GeeFu v0.9
An extensible RESTful Ruby on Rails web-service database for GFF data.

Dan MacLean
dan.maclean@tsl.ac.uk

Contents

1 About GeeFu

GeeFu 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. It can use bam files from next-generation alignments directly as a data source. Simple web-requests via URLs can be used to return data in common web-formats like XML or JSON. GeeFu can be used in conjunction with web-service viewports such as AnnoJ [2] to create very fast, data-rich, attractive, RDBMS agnostic genome browsers and can be easily extended into custom web-applications using the powerful Rails framework.

1.1 Web services and GeeFu

Wikipedia [3] describes web services as follows

> 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.

GeeFu is an application centred around a database schema that holds GFF3 [4] formatted feature data. It has been designed with the needs of researchers wanting to work with high throughput sequencing data from a central place. It is a tool for sharing and storing next-gen data with a minimum of fuss. It is easy to retrieve data via scripts or URL requests, upload data and visualise 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 GeeFu can be used with any type of RDBMS currently supported by Rails [1] without modification. At this stage in development GeeFu is capable of receiving and handling requests via a simple interface and can

send data to return AnnoJ [2], a web service based viewing engine for genomic data. It returns data which AnnoJ is able to render.

1.2 Obtaining GeeFu

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

2 Installing GeeFu

2.1 Pre-requisites

GeeFu requires the following software and Ruby gem packages to be installed

Software	Version
Ruby	1.8.7*
RDBMS	eg MySQL (version 5.1.39)

*Ruby 1.9.1 does not yet work with the mongrel webserver we need for the development and tutorial environments, so isn't supported, yet. When this is sorted, we'll be happy to support it. We are also looking at using Rubinius as an alternative implementation of Ruby, but are having some hiccups with our samtools-ruby module. Usage of features solely from the database looks fine.

Ruby gem	Version
rails	2.3.4 or later
RubyGems	1.3.5 or later
bio	1.3.1 or later
json	1.1.9 or later
ffi	0.6.3 or later
mysql*	2.8.1 or later
mongrel	1.1.5 or later

*or whatever the gem is for your RDBMS of choice.

The prerequisites for these will be required too. For an easy life use the RubyGems package manager to install them for you. If you intend to use AnnoJ as a viewer,

then you will need to have an internet connection, some of the components of AnnoJ need to be retrieved on request from the AnnoJ server, it too is a Web Service.

2.2 Installation

GeeFu is just an instance of a Rails application and has no special requirements other than those above. Rails installation is well documented and in most cases is very easy. See the Rails website [1] for full documentation. GeeFu is very easy to install on a local machine for devlopment and setup purposes, for larger production installs see the instructions in the Rails documentation. 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). It need not be in the cgi-bin or equivalent, since Rails comes with its own server for development. See the Rails documentation for further information when it comes to deploying the database or browser you have built into a production environment. Once the gee_fu folder is copied, thats it installed and it is ready to configure.

2.3 samtools-ruby and GeeFu

The Ruby module samtools-ruby, written by Ricardo Gonzalez-Ramirez `http://github.com/homonecloco/samtools-ruby` is a Ruby wrapper around the popular and very useful samtools C library, and is the method by which GeeFu is able to use BAM files as a source of read information. samtools-ruby is in early development and is not yet easy to install for all platforms, Ricardo does plan to have a gem soon. For the tim being the Ruby code is provided with GeeFu so doesn't need installing especially for GeeFu. However, the libbam file at the heart of the samtools C library does need to be compiled on a per machine basis. The GeeFu package includes a libbam.dylib for 64 bit Apple machines so should work fine on that platform. For other systems you will need to download the libbam source code and compile it for your own platform then make sure it is accessible to GeeFu. The server root (e.g $RAILS\_ROOT$) is a common place for it to go. It is beyond the intent of this document to describe the process of installing and compiling the libbam for multiple systems especially as the problem is likely to go away once samtools-ruby is available as a gem. Hopefully this will be resolved soon. If you do have lots of problems, email us and we'll do our best to help.

2.4 Quick Start

In this section we'll look at setting up and adding some sample data into GeeFu as a way of understanding the basic workflow of set up and usage.

