**Red Team**

**Stargazer**

**Automatic Telescope Control System**

**Developer’s Guide**

**Team Members:**

Jason Dunscombe

Rob Grmek

Robert Smith

**Instructor:**

**Youry Khmelevsky**

**Course:**

**COSC 470**

**Date:**

**November 13th, 2009**

# ****Revision History****

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/21/2009 | 1.0 | Document created; have draft of:   * Architecture * Use cases * Configuration Management * Installation Guide * Appendix: Glossary * Appendix: References | Robert, Rob |
| 10/28/2009 | 1.1 | Added more installation guide information. | Robert, Rob, Jason |
| 11/08/2009 | 1.2 | Added more installation guide information, Git/SCM information, and architectural diagram. | Rob |
| 11/13/2009 | 1.3 | Added class diagram, updated resources and use case diagram. | Rob |

# ****Table of Contents****

**Page**

|  |  |
| --- | --- |
| Introduction | 1 |
| Installation Guide | 2 |
| Configuration Management | 4 |
| Architecture | 6 |
| Use Case: Schedule Telescope Position | 10 |
| Use Case: Compile Composite Image | 12 |
| Use Case: Notify Operator of New Objects | 13 |
| Appendix A: Glossary | 14 |
| Appendix B: References | 15 |
| Appendix C: Using Git | 17 |
| Appendix D: Using Ruby on Rails | 21 |

# ****Developer’s Guide****

In this document contains the information for:

* System installation
* System backup and recovery
* System architecture
* Coding standards
* Versioning control system access

**Virtual Server Information:**

Windows Server 2003 R2:

|  |  |
| --- | --- |
| IP Address: | 10.1.144.60 |
| Name: | CIS470STAR09 |

Fedora 11: (For developing the Ruby on Rails web system which handles user input)

|  |  |
| --- | --- |
| IP Address: | 10.1.144.61 |
| Name: | CIS470STAR09F |

Windows XP: (For developing the Java web application for controlling the telescope)

|  |  |
| --- | --- |
| IP Address: | 10.1.144.62 |
| Username: | ocstudent |
| Password: | stargazer09 |

# Installation Guide

The following guides and references can be utilized to:

* Install the developmental tools and software needed to make modifications.
* Install the web system.
* Backing up the web system.
* Recovering the web system.

#### Setting-up Development Environment: (For the Web System using Fedora 11)

Reference used: <http://www.technetra.com/2009/04/22/howto-setting-up-ruby-on-rails-for-fedora-10-and-11/>

See if tools are installed:

which gcc make git

Installing SQLite: (the development database)

sudo yum install sqlite sqlite-devel

Installing MySQL (optional): (another database option)

sudo yum install mysql-server mysql-libs mysql-devel

Installing Ruby: (the language)

sudo yum install ruby ruby-devel ruby-libs ruby-mode ruby-rdoc ruby-irb ruby-ri ruby-docs ruby-mysql ruby-sqlite3

Installing Ruby Gems: (packaging system for Ruby libraries)

wget -q http://rubyforge.org/frs/download.php/55066/rubygems-1.3.2.tgz

tar xzf rubygems-1.3.2.tgz

cd rubygems-1.3.2

sudo ruby setup.rb

Installing Ruby on Rails: (the framework)

sudo gem install rails

Installing Mongrel: (the development web server)

sudo gem install mongrel mongrel\_cluster

Installing JSON: (XML alternative)

sudo gem install json

#### Installing SQLite-Ruby gem:

sudo gem install sqlite3-ruby

#### Installing Rake gem:

sudo gem install rake

Installing Hirb (displays table data in console better):

sudo gem install hirb

Installing nifty-generators (for layouts):  
sudo gem install nifty-generators

For authentication/login subsystem:

sudo gem install authlogic

For time validation:

sudo gem install validates\_timeliness

For automatically creating class diagrams from Rails applications:

sudo gem install railroad

If you’re missing any dependencies run:

sudo rake gem:install

#### Installing the Web System:

This can simply be done using the Git clone command to copy the GitHub repository.

This will also create the directory that will hold the files.

So first, go to a folder location, in this case the user’s home directory.

cd ~

Issue the Git clone command:

git clone [git@github.com:RedTeamCOSC470/Stargazer.git](mailto:git@github.com:RedTeamCOSC470/Stargazer.git)

Now all the project files will be retrieved and put into the newly created directory: ~/Stargazer/

Then, create the database, run the database migration and start the web server.

