Research Report

Earth Gui plugin

Jian Huang

1. BACKGROUND

In computing, a plug-in consists of a computer program that interacts with a host application (a web browser or an email client, for example) to provide a certain, usually very specific, function “on demand” [1]. There are lots of reasons for us to create and / or use plug-in. Some main benefits of it are:

- Easy way for developers to create new capability to extend applications.
- Separating application codes so that each unit can be fixed or updated on their on release schedule.
- Unnecessary for core developers to include every new feature under the sun [2].

Meanwhile, Plug-ins differ slightly different from extensions, which modify or add to existing functionality. Plug-ins generally rely on the host application's user interface (API) and have a well-defined boundary to their possible set of actions. [1]

2. INTRODUCTION

Plug-ins currently can be created in different ways, such as using API(s), jointing Extension Points. And of course, we can also use some plug-in tools. Most of them are very handy and powerful, like Plux.NET for .net, Java Plug-in Framework (JPF) for Java. Even the new web application framework, Ruby on Rails, has its own plug-in tool, Rails Plugin. However, this research report will talk about another way to create plug-ins into a particular Ruby on Rails’ project. And use an open source - Earth for instance, which is set up by the company Rising Sun Pictures (RSP) using Rails [3].

3. Rails Plugins

Rails has a lot of features but the core team is very cautious about adding any new functionality. Part of what has made it such a good framework is that they don't allow features in that aren't necessary or highly useful. This means that most of the cool add-ons we'd like to see have to end up as plugins [4]. Therefore, Rails Plugins is aim to develop for various parts of the Rails framework [5]. Here, you can see the basic layout of a plugin by running the standard plugin generator.

```
# Use the generator to make a rails_plugin_example plugin
./script/generate plugin rails_plugin_example

create vendor/plugins/rails_plugin_example/lib
create vendor/plugins/rails_plugin_example/tasks
create vendor/plugins/rails_plugin_example/test
create vendor/plugins/rails_plugin_example/README
create vendor/plugins/rails_plugin_example/Rakefile
create vendor/plugins/rails_plugin_example/init.rb
create vendor/plugins/rails_plugin_example/install.rb
create vendor/plugins/rails_plugin_example/lib/rails_plugin_example.rb
create vendor/plugins/rails_plugin_example/tasks/rails_plugin_example_tasks.rake
create vendor/plugins/rails_plugin_example/test/rails_plugin_example_test.rb
```

Numbers of these files are necessary, but they each do different things. You might write a plugin that only has a tasks folder or another that only has a lib folder. Here is what each piece does [6]:

- init.rb: Runs everytime the Rails app is started. Useful for mixing-in a helper module so all your views can use it.
- install.rb: Runs one time only when the plugin is first installed. Can be used to copy required files or to show installation instructions.
- Rakefile: Generates documentation and runs tests for the plugin.
- lib/: Contains actual Ruby code. New models and other libraries in this folder will be automatically available to your Rails app. Helpers can be included specially, using the init.rb file.
- tasks/: Drop a .rake file here as you would in lib/tasks. It’s automatic…no other action is needed!
- test/: Write tests to verify the operation of your plugin.

Now, it is easy to find that rails plugin can do more in a range of areas, including:

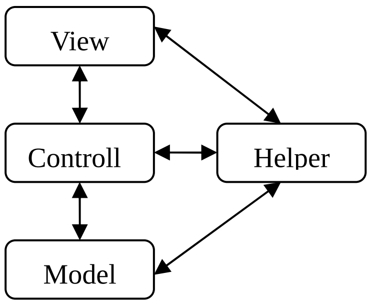
- Extensions to ActiveRecord functionality
- Helper methods
- New template engines

There are advantages of using Rails Plugins. However, the following are some of its setback:

1. There is no easy-to-use plugin manager created to simplify the installation and management of plugins [5].
2. Controller plugin cannot be directly to written.
3. It cannot avoid changing original coding to use plugin.