1. Copy the GeeFu directory to somewhere on your disk, this is now $RAILS\_ROOT$
2. Open a terminal or command line and cd into $RAILS\_ROOT$
3. Open the file $RAILS\_ROOT/config/database.yml$ in your text editor. It looks like this.

```
development:
    adapter: mysql
    database: geefu_sample
    username: my_user
    password: ''
    host: localhost
    timeout: 5000
```

   In the development block change the value of the adapter attribute to the type of RDBMS that you have (see the Rails documentation for more info) and change the value of the database attribute to the name you want for your database. Also change the database permissions (username, password and host) as appropriate. The database itself will be created later.

4. Edit the $RAILS\_ROOT/config/config.yml$ file. The file is already set up for this tutorial so you don't need to make changes this time, but its worth a look. You will need to edit it every time you start a new project and add a new experiment.
5. Open and edit the $RAILS\_ROOT/config/genome.yml$ file. The file is also already set up for this tutorial so you don't need to make changes. You will need to edit it every time you start a new project.
6. Use a fasta file of reference sequences and add the length and names of the references to the config.yml file

```
rake add:assembly_to_yml fasta=public/sequences/
   sample_reference_TAIR9_Chr1.fna
```

7. In a terminal cd to $RAILS\_ROOT$ and create the database based on the info in database.yml.

```
rake db:create
```

8. If the username you specified in database.yml doesn't exist in your RDBMS, create it and grant select and insert privileges on the database you specified in database.yml.

9. Add the schema to the database.

```
rake db:schema:load
```

10. Add the information from config.yml to the genomes table.

```
rake add:config_to_db
```

11. The $max\_allowed\_packet$ variable in your RDBMS may need to be increased to allow for the size of the largest reference sequence in your fasta file, see your RDBMS manual for information on how to do this. For this example data and with MySQL you can do this by executing

```
set global max_allowed_packet=1000000000;
```

    Where the number is a big enough amount of memory in bytes to take your longest sequence. See the information here `http://dev.mysql.com/doc/refman/5.1/en/packet-too-large.html`

12. Creates the file $RAILS\_ROOT/public/javascript/config.js$, which is required by AnnoJ by executing

```
rake create:config
```

    This is really just a JSON version of config.yml, so if you make any changes to config.yml and add them to the database, remember to regenerate the config.js file or AnnoJ won't pick them up.

13. Add the experiment info from the sample data files First add the gene models from the sample gff, this will get the id 1 automatically.

```
rake add:features desc="TAIR 9 Sample Features" exp="gene_models"
    gff=public/sample_gffs/sample_features.gff genome=1
```

Then add the read information into the RDBMS from the gff file

```
rake add:features desc="sample_reads" exp="
    gff_format_reads_from_rdbms" gff=public/sample_gffs/
    sample_reads.gff genome=1
```

Then link the database to a bam file containing read alignments

```
rake add:features desc='sample reads from bam' genome=1 exp='bam
    format sample reads from bam file' bam=public/bam/aln.sort.bam
    desc='sample reads from bam' genome=1
```

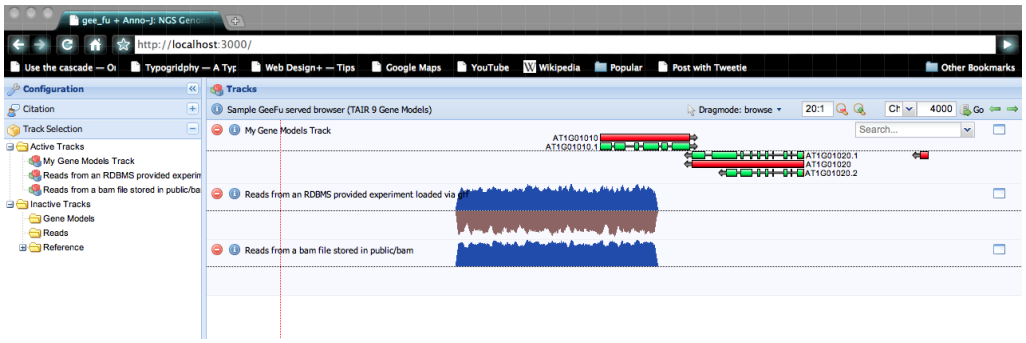
Now add a reference sequence track from the fasta file provided

```
rake add:sequences fasta=public/sequences/
    sample_reference_TAIR9_Chr1.fna genome=1
```

14. Start Rails' Mongrel server on the local machine

```
script/server
```

Point your browser at localhost:3000 and you should get a view of the couple of models and reads as below.



