Chair of
Software Engineering

“Computer Science Event List” Quick-start Guide

A Tutorial to the Eiffel Web Framework

Semester Thesis

By: Peizhu Li

Supervised by: Marco Piccioni

Prof. Bertrand Meyer

Student Number: 02-925-899

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik
Computer Science

This tutorial is a step by step guide for "**The Computer Science Event List**" application implementation, with focus on the usage of "**Eiffel Web Framework**" developed along this semester thesis work.

It's recommended to read the thesis to get familiar with some basic concepts about the Framework before using this tutorial.

The Task

The Computer Science Event List was produced by the Chair of Software Engineering at ETH Zurich as a service to **Informatics Europe**[1], The Research and Education Organization of Computer Science and IT Departments in Europe, as a repository of conferences and other events in computer science, information technology and related fields.

Starting from the current implementation[2], we will re-implement the application here by utilizing the Eiffel Web Framework, as presented in the semester thesis, to demonstrate some typical scenarios in applying the Framework in a real-word Web application development.

Step 1. Requirement Analysis

The current implementation[2], it's a simple but typical web application; it needs to take care of session support, user authentication and form validation. In addition to this, it must provide a set of operations concerning users and events, and generate the corresponding web pages.

Let's assume the following requests will be handled by our application:

Applied on	***As a (role)***	***Requests***
User	Guest	register
	Normal User	register, login, logout, update, display details
	Admin	add, login, logout, display details, edit, suspend, activate, delete
Event	Guest	list, display details
	Normal User	list, display details, submit, list events submitted by himself, edit his own yet not approved event
	Admin	list, display details, add, edit, display details, approve, reject, delete
(request a page)	(all)	display contact, about page

After taking advantage of the USER_MANAGER implementation included in the Web Framework for user related operations, it seems that except some classes for event and event storage related operations, there are not much business logic that needs to be specially addressed on the server-side.

Source codes implemented in all steps are supposed to be found in *tutorial**informatics_events* folder together with the Web Framework package.

Step 2. Application Design and Initial Implementation

We create an INFORMATICS_DISPATCHER that inherits from REQUEST_DISPATCHER class for the dispatching. We can also use a USER_HANDLER, an EVENT_HANDLER, and a GENERAL_HANDLER, focused respectively on processing user, event data and informative

pages, and generating the resulting HTML pages for relevant requests. We can furthermore create an INFORMATICS_USER class inheriting from the USER class in the Framework, but using USER_MANAGER and supplied USER_MANAGER_FILE_IMPL for necessary functionalities and storage. Besides, we need to implement several classes for event related operations and event storage, and a class to wrap up some session related operations for our application.

- INFORMATICS_USER

Based on the information presented in the current implementation[2], we inherit an INFORMATICS_USER class from USER and simply add necessary attributes, respective accessing functions, and a role and a status member, together with a supporting class APPLICATION_CONSTANTS, for a centralized constant definition.

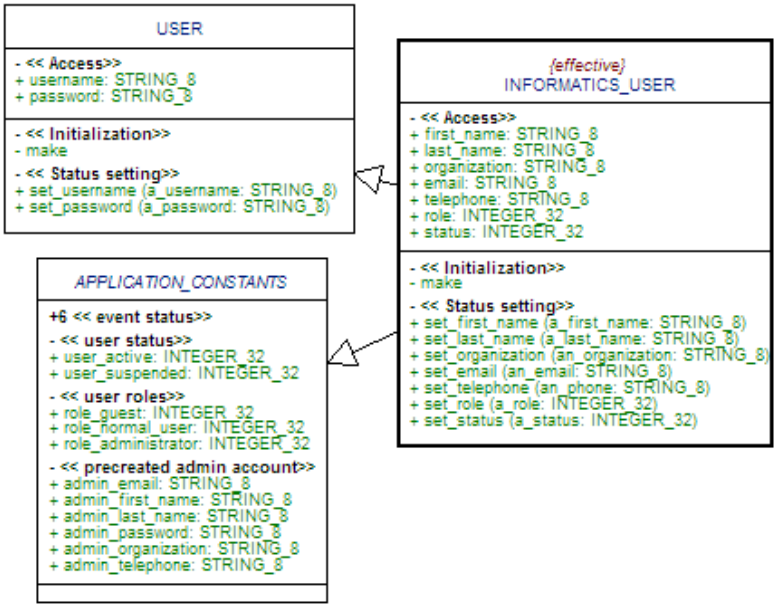

Fig 2.1 INFORMATICS_USER Implementation

Check *informatics_user.e*, *application_constants.e* for implementation details.