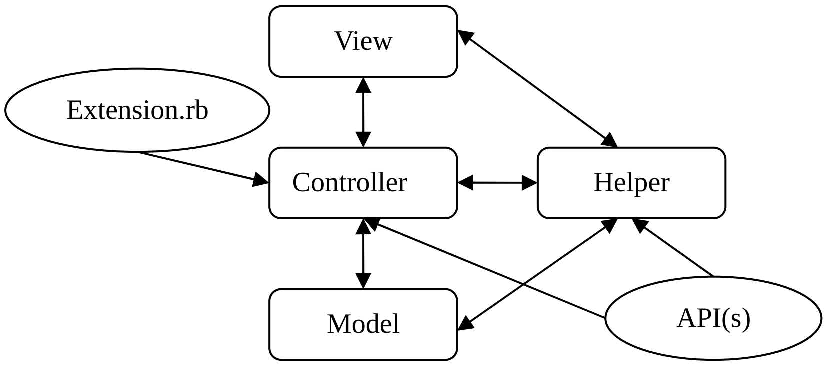
4. Development

Most of Rails development happens within the app directory [7]. The framework likes



1. Content

In this report, two files will be added.



Extension.rb will be used to store plugins function. API(s) offers the functions plugin needs. Both of them can keep original code clean. Meanwhile, it will be better to write own manager for this particular project.

API(s)
In earth, two basic APIs are created:
1. `add_tab_info (title, controller, action)`
2. add_secton (name)

These APIs will be used in the plugins, use_usage and radial, later.

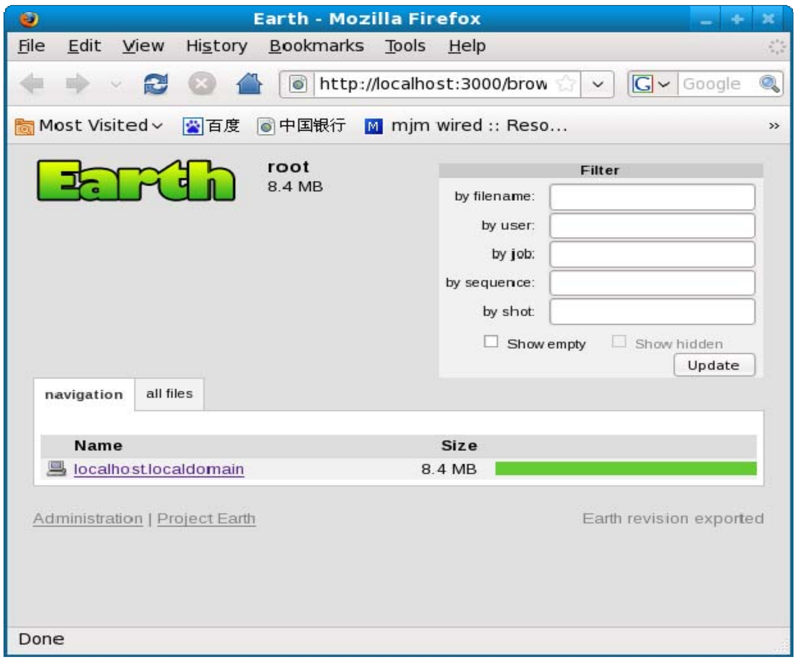
Plugins
In this report, there are two kinds of plugins. One is to create new thing, which means that view, controller, model, and relative things are new. The other is to create new object. It means to extend already things’ functionality.

Plugins radial is the former one, and use_usage is later.

Management
Use own management function instead of Rails Plugins’. It can be put into the daemon, or set as part of Gui. Moreover, new management does not need to be tied with rails so that it can treat plugins as developers expect. Management will search decided folder, and list the whole files. If the file is plugin or part of plugin, it will begin to do relative operations. Most of them are IO operations.

2. How does it work?

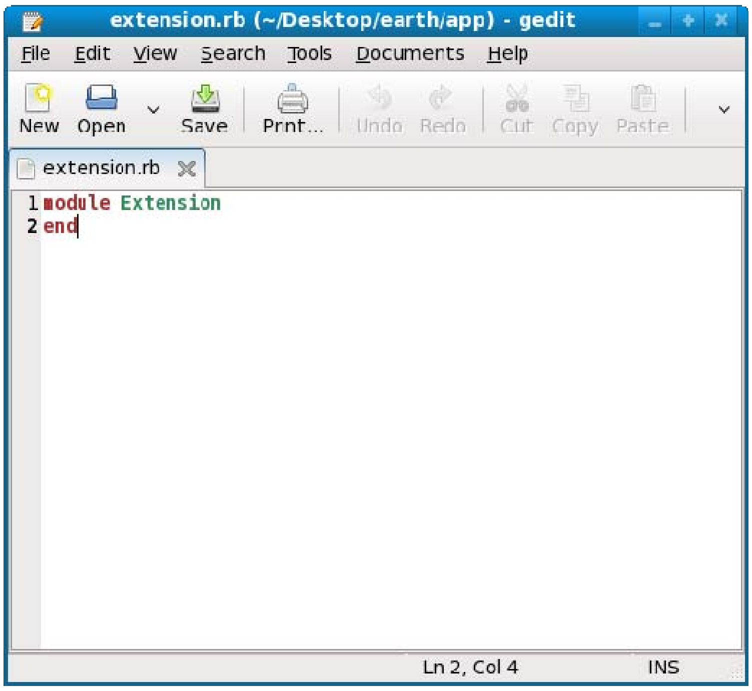
Before adding plugins, the original webpage and extension.rb file look like these. See picture 1 and 2.