Zoom in a bit and the reads tracks should turn into individual reads, eventually with the sequence (you may need to use the resize and scale functions to see the reads and bases). To see the reference sequence you can open the Reference folder on the left and drag the inactive reference track to the Active Tracks above, it will then render in the browser's view.

3 Setting up GeeFu

In this section I will describe in a bit more detail the different stages in setting up GeeFu.

3.1 Setting up the database

3.1.1 Configuring the database with database.yml

You will need an RDBMS and an account with which you can create, insert and select. 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, this is done by a Rake task, it mustn't exist already. The top of the file looks like this (for a mysql database), it is in YAML format.

```
1 development:
2   adapter: mysql
3   database: db_name
4   username: my_user
5   password: 'my_pass'
6   host: localhost
7   timeout: 5000
```

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

3.1.2 Creating the database

The database is created based on the information in database.yml by calling the rake task $rakedb : create$. The schema is added to the database with $rakedb : schema : load$ from $RAILS\_ROOT$, this takes the schema from $db/schema.rb$.

3.1.3 Adding a new genome to the database

GeeFu can hold multiple genome versions. Genomes and metadata for the service etc are added to the genome table in the database from the genome.yml file via $rakeadd : config\_to\_db$. The genome.yml file looks like this, again it is in YAML format:

```
1 ---
2 institution:
```

```
  name: Your Institute
  logo: images/institute_logo.png
  url: http://www.your_place.etc
service:
  format: Unspecified
  title: Sample GeeFu served browser (TAIR 9 Gene Models)
  server: Unspecified
  version: Vers 1
  access: public
  description: Free text description of the service
genome:
  assemblies:
  - size: 30427671
    id: Chr1
  version: TAIR9
  description: Chr1 from TAIR9
  species: Arabidopsis thaliana
engineer:
  name: Mick Jagger
  email: street.fighting.man@stones.net
```

The genome stanza is the important one, in this section the id and length of each of the reference assemblies/chromosomes are defined for that genome. This stanza can be populated automatically from a fasta file of sequences using $rakeadd : assembly\_to\_yml$ before adding the information to the database. The genome table holds only metadata about the genome, who made it, who serves it etc. As the genome table is populated a linked table called references is populated with the size and id information for each of the reference assemblies/chromosomes. When a genome is added it gets an id and the references are linked to the genome with this id.

3.2 Adding features as part of an experiment

The Experiments table holds metadata about an experiment and all features belong to an experiment, although the Features table holds feature data, whether that is gene model information or next generation sequencing reads. The basis for grouping related features in the database is as an Experiment. It is therefore a good idea to split the features for different experiments into different gff files. The feature record will be marked with the experiment id to which it belongs. It is perfectly possible to add features data to an experiment later if you need to, reads or models from the same experiment but collected at different times can be uploaded separately but will stay connected to the same experiment. Adding

features into experiments is done with the rake task $rakeadd : features$, which will also create the experiment if need be. This task has lots of options.

Option	Summary
exp= (required)	The short string that the experiment will be identified by, the task will look for an existing experiment with this name and add the features to it if it exists. Otherwise a new experiment with a new ID is created.
desc= (optional)	A piece of text that describes the experiment
gff= (optional)	A gff file containing the gene models or read features to be added. If a gff file is used the features are added to the database which is indexed as loading occurs. For gene model features composed of multiple features like mRNAs made of exons the parents are identified by the gff group ID attribute and linked via the FeatureParents and Parents table. This behaviour along with the index makes uploading a gff quite slow, though not unbearable. The Parent lookup means that a feature must have its parent present in the database already, or it can't be associated. Keep this in mind when preparing your GFF and put all the parent features into the file above the child features, later versions of GeeFu will contain a GFF validator and maybe even a db fixer to find parents when this gets messed up. If a GFF is provided a BAM file must not be.
bam= (optional)	A path to a sorted BAM file containing an alignment of reads to a reference. The reads should be aligned to a reference in the genome this bam file will be linked with.
genome= (optional)	The database ID of the genome which these features belong to. If not provided the task will assume the genome ID is 1
no_parents= (optional)	true or false Stops the task from looking up parents for features that don't have them anyway, like reads or repeats
trackinfo= (optional)	a YAML file like genome.yml file that describes some metadata for this experiment. If omitted the genome information already loaded will be used.
practice=	true or false Allows you to run through the feature upload without actually committing features to the database. A useful check for some errors in GFF formatting and restricted term usage.