- EVENT related classes

We implement an EVENT class for event data, wrap it as an ARRAYED_LIST into an EVENT_LIST class, and then implement EVENT_DAO and EVENT_DAO_FILE_IMPL using file serialization for storage as before. We then use a simple ID_GENERATOR class for event id generating, storing and retrieving. We then add event status to APPLICATION_CONSTANTS.

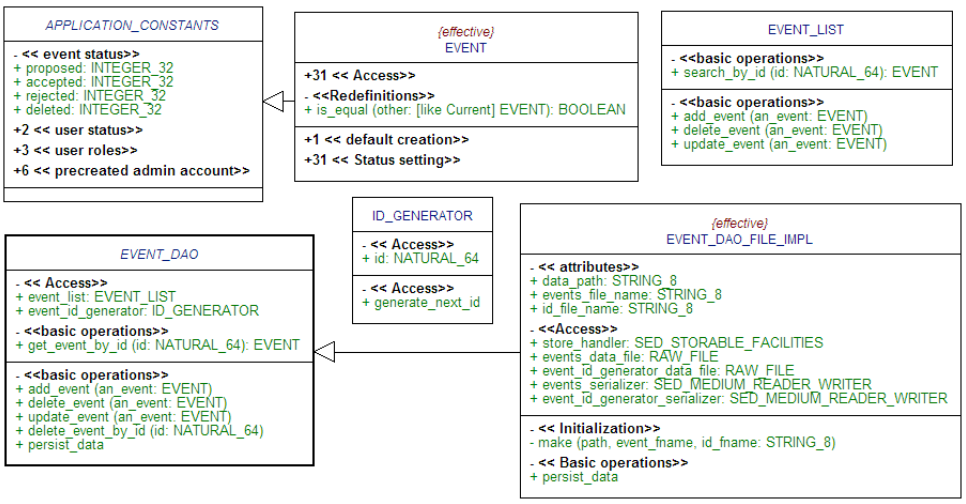

Fig 2.2 EVENT related Implementation

- INFORMATICS_SESSION

To be able to track current user sending requests, a simple solution is to include user information in the actual session object. Here we’d like to include both username and email to be saved in the session, and wrap the relevant operations in an INFORMATICS_SESSION class.

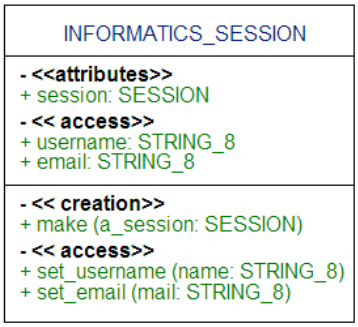

Fig 2.3 INFORMATICS_SESSION

- INFORMATICS_DISPATCHER

Simply inherit INFORMATICS_DISPATCHER from REQUEST_DISPATCHER, and redefine the make creation procedure, including a local variable for each handler type we are planning to use:

```
make is
      -- simply call parent's make procedure, but define a local variable for each
      handler to ensure respective handler classes will be included in the application
   local
      event_handler: EVENT_HANDLER
      user_handler: USER_HANDLER
      general_handler: GENERAL_HANDLER
   do
      PRECURSOR {REQUEST_DISPATCHER}
   End
```

UML diagram together with REQUEST_DISPATCHER as *Fig 2.3*.

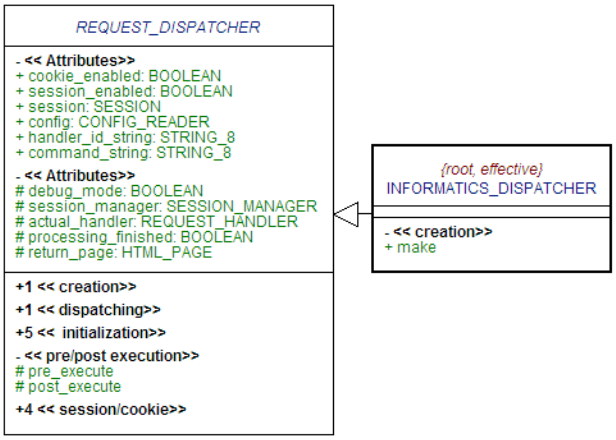

Fig 2.3 INFORMATICS_DISPATCHER

- `Handlers`

We would like to introduce an `INFORMATICS_HANDLER` class as the parent for `USER_HANDLER`, `EVENT_HANDLER` and `GENERAL_HANDLER`, so some common tasks on the `HANDLER` level can be injected there with *`pre_processing`* and *`post_processing`* feature implementation.