root
8.4 MB

Name	Size
localhost.localdomain	8.4 MB

(Picture 1: Original web pages)


extension.rb (~/Desktop/earth/app) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste
extension.rb
Ln 2, Col 4
INS

```
module Extension
end
```

(Picture 2: Original extension.rb file)

Plugin 1 - use_usage

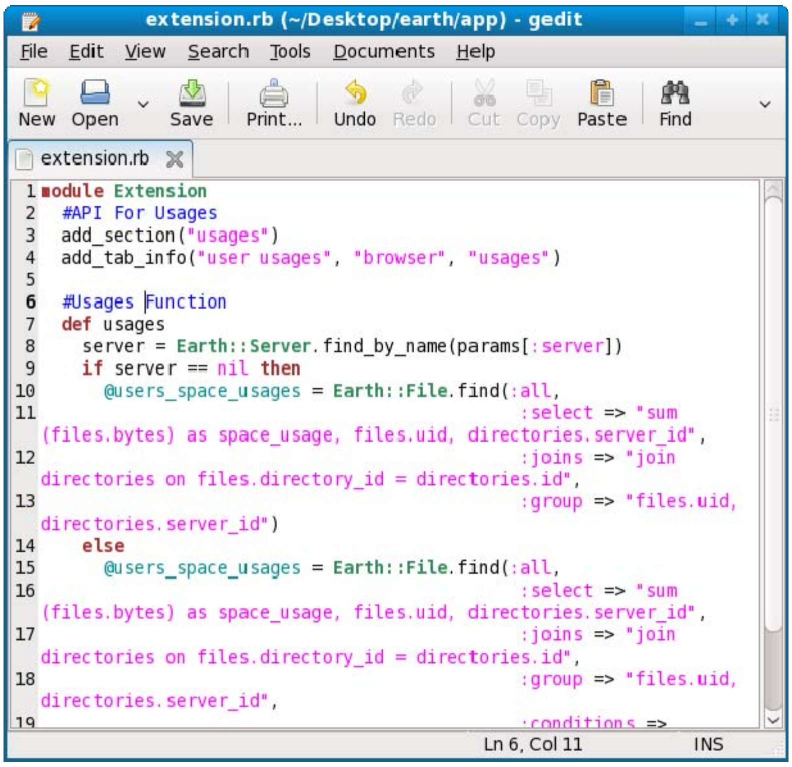
Use_usage will get users’ space usage. It will be treated as part of existing controller file. In this plugin file, there are two APIs, and one def sentence, which is get users’ space usage:

API: add_section ("usages")

API: add_tab_info ("user usages", "browser", "usages")