3.3 Removing features

All features in a particular experiment can be removed with $rakeremove : featuresexp = some\_experiment$. This action is non-reversible.

3.4 Adding reference sequences to the database

Fasta sequence of the reference can be added into the database with $rakeadd : sequencesfasta = some\_fasta\_filegenome = some\_genome\_id$ If no genome id is provided then ID 1 is assumed. You'll need to have added a genome already for this to work. For long sequences, you may have problems with the maximum amount of data that the RDBMS will allow to be uploaded. Check the RDBMS documentation for fixes, in MySQL opening a new mysql process and entering $setglobalmax\_allowed\_packet = 1000000000;$ will probably sort this out. The number in bytes should be big enough to hold your longest sequence.

3.5 Altering the list of valid feature types

GeeFu by default uses the SONG ontology to restrict the terms allowed for feature types. These are defined in song.yml. If you want to overwrite this with your own set of terms or update to a newer version provide a DagEdit file to the task $rakecreate : song\_listdag = some\_dag\_edit\_file$.

Valid read feature types are restricted elsewhere, in the actual code of GeeFu. These will be editable in a future version but are currently restricted to :
SO:0001423 dye_terminator_read
SO:0001424 pyrosequenced_read
SO:0001425 ligation_based_read
SO:0001426 polymerase_synthesis_read
SO:0000150 read
The SO:XXXXXXX or restricted term can be used, not both.

4 GeeFu's approach to valid GFF

GeeFu tries to support the GFF3 specification closely. In particular it actively enforces that the feature type is a member of the SONG SOFA ontology, either SO:xxxxx or the reserved term. If a feature's type is not in this list (case sensitive) during upload GeeFu will abort and try to let you know where the GFF file went

wrong. You can always check by using the practice option of the add:features task
GeeFu does not do any sanity checking of parent/child relationships.
GeeFu requires that the GFF ID (the ID attribute in the GFF group field) is unique, but does not test this, yet. Records with duplicate GFF ID may get lost in certain searches and confuse the creation of parent/child relationships.
The BioRuby GFF parser that GeeFu uses also requires escaping of the following characters in GFF lines, '% = & ,'. These must be escaped with the HTML codes or GFF parsing will fail. Here are some Perl-like regular expressions that will carry out this substitution.

```
1     $gff =~ s/%/%25;/g;
2     $gff =~ s/;/%3B;/g;
3     $gff =~ s/=/%3D;/g;
4     $gff =~ s/&/%26;/g;
5     $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. GeeFu 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. For example