An instance of `USER_MANAGER` is included in `INFORMATICS_HANDLER`, together with an `ENCRYPTOR` to enable `USER_MANAGER` encrypting user data on serialization. Also included are an `INFORMATICS_USER` instance, to identify the current user, an `INFORMATICS_SESSION`, and an `HTML_TEMPLATE_VIEW`, because we are planning to produce all result pages based on a set of templates.

As mentioned in the Framework documentation, the initialize feature must be redefined to let include all necessary instantiation routines for a correct initialization after having generated a `HANDLER` with Eiffel reflection.

We include an `EVENT_DAO` variable in `EVENT_HANDLER` and instantiate it with an `EVENT_DAO_FILE_IMPL` instance in *`handling_request`*. Otherwise we could simply implement an empty body of deferred feature *`handling_request`* for `USER_HANDLER` and `GENERAL_HANDLER` (we will extend them later).

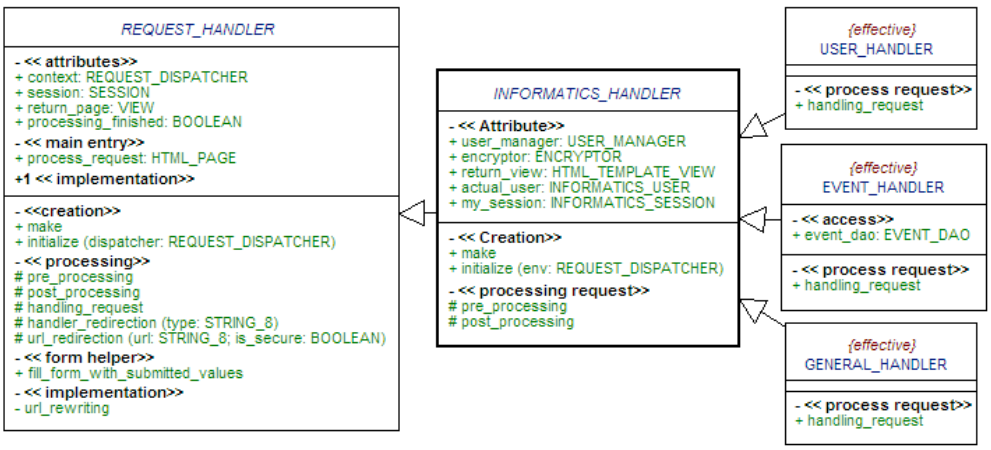

Fig 2.4 HANDLERs

So far we already prepared the framework of our application. You can download *step2.zip* and get it compiled together with the *webex* Web Frame cluster, for a more clear view over this initial implementation we achieved. We will focus on the user related part to continue our implementation.

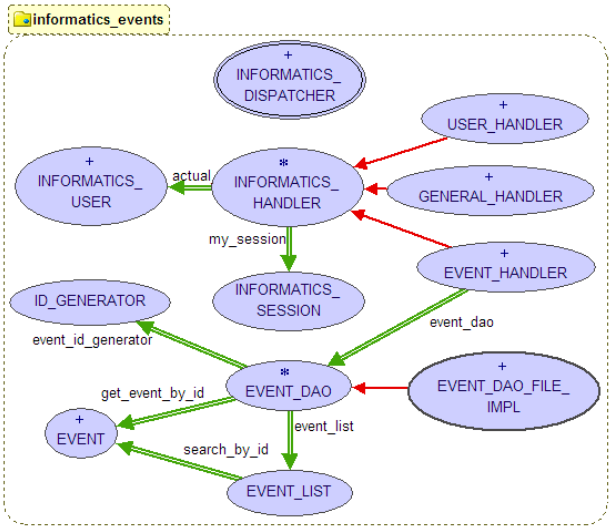

Fig 2.5 BON diagram for the Initial Implementation

Step