#### System Backup:

System backup is achieved through regular commits both locally and to the remote GitHub repository.

See Appendix C for instructions for using Git.

#### System Recovery:

System recovery is achieved from pulling from the GitHub repository.

See Appendix C for instructions for using Git.

# Configuration Management

The purpose of this configuration management document is to mention versioning control methods, naming conventions, coding conventions, and methods for system backup and recovery.

#### Versioning Control:

All project files will be protected through the versioning control system ‘Git’.

All web application project files will be uploaded to the ‘Stargazer’ directory.

All documentation files will be uploaded to the ‘Documentation’ directory.

By using Git we avoid several problems such as team members simultaneously updating files as well as errors resulting in loss of data or system failure. In this case an older version would need to be retrieved and used.

Tutorials for using Git can be found in Appendix C.

#### Github Account:

Login at: <http://github.com/>

Username: RedTeamCOSC470

Password: stargazer09

#### Web Application Repository Information:

Name: Stargazer

Public Clone URL: <git://github.com/RedTeamCOSC470/Stargazer.git>

My Clone URL: [git@github.com:RedTeamCOSC470/Stargazer.git](http://github.com/RedTeamCOSC470/git@github.com:RedTeamCOSC470/Stargazer.git)

#### Documentation Repository Information:

Name: Documentation

Public Clone URL: [git://github.com/RedTeamCOSC470/Documentation.git](git://github.com/RedTeamCOSC470/Stargazer.git)

My Clone URL: [git@github.com:RedTeamCOSC470/Documentation.git](mailto:git@github.com:RedTeamCOSC470/Documentation.git)

#### Setting up Git and using Git:

See Appendix C.

#### Naming Conventions:

Each document that is created is named:

1. Prefix starting with ‘Stargazer’.
2. No spaces are used, underscores are used instead.
3. Only alphabetic characters and underscores are used.
4. Book title capitalization is used.

An example of a document name: Stargazer\_Vision.doc

#### Coding Standards:

The following code conventions are used:

Don’t use tab indenting, instead use 2 spaces.

Line length: maximum 80 characters.

Wrapping lines: break after comma and break after.

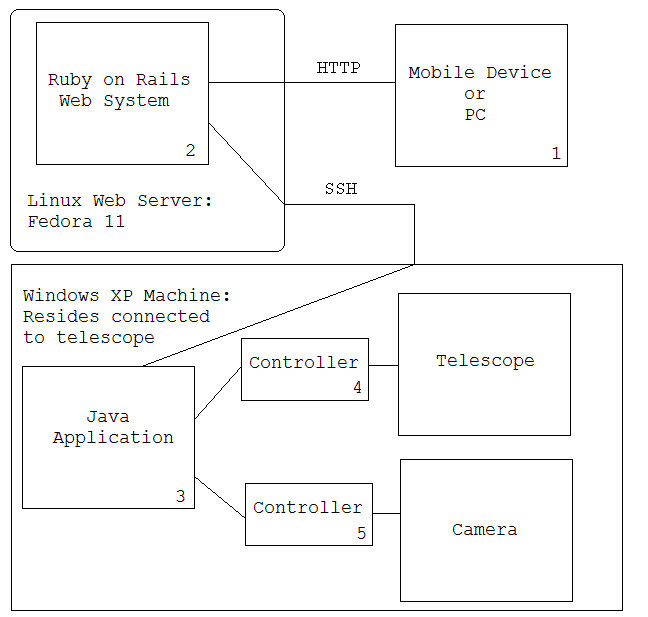
|  |  |  |
| --- | --- | --- |
| Syntax Type | Correct | Incorrect |
| Method Definitions | my\_method(my\_arg) | my\_method( my\_arg ) my\_method my\_arg |
| Boolean Testing | && or || | AND or OR |

# Architecture

The system has three main functional components:

* scheduling the telescope to orient itself given the coordinates, location and date entered by the operator
* processing the images taken by the camera to compose a larger, high resolution image so it can compare the previous images with the new image
* if new objects are found, to notify the operator

### Architecture Diagram:



Legend:

1. Mobile Device and PC access:

Allows users to connect to the Linux web server

* + Using a web browser, users will login to the web application and then be able to schedule telescope positions
  + Also, they can see low resolution composite images (approx 120x320 pixels)

