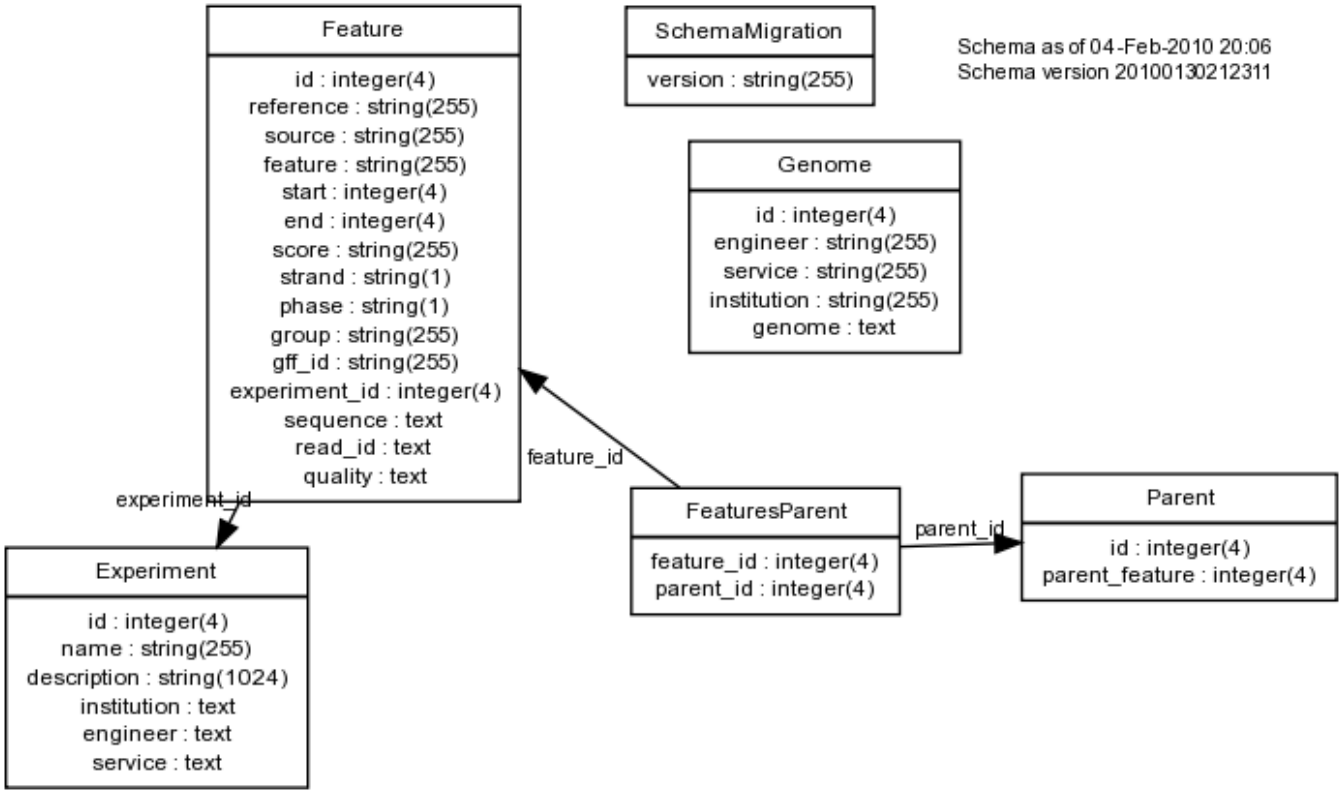
Once data is added to the database you can search it with the Rails console. This is a standard part of Rails and there are plenty of web-tutorials.

7 Scripts

In the same way that you can use the console you can access the data in the database script-o-matically with your favourite language, but why not try Ruby and ActiveRecord, the ORM layer in Rails.

8 Schema

The schema is very straightforward and easily extended. It consists of a central Features table and a many to many join table Parents that indicates which features are parents (according to their gff records) of which other features.



9 Extending the database and creating new functionality

You can extend the database exactly as if it were any other Rails application. See the Rails documentation for conventions for creating and naming new tables, Rails prefers convention over configuration so you should pay attention to these. Adding new functionality to the web app is covered by the same documentation.

10 Where to find more info

If you get really frustrated with the software, feel free to complain to me dan.maclean@tsl.ac.uk. I freely admit its early days for this project so it is likely to wobble, however, a lot

of your initial problems will be answered in the Rails community pages, please look there if your problem looks like it might be on related to Rails more directly than this particular instance of a Rails app.

References

[1] Ruby on rails. http://rubyonrails.org/.

[2] Annoj. http://www.annoj.org/.

[3] Wikipedia web service article. http://en.wikipedia.org/wiki/Web_service.

[4] The gff3 standard. http://song.sourceforge.net/gff3.shtml.

[5] Wikipedia mvc article. http://en.wikipedia.org/wiki/Model-view-controller/.

[6] Json data format. http://www.json.org.