Def: usages

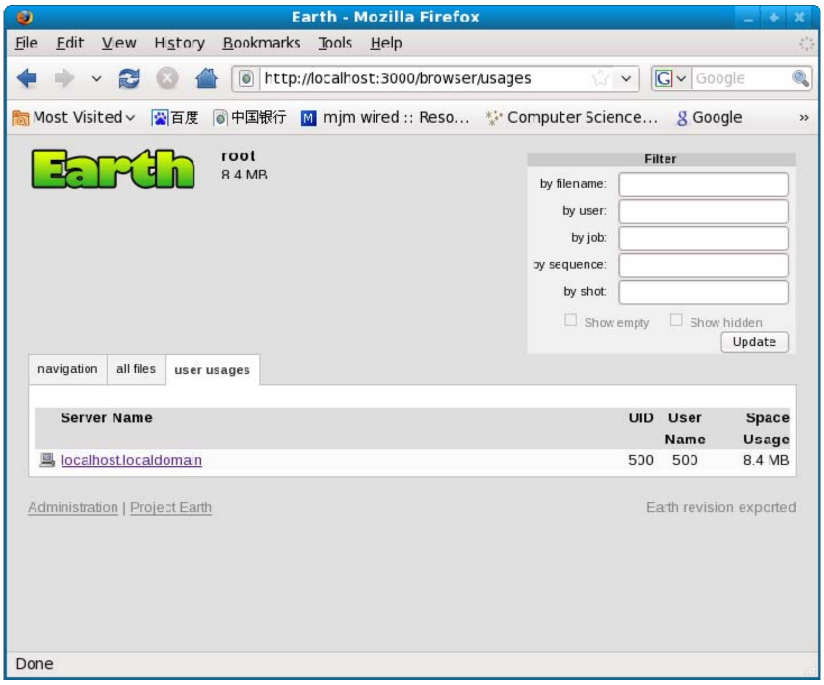
Management will take this plugin, put these three function into extension.rb. So, when web system starts, system will execute APIs first to install a new tab. After that, when user clicks this tab, the def usages will be triggered to send user's space usages directly. See picture 3 and 4.


extension.rb (~/Desktop/earth/app) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find
extension.rb
Ln 6, Col 11 INS

```
module Extension
  #API For Usages
  add_section("usages")
  add_tab_info("user usages", "browser", "usages")

  #Usages Function
  def usages
    server = Earth::Server.find_by_name(params[:server])
    if server == nil then
      @users_space_usages = Earth::File.find(:all,
                                             :select => "sum
(files.bytes) as space_usage, files.uid, directories.server_id",
                                             :joins => "join
directories on files.directory_id = directories.id",
                                             :group => "files.uid,
directories.server_id")
    else
      @users_space_usages = Earth::File.find(:all,
                                             :select => "sum
(files.bytes) as space_usage, files.uid, directories.server_id",
                                             :joins => "join
directories on files.directory_id = directories.id",
                                             :group => "files.uid,
directories.server_id",
                                             :conditions =>
```

(Picture 3: extension file for user usage)



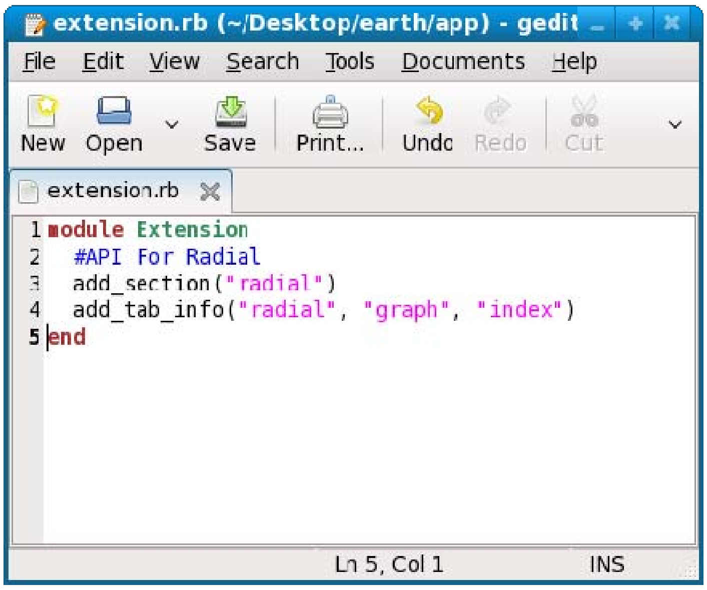
Server Name	UID	User Name	Space Usage
localhost.localdomain	500	500	8.4 MB

(Picture 4: Result for adding plugin user usage)

Plugin 2 - radial
Radial will create a new page to show more detail of user space. And, of course, there is a link to the web service, which is tab.