1. Ruby on Rails web application:
   * Operating system: Fedora 11
   * The web server handles client requests from user devices (#1)
   * Has authentication; users will need to login in order to access the web application’s functionality
   * Will allow users to schedule telescope position; will display a form with the necessary inputs
   * Will allow users to review scheduling logs to see previous history or make changes to future schedules
   * Will display latest composite images that have been captured.
   * Handles scheduling and jobs; once the next time occurs to move the telescope, will make a connection to the Java application (#3) and run the program as a command line argument to move the telescope.
   * Handles email notifications.
   * Receives composite images from the Java application (#3).
2. Java program:
   * Operating system: Windows XP
   * Hosted on the same computer which is connected to the telescope
   * Handles actual communication with the telescope and camera through their respective drivers. This allows for example: moving the telescope, taking pictures, etc.
   * Handles image processing; will compile the many hundred smaller images into a large composite image
   * Handles image recognition; checks composite images with libraries to see if new objects have been identified
3. Telescope controller:
   * Has drivers which allows for the Java program (#3) to communicate with the telescope.
4. Camera controller:
   * Has drivers which allows for the Java program (#3) to communicate with the camera.

#### Hardware Components:

|  |  |
| --- | --- |
| Camera | Sony a900 DSLR |
| Telescope | Meade ETX-60AT-TC |

#### Development Environment: Web Server

|  |  |
| --- | --- |
| Operating System | Fedora 11 |
| Development Framework | Ruby on Rails |
| Unit Testing | Ruby’s built-in Unit Tests |
| Text Editor | GEdit, E, Emacs, VIM |
| Versioning Control System | Git |
| Web Server | Production: Apache  Development: Mongrel |
| Database | Production: PostgreSQL  Development: SQLite |

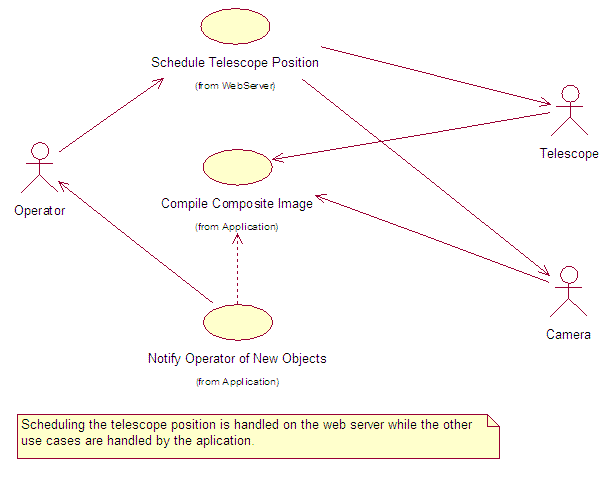
### Domain Model:

### C:\Documents and Settings\Robert Grmek\rails\documentation\Stargazer_Domain.gif

### Class Diagram:

### C:\Documents and Settings\Robert Grmek\rails\Stargazer\diagrams\class_diagram.png

### Use-Case Diagram:



# Use Case: ****Schedule Telescope Position****

#### Brief Description:

This use case allows the operator to program the telescope with one or more specific times and dates and corresponding locations for observation. The system will instruct the telescope to point to the coordinates of that location at the scheduled time. The camera will then be instructed to take pictures of that area of the sky.

The main actor in this use case is the operator.

#### Flow of Events:

The use case begins after the operator has logged into the web site.

Basic Flow:

1. The operator selects the Program option from the Operator Control page of the web site.
2. The operator is presented with an input form for entering observation information.
3. The operator enters a date in the date text field.
4. The operator enters a time in the time text field.
5. The operator enters a location in the location text field.
6. The operator clicks the submit button to submit the observation information to the system.
7. The web site shows a confirmation message that the information was entered as well as the input form so they can continue entering other observation information.

#### Special Requirements:

No special requirements have been specified for this use case at this time.

#### Pre-Conditions:

1. The operator is logged in:

Before this use case begins the operator has logged onto the web site.

#### Post-Conditions:

1. The telescope is told to move to the specified coordinates on the date provided by the operator.
2. The camera begins taking pictures through the telescope once the telescope is in position.

#### Extension Points:

There are no extension points associated with this use case.

# ****Use Case: Compile Composite Image****

#### Brief Description:

This use case allows the images captured from the camera to be compiled into a composite image with a larger resolution. The images are to be processed in an attempt to find new objects. If any new objects are found, the operator is notified.

The main actor in this use case is the camera.

#### Flow of Events:

The use case begins after the camera has finished taking a set of pictures.