```
1 Note=<sequence>ATCG<\/sequence><quality >!!!! <\/quality >;
```

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.

5 Getting data from the database with the RESTful interface

GeeFu can be accessed with a URL GET request and appropriate parameters, just typing the URL into a browser will return data but this is most useful for serving data to web services or web pages or scripts that can access a server via http, this is what makes GeeFu so good for sharing data. Data can be returned in XML or JSON format. All the URLs in this section assume that the base URL is prepended if not shown, eg $www.myserver.ac.uk/specific\_gee\_fu\_stuff$ is just $/specific\_gee\_fu\_stuff$. Typically the syntax for all records in a table is $/table.format$ eg $/genomes.xml$. The syntax for individual records is $/table/id.format$

eg $/genomes/1.xml$. Optional parameters appear after this with the format $?param = value$ and multiple parameters are separated by &. The Rdoc at $RAILS\_ROOT/doc/app/index.html$ describes the available methods and usage. You should look for files called XxxxController, where Xxxx is a table in the database that you are interested in getting information from.

5.1 Getting the list of genomes

Genome information is accessed using the standard RAILS route $/genomes$. The format is specified by the extension $/genomes.xml$ returns XML formatted data $/genomes.json$ returns JSON formatted data.

5.2 Getting a specific genome

Use $/genomes/id.format$ eg $/genomes/1.xml$ for XML data, $/genomes/2.json$ for JSON data. Accessing an ID that doesn't exist will return an empty result.

5.3 Getting the list of experiments

Experiment information (not features) is accessed using the standard RAILS route $/experiments$. eg $/experiments.xml$ or $/experiments.json$.

5.4 Getting a specific experiment

Use $/experiment/id.format$ eg $/experiments/1.xml$ for XML data, $/experiment/2.json$ for JSON data. Accessing an ID that doesn't exist will return an empty result.

5.5 Getting the list of references for all genomes

Reference information (not sequence) is accessed using the standard RAILS route $/reference$. eg $/references.xml$ or $/references.json$.

5.6 Getting the list of references for a specific genome

Use $/references/genome\_id.format$ eg $/references/1.xml$ for a list of references for genome 1 as XML data, Accessing an ID that doesn't exist will return an empty result.

5.7 Getting a specific reference by genome id and name

Use the name parameter after the genome id
ie $/references/genome\_id.format?name = reference\_name$. For example, $/references/1.xml?nam$
$reference_name$

5.8 Getting a reference's sequence

The general parameter for returning the sequence for the reference is sequence=true as a GET parameter, eg $/references/1.xml?sequence = true$

5.9 Getting Feature information

It isn't usually feasible to get a list of features, rather features are returned in answer to a request that sets limits on the genome, experiment, reference, start and end. Entire objects or depth over the range can be returned.

5.9.1 Getting all feature objects within a start and end point on a reference in an experiment

Use $/features/objects/experiment\_id?parameters$. Required params are reference=ref_name, start=start_nucleotide, end=end_nucleotide, eg /features/objects/1?reference=Chr1&start=1&end=10000. Optional parameters: format=json or xml (default = xml), sets the format for return data overlap=true or false (default = false) if true objects that overlap the limits will be returned and not just those completely contained within. If the data is coming from a BAM file the behaviour of samtools-ruby is currently such that the objects returned will always be selected as if overlap=true. type=some GFF feature type, only objects with GFF feature type will be returned.

5.9.2 Getting a summary of coverage information between a start and end point on a reference in an experiment

Use $/features/depth/experiment\_id?parameters$. Required parameters are reference=ref_name, start=start_nucleotide, end=end_nucleotide,
eg $/features/depth/2?start = 10$&$end = 1080$&$format = json$&$reference = Chr1$
Optional Parameters: overlap=true or false (default = false) if true objects that overlap the limits will be returned and not just those completely contained within.

If the data is coming from a BAM file the behaviour of samtools-ruby is currently such that the objects returned will always be selected as if overlap=true.

The hash summarising depth in the region selected is quite complex, roughly it is position => strand => nucleotide = count of nucleotides in JSON it looks like this

```
{"some_nucleotide_position":
  {
  "+":
    {"A":0,
     "T":0,
     "G":0,
     "C":0,
     "N":0,
     "strand_total":0
     },
  "-":
    {"A":0,
     "T":0,
     "G":0,
     "C":0,
     "N":0,
     "strand_total":0
     },