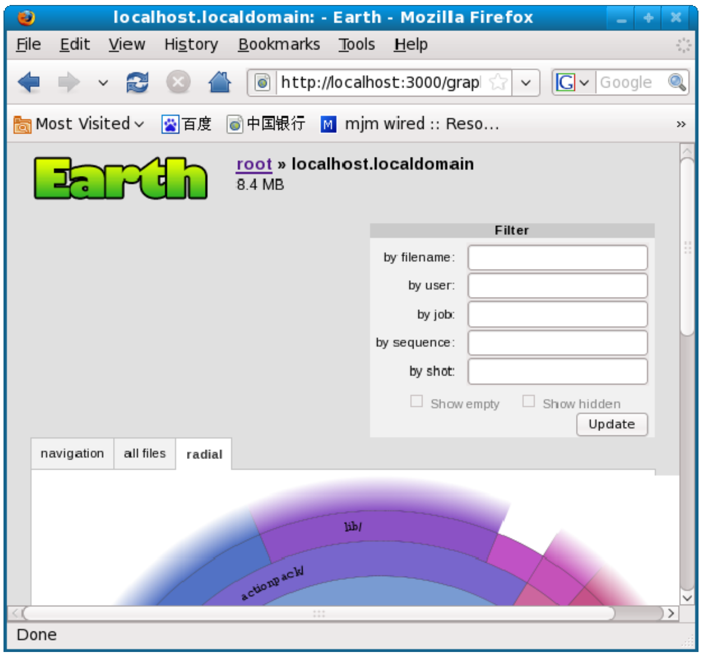
View, Controller and Model should be created separately. It likes Rails Plugins. At same time, an extra file, radial.rb, is needed to use APIs to set up new tab.

When running web system, management will take APIs from radial.rb into extension.rb, send other files into relative folders and store track into a log file, which will be used when uninstall the plugin. See picture 5 and 6


extension.rb (~/Desktop/earth/app) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut
extension.rb
Ln 5, Col 1
INS

```
module Extension
  #API For Radial
  add_section("radial")
  add_tab_info("radial", "graph", "index")
end
```

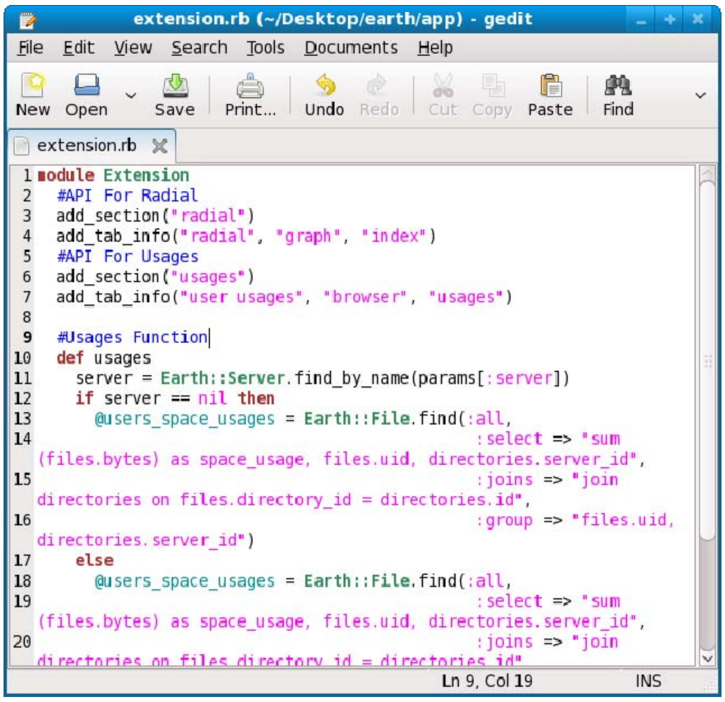
(Picture 5: extension file for radial)


localhost.localdomain: - Earth - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:3000/grap
Google
Most Visited 百度 中国银行 mjm wired :: Reso...
Earth
root » localhost.localdomain
8.4 MB
Filter
by filename:
by user:
by job:
by sequence:
by shot:
Show empty
Show hidden
Update
navigation
all files
radial
lib/
actionpack/
Done

(Picture 6: Result for adding plugin radial)

