READ-ONLY: This project has been archived. For more information see this post.

errfixExamples

Updated Dec 29, 2008 by peterjoh...@gmail.com

Introduction to errfix with examples

errfix creates a software model of your system, from a simple CSV file or using its built in API. errfix lets you create *Random Walks* across your model, moving from State to State by executing actions.

Example 1: Random Walk using a CSV defined model (Google Homepages Example)

This example script creates a model and outputs a text description of a random walk. (It doesn't run any tests.)

CSV data

The CSV file contents should be:

Start/End	WEB	IMAGES	MAPS	SHOPPING	MAIL
WEB		click_images	click_maps	click_shopping	click_mail
IMAGES	click_web		click_maps	click_shopping	click_mail
MAPS	click_web	click_images		click_shopping	click_mail
SHOPPING	click_web	click_images	click_maps		click_mail
MAIL	click_web				

You can download this CSV file here.

Source Code

The example errfix Google Searches code is:

```ruby
# errfix HelloWorld example.
#
require 'rubygems'
require 'errfix'

# Create model, Load in the CSV file with our Google Searches model
smc=StateModelCreator.new
google_searches = smc.load_table 'google_searches.csv'

# Write the model's graph to the filesystem (PostScript format)
google_searches.create_dot_graph.output("google_searches.dot")

# Create a random walk across the model.
# Remember its random, so each time you run it, it might give you a different path...
google_walk_1 = google_searches.random_walk("WEB")

# Step Through the walk.
google_walk_1.steps.each do |a_transition|
  puts "Step: #{a_transition}"
end # end the walk
```

You can download errfix_google_eg.rb here.

How to run the code

Place the above 2 files in the same folder, open a UNIX Terminal or DOS prompt in the same folder and type:

```
ruby errfix_google_eg.rb
```

The output will look something like this:

```
Step:WEB,click_images => IMAGES
Step:IMAGES,click_web => WEB
Step:WEB,click_shopping => SHOPPING
Step:SHOPPING,click_web => WEB
Step:WEB,click_maps => MAPS
Step:MAPS,click_mail => MAIL
Step:MAIL,click_web => WEB
Step:WEB,click_mail => MAIL
Step:MAIL,click_web => WEB
Step:WEB,click_shopping => SHOPPING
Step:SHOPPING,click_web => WEB
Step:WEB,click_images => IMAGES
Step:IMAGES,click_web => WEB
Step:WEB,click_maps => MAPS
Step:MAPS,click_images => IMAGES
Step:IMAGES,click_maps => MAPS
Step:MAPS,click_mail => MAIL
Step:MAIL,click_web => WEB
Step:WEB,click_images => IMAGES
Step:IMAGES,click_maps => MAPS
```

If you run the script again, you'll probably get a slightly different output. Thats because the script is generating *random* walks, therefore each time it will likely make a different random choice of steps.

If all is well you should get a DOT file placed in the same folder called google_searches.dot DOT files can be viewed using GraphViz. The model when rendered in Graphviz should look similar to this:

Example 2: Random Walk Driving a (Fake) System Under Test (Google Homepages Test Driver Example)

Once you have created a random walk, you can the use this walk to drive your test system using the Walk#drive_using method. Pass in your Test driver code and allow the Walk to drive your System under test.

Source Code

First lets print out the coverage statistics for this walk:

```
puts "State Coverage: #{google_walk_1.state_coverage}"
puts "Transition Coverage: #{google_walk_1.transition_coverage}"
```