  "position_total":0
  },
  ...
  "region_total":0
}
```

For every covered nucleotide position there is a hash with keys for + and - strand and total read bases covering that position (position_total). Each strand has a hash that has keys for each of A,T,C,G or N and total read bases covering that position on the strand (strand_total). The whole thing has a key called region total that counts the total number of read bases in the entire region.

6 Getting data from the database with the Rails console

Rails has an interactive console interface with which you can use to access the data and objects using the Models and methods defined in GeeFu. An enthusi-

astic introduction can be seen here
`http://slash7.com/2006/12/21/secrets-of-the-rails-console-ninjas/`
. The syntax is pure Ruby and is very straightforward. The models from GeeFu that are already loaded are the tables of the database and the standard methods credited to the objects are available. Some useful ones include the $Feature.find\_in\_range$ and $Feature.find\_in\_range\_no\_overlap$, (the Rdoc for the Feature class in $RAILS\_ROOT/doc/app/index.html$ describes each of the methods.

7 Getting data from the database with Ruby scripts

8 Using AnnoJ as a view port

AnnoJ is a "Anno-J is a Web 2.0 application designed for visualizing deep sequencing data and other genome annotation data". AnnoJ does require that some sort of service send it the data for it to render in the first place, GeeFu is great for this and can use AnnoJ as a fairly straightforward way to visualise the data it holds. GeeFu is set up to enable interaction with this view port, but can be easily modified to work with others.

8.1 Requirements

8.1.1 Browser requirements

AnnoJ insists on W3C compliant browsers, Internet Explorer doesn't hack it. Right now, Firefox 3.6 is giving some rendering issues though Firefox 3.5 seems fine. The latest versions of Chrome and Safari on my Mac seem fine too.

8.1.2 Connection requirements

AnnoJ is a web-service browser and the idea is that all of its components are retrieved from over the internet. GeeFu carries most of the components of AnnoJ to make GeeFu as stable as possible but some can't be retrieved or kept locally so an internet connection is required.

8.2 Configuring AnnoJ for use with GeeFu

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 GeeFu allows you to maintain a YAML file and convert it to the $RAILS\_ROOT/public/javascripts/config.js$ AnnoJ uses. 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. As far as I can tell from the undocumented Annoj it looks like there are 3 track types: ModelsTrack, for aggregated transcript style gene models; RepeatsTrack, for strandless block objects like repeats and ReadsTrack, for high throughput sequencing reads that will be be shown at various zoom-levels and the reference sequence GeeFu provides that is sent to AnnoJ as if it is a big read. Here are the different configuration options for the rest

```
1     -id:    # a numeric unique id for the track
2      name:    # free text name
3      type:      # one of ModelsTrack, RepeatsTrack, ReadsTrack
4      path:      # the place in the track selection panel
5            # where this track will appear
6
7      data: /features/method/x  # method = annoj - for feature (models,
           reads and other GFF stuff)
8            #       or chromosome for the reference sequence track
9            # x = when method = annoj x is the experiment id when
                 method =  genome id from the gee_fu database
10     showControls:    # either true to see the individual lane
11           # control button or just absent
12     height: 80  # start up height of the
13           # track in pixels
14     minHeight: 20  # height of track
15                # when minimised
16     maxHeight: 60 # height of track when maximised
17     single:    # true if the track is strandless
18           # 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 GeeFu. Usually 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, you may wish to prepare $RAILS\_ROOT/config/genome.yml$, which as well as adding the genome information for GeeFu 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 GeeFu, a rake task, that can do this for you. This can be done by executing $rake add : assembly\_to\_yml from RAILS\_ROOT$.

8.3 Accessing the view in a web browser

The configured GeeFu/AnnoJ instance is accessed via the index.html which resides in the $RAILS\_ROOT/public folder$. Point your browser at this to start the view. See Rails documentation for more information. If you used the built in server for development try localhost:3000 otherwise the AnnoJ view will be wherever your sys-admin installed the server for you.

8.4 GFF and AnnoJ

Although GeeFu handles all the feature types of SONG, AnnoJ's support for complicated aggregated features like transcripts isn't well documented and in my experience sometimes some feature types render over others or not at all. For an easy life with model tracks in AnnoJ use genes and mRNA, five_prime_utr, three_prime_utr and CDS and exon 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. As a workaround to this behaviour GeeFu will only attempt to allow aggregation of the following sub-feature types 'intron', 'exon', 'CDS', 'five_prime_UTR', 'three_prime_UTR', 'start_codon', 'stop_codon', 'match_part' , any other feature types won't aggregate. You can alter this by altering the list in the to_annoj method in $/RAILS\_ROOT/app/models/feature.rb$. We may make this a configurable list in the future.

8.5 Changing feature rendering in AnnoJ

The colour and fill of feature objects in the view are determined in the $RAILS\_ROOT/public/stylesheets/plugins.css$ file. The class of the element refers to the GFF feature type, so $.mRNA\{someCSS\}$ is what defines the look of mRNA features. The images can be changed if you wish and are stored in RAILS_ROOT/public/images. If you want different tracks with the same feature type to have different appearances you are out of luck. To achieve this you will have to change feature types on one of the tracks.