Add tow more plugin

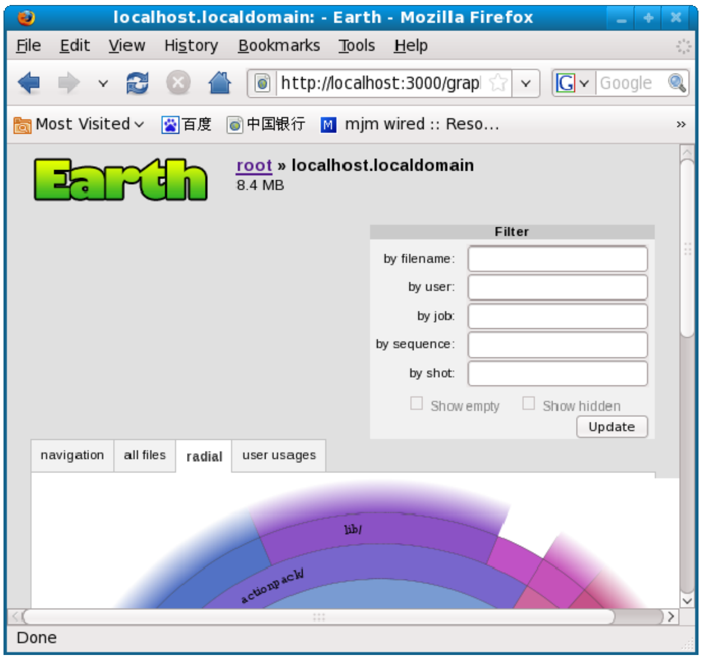
In here, plugin user_usage and radial share the same two APIs. See picture 7 and 8


extension.rb (~/Desktop/earth/app) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find
extension.rb

```
module Extension
  #API For Radial
  add_section("radial")
  add_tab_info("radial", "graph", "index")
  #API For Usages
  add_section("usages")
  add_tab_info("user usages", "browser", "usages")

  #Usages Function
  def usages
    server = Earth::Server.find_by_name(params[:server])
    if server == nil then
      @users_space_usages = Earth::File.find(:all,
                                            :select => "sum(files.bytes) as space_usage, files.uid, directories.server_id",
                                            :joins => "join directories on files.directory_id = directories.id",
                                            :group => "files.uid, directories.server_id")
    else
      @users_space_usages = Earth::File.find(:all,
                                            :select => "sum(files.bytes) as space_usage, files.uid, directories.server_id",
                                            :joins => "join directories on files.directory_id = directories.id"
```

(Picture 7: extension file for user usage and radial)



(Picture 8: Result for adding plugin user usage and radial)

5. (Un)install

Instead of a bit complex Rails Plugins’ way, user just needs to put plugin into plugin folder, if he wants to use it, or remove it, if it is unnecessary.

6. Improvement

So far, the benefits of new way are:

- ✓ Do not need to add / change any code in original code
- ✓ (Un)install plugin easily
- ✓ Manage plugin variedly.
- ✓ Strengthen functions continently

Some people may like to say the plugins created by Rails Plugins are easy to be test, and web service is set up by rails, which could be maintained by Rails Plugins

To be honest, these arguments are hard to defend. Rails Plugins, at first, is used to extend rails core framework, and what this report says is to set up plugin for a web system project. Second, there are enough ways for unit tests, and we also can choose alternative method to achieve:

1. Set management standards and then use Rails Plugins to create and test
2. Each APIs, we use Rails Plugins to build and test, too.
3. Plugins, Rails Plugins also can take them done.

7. Conclusion

Currently, lots of methods can create plugins for variety of aspects. It's hard to say which is better or which is worse. It depends what developers want the system looks like. The aim is easy for users, for developer or for maintainer?

This report shows another way to set up plugin system. It is convenient for others to use.

8. REFERENCES

[1] 'Plug-in (computing)' 2008, *Wikipedia Foundationi, Inc.*, a US <http://en.wikipedia.org/wiki/Plugin>

[2] Coward 2008, 'Ruby on rails plugin', *Phusion Passenger and Ruby Enterprise Edition*, <http://wiki.rubyonrails.org/rails/pages/Plugins>

[3] 'Open Source at Rising Sun Pictures' 2008, *RSS Entries and RSS Comments*, <http://open.rsp.com.au/>

[4] 'HOWTO: Make A Rails Plugin From Scratch' 2008, *Rails forum*, <http://railsforum.com/viewtopic.php?id=682>

[5] Patrick Lenz 2007, *Ruby on Rails*, 1st edn, SitePoint Pty. Ltd, pp323.

[6] Topfunky 2006, The Complete Guide to Rails Plugins: Part I, *RailsMachine*, < http://nubyonrails.com/articles/the-complete-guide-to-rails-plugins-part-i >

[7] Rob Orsini 2007, *Rails Cookbook*, 1st edn, O'Reilly, chapter 2.