Basic Flow:

1. The pictures are stored on the hard drive of the server.
2. All pictures are combined together using an algorithm to produce a higher resolution picture.
3. The composite, high resolution picture is then compared to older pictures from similar locations to identify any previously unrecorded objects.
4. The high resolution composite image is stored with the original images for later review.

#### Special Requirements:

No special requirements have been specified for this use case at this time.

#### Pre-Conditions:

1. The camera has finished taking pictures:

Before this use case begins the camera must have completed all pictures for the specified observation period.

#### Post-Conditions:

There are no post-conditions associated with this use case.

#### Extension Points:

1. If new object(s) are found:

If a new object is found after image processing, then the operator is sent a notification. (See ‘Use Case: Notifies Operator of New Objects’).

# ****Use Case: Notify Operator of New Objects****

#### Brief Description:

This use case informs the operator of any new objects detected while processing the images in the Compiles Composite Image use case.

The main actor in this use case is the operator.

#### Flow of Events:

The use case begins after a new object has been detected while compiling the composite image.

Basic Flow:

1. The high resolution picture containing the previously unidentified object is stored on the hard drive of the server.
2. A notice is placed on the web site alerting the operator of the new object.
3. The picture of the object is shown to the operator as well as some indicator of where the new object is located.
4. The user clicks the save button to store the high resolution image to a separate location.

#### Special Requirements:

No special requirements have been specified for this use case at this time.

#### Pre-Conditions:

1. The system has found an unidentified object in a picture:

Before this use case begins an unidentified object must have been identified by the system.

#### Post-Conditions:

There are no post-conditions associated with this use case.

#### Extension Points:

There are no extension points associated with this use case.

# Appendix A: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Git | The versioning control system the project team uses. |
| Ruby on Rails | Framework used for rapid web development of a dynamic web system. |
| Stargazer | Internal project name for the automated telescope control system. |

# Appendix B: References

For setting up a Ruby on Rails development environment on Fedora 11, see here:

Technetra:

<http://www.technetra.com/2009/04/22/howto-setting-up-ruby-on-rails-for-fedora-10-and-11/>

For setting up a Ruby on Rails development environment on Windows, see here:

Akitaonrails – Setting up the Best Rails Environment on Windows:

<http://www.akitaonrails.com/2009/1/13/the-best-environment-for-rails-on-windows>

O’Reilly - Rolling with Ruby on Rails: (another installation guide)

<http://oreilly.com/ruby/archive/rails.html>

Oracle guide for installing Rails with Oracle with a practice tutorial for creating a simple application:

<http://www.oracle.com/technology/pub/articles/haefel-oracle-ruby.html>

Using Git on Windows:

<http://beans.seartipy.com/2008/12/09/setting-up-ruby-on-rails-projects-with-git-and-github/>

Other resources:

Official Ruby on Rails guides:

<http://guides.rubyonrails.org>

Tutorials Point guides:

<http://www.tutorialspoint.com/ruby-on-rails-2.1/index.htm>

Rails Video Tutorials:

<http://railscasts.com>

<http://asciicasts.com/>

Distributing Ruby Applications:

<http://www.erikveen.dds.nl/distributingrubyapplications/rails.html>

Git and Rails:

<http://harryseldon.thinkosphere.com/2009/01/14/git-and-rails-a-detailed-tutorial-including-plugins-submodules-development-and-production>

Validates\_Timeliness Plugin Information:

<http://www.railslodge.com/plugins/1160-validates-timeliness>

Authlogic Plugin Tutorial:

* <http://github.com/binarylogic/authlogic_example>
* <http://asciicasts.com/episodes/160-authlogic>

Railroad Plugin Information:

<http://www.railroad.rubyforge.org/>

Understanding the Ruby language:

* <http://www.robertsosinski.com/2008/12/21/understanding-ruby-blocks-procs-and-lambdas/>
* <http://rubylearning.com/satishtalim/tutorial.html>
* <http://www.meshplex.org/wiki/Ruby/Ruby_on_Rails_programming_tutorials>
* <http://www-users.math.umd.edu/~dcarrera/ruby/0.3/chp_01/first_steps.html>
* <http://www.tekniqal.com/>

# Appendix C: Using Git

This set up takes place after a Github account has been created and a repository has been created. Currently the information for the account and repository are:

#### Github Account:

Login at: <http://github.com/>

Username: RedTeamCOSC470

Password: stargazer09

#### Repository Information:

Name: Stargazer

Public Clone URL: <git://github.com/RedTeamCOSC470/Stargazer.git>

My Clone URL: [git@github.com:RedTeamCOSC470/Stargazer.git](http://github.com/RedTeamCOSC470/git@github.com:RedTeamCOSC470/Stargazer.git)

Information for setting up Git: (taken from GitHub)

#### Global setup:

* Initial setup of Git on a computer.

|  |
| --- |
| # First, download and install Git (see installation guide)  # Add configuration information:  git config --global user.name "Your Name"  git config --global user.email "Your Email"  # Then, add your public key (see below) |

#### Adding a public key:

* A public key needs to be added to validate the computer as the owner of the repository so commits can be made.

|  |
| --- |
| # Generate a public key:  ssh-keygen  # Use the ‘cat’ command on the file that was created in  # directory such as:  cat ~/.ssh/id\_rsa.pub  # Then, add the public key to the github account under:  # Account Settings -> SSH Public Keys |

#### Cloning the Repository:

* Do this to recreate the directory structure with all project files.
* Do this if the local Git repository has not already been made.
* Can be used after the development environment is setup and freshly installed.

|  |
| --- |
| git clone [git@github.com:RedTeamCOSC470/Stargazer.git](mailto:git@github.com:RedTeamCOSC470/Stargazer.git) |

#### Other commands:

|  |
| --- |
| # create a new local repository:  git init    # add a file to the staging area:  # in other words, to setup file(s) before a local commit  git add [filename]  # to add all files use this:  git add .  # check status of the staging area files:  git status  # commit the staging area files to the local repository  # using the –m switch includes a message:  git commit –m “This is a commit message”  # show commits:  git log  # change username for only the local git repository:  git config user.name “[User Name]”  # adding a new remote destination called “origin”:  # a remote destination in this case is our  # GitHub repository  git remote add origin [git@github.com:RedTeamCOSC470/Stargazer.git](mailto:git@github.com:RedTeamCOSC470/Stargazer.git)  # list remote destinations:  git remote  # pushing the local committed files to the remote destination  # from the master branch “origin”:  # in other words, this is to commit the files to the  # GitHub repository.  git push origin master  # check if the local commit is not already pushed to  # the remote destination:  git log --pretty=oneline master...origin/master  # update the local GitHub origin master branch:  git fetch origin  # merge GitHub’s remote changes into the local master branch:  git pull origin master  # in the project root, create a file for Git to use  # to ignore certain files:  vim .gitinore  # then write the filename into the file  [filename]\*  :wq!  #then add the file  git add .gitignore  # see changes since files have last been stages:  # (file is added, but not committed)  git diff  # remove any changes in the working directory:  git checkout -- .  # unstage the file, but still have changes to the file:  git reset HEAD [filename]  # start from the last commit; remove everything:  git reset -hard    # create a branch:  git branch [branchname]  # list branches:  git branch  # checkout 1 commit back  git checkout HEAD^  git checkout HEAD~1  # 2 commit backs, etc  git checkout HEAD^^  git checkout HEAD~2  # go back to master  git checkout master  # use gitignore command to ignore some files  touch .gitignore  # add files you wish git to ignore (i.e. doc/api)  doc/api  # create a branch and checkout that branch  git checkout –b manage\_users  # or  git branch manage\_users  git checkout manage\_users    # merge the changes to master branch and delete old branch  git checkout master  git merge manage\_users  git branch –D manage\_users |

# Appendix D: Using Ruby on Rails

Run the rails command to create the initial files:

rails stargazer

Create the database:

rake db:create

Start the Mongrel web server:

script/server

To see if all gems have been installed:

rake gem:install

To list scripts:

script/generate

To drop tables:

rake db:version

rake db:migrate:down:version

To create the tables:

rake db:migrate

To generate documentation:

rake doc:app

To run tests:

rake test

rake test:units

rake test:functionals

#### Using Plugins: Annotate

This plugin will annotate models and unit tests with the table data for that object.

To run simply issue the command:

annotate

#### Using Plugins: yUMLmeRails

This plugin will create a class diagram from the models in the project directory. For help, see: <http://github.com/nelsonsilva/yUMLmeRails/>

To run simply issue the command:

Rake yUMLmeRails:download

#### Using Plugins: RailRoad

To create modeling diagrams using RailRoad: (see: <http://railroad.rubyforge.org/>)

Note: Run RailRoad on the Rails application's root directory.

Produces a models diagram to the file '**models.dot**':

rake doc:diagrams

Controller diagram in **PNG** format:

railroad -C | neato -Tpng > controllers.png

Model diagram in **PNG** format:

railroad -M | dot -Tpng > models.png