9 Working with GeeFu

9.1 Rake Tasks

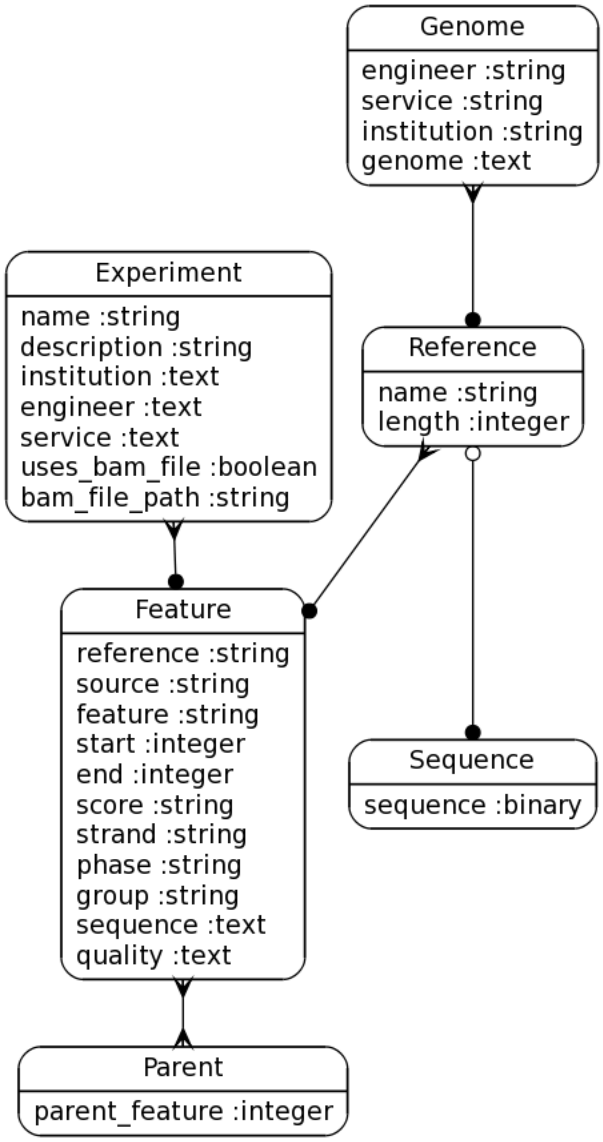
Rake tasks are Rails built in way of automating certain processes during development of the database. from RAILS_ROOT you can execute $rake --tasks$ to get a full list. The majority in there are the standard Rails ones, but here are the ones added specifically for GeeFu.

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.yml

9.2 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. References all

belong to a genome, as do Experiments and features all belong to an experiment. Sequences belong only to a reference.



9.3 The tools page

We provide a model-less controller as a place to hold web tools that act on the database. Currently we have a genome sequence extraction tool which renders

a web-form and retrieves sequence as per the information specified by the user in the form using the code $/tools$. The web form itself is specified in $RAILS\_ROOT/app/views/index.html.erb$ and the code it calls is the method $genomic\_sequence$ in $RAILS\_ROOT/app/controllers/tools\_controller.rb$. The output goes to the output webpage $/RAILS\_ROOT/app/views/genomic\_sequence.html.erb$ in a Ruby instance variable and is interpreted by the erb (embedded Ruby) in that file.

9.4 Extending the database and creating new functionality

You can extend the database exactly as if it were any other Rails application. Adding new tables and datatypes is done easily, just add these as if they were new Rails models. A good way is using Rails migrations to build your tables (that is, get Rails to do the heavy lifting in the Sql side and version your database for you) then use Rails lovely magic convention over configuration to create all the code and models. This is really the start of building Rails apps so see the Rails documentation. 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. Briefly, the workflow is to create some web form for collecting input, that is accessible at some url, like the tools index.html.erb. The form sends information to the controller and action specified, like $RAILS\_ROOT/app/controllers/tools\_controller.rb$ and $genomic\_sequence$. The controller and action use the information from the form to do some work on a database, put the result into instance variables and call the view that matches the action like $RAILS\_ROOT/app/views/tools/genomic\_sequence.html.erb$. The view knows how to render the data in the instance variables.

10 Tips for a moving GeeFu to a production server

Once you've got comfortable using GeeFu and want to move to a production server the web-server you choose can be very important in determining the performance of the application. We have had excellent results from Phusion Passenger `http://www.modrails.com/`

11 And Finally...

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 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.