Now lets create a simple System Under Test (SUT). This could be a class you are testing, but also a driver to the SUT. Typically this driver would direct your test tool, Selenium RC or WATiR.

```
class SystemUnderTestDriver
  # Action Methods that errfix will look for...
  def click_web;      puts "action: click_web"; end # end method
  def click_images;   puts "action: click_images"; end # end method
  def click_maps;     puts "action: click_maps"; end # end method
  def click_shopping; puts "action: click_shopping"; end # end method
  def click_mail;     puts "action: click_mail"; end # end method
  # Include these methods to verify state of the SUT
  def test_WEB;       puts "testing: WEB"; end # end method
  def test_IMAGES;    puts "testing: IMAGES"; end # end method
  def test_MAPS;      puts "testing: MAPS"; end # end method
  def test_SHOPPING;  puts "testing: SHOPPING"; end # end method
  def test_MAIL;      puts "testing: MAIL"; end # end method
end # class under test
```

The above driver code would be required normally, even when not using errfix. We've created it here to allow you to see how errfix might interact with your system.

errfix can now easily drive through your model, along the path of the Random Walk. errfix will call methods with the same name as your actions (in the CSV file). It will also call a method called test_STATE-NAME after each action. These 'test*' methods can be used to execute tests, ensuring your SUT has correctly reached a new State.*

```
google_walk_1.drive_using(SystemUnderTestDriver.new)
```

You can download the code here.

How to run the code

Place the above file (and the CSV file mentioned above) in the same folder, open a UNIX Terminal or DOS prompt in the same folder and type:

```
ruby errfix_google_eg_2.rb
```

The output will look something like this:

```
State Coverage: 100.0
Transition Coverage: 64.7058823529412

Lets get our Random Walk to drive our SUT or at least our SUT driver code...
testing: WEB
action: click_images
testing: IMAGES
action: click_mail
testing: MAIL
action: click_web
testing: WEB
action: click_maps
testing: MAPS
action: click_web
testing: WEB
...
```

Example 3: Guarded actions using the errfix DSL (Foobar Media EFSM Example)

Our hypothetical media organisation *FooBar Media Intl*, wants to model its online media system. Like all good companies they are developing iteratively. (At least thats there reason for only having two pieces of content). An errfix generated graph can be seen here:

The above graph is created automatically in DOT format, by errfix.

Source Code

The code below first defines all the 'Actions'. One of the actions ':click_log_in' sets a variable '@logged_in' to *true*:

```ruby
require 'rubygems'
require 'errfix'

# Create a FSM and draw a graph
#
smc = StateModelCreator.new

# The Actions...
smc.define_action :click_home
smc.define_action :view_content
smc.define_action :show_more
smc.define_action :click_log_in do
  @logged_in=true
end # end action
```

We then define 'Guards', these stop the actions from being *actioned* or completed unless the conditions have been met:

```ruby
# The Guards...
smc.define_guard_on :view_content do
  # You can only view content if you are logged in.
  if @logged_in
    guard=true
  else
    guard=false
  end # end if else
  guard
end # end guard

smc.define_guard_on :click_log_in do
  # You can't log in if you are already logged in.
  if @logged_in
    guard=false
  else
    guard=true
  end # end if else
  guard
end # end guard
```

Now we just need to state where these Actions (and their Guards), will be used. For this we attach the transition, naming the start and end states as well as what action connects them:

```
smc.attach_transition(:HOME,:view_content,:SHOWING_CONTENT)
smc.attach_transition(:SHOWING_CONTENT,:show_more,:MORE_CONTENT)
smc.attach_transition(:HOME,:click_log_in,:LOG_IN_COMPLETE)
smc.attach_transition(:LOG_IN_COMPLETE,:click_home,:HOME)

my_efsm = smc..state_machine
```

Finally, above, we get a copy of the completed State Machine.

```
my_efsm.create_dot_graph.output("my_fsm_graph.dot")
```

...and then output the graph in DOT format.

What if I want to output a random walk across this model?

```
puts my_efsm.random_walk(:HOME)
```

Well thanks to the guards, a walk starting at :HOME can only take one path.: HOME, click_log_in => LOG_IN_COMPLETE, click_home => HOME, view_content => SHOWING_CONTENT, show_more => MORE_CONTENT, MORE_CONTENT

You can download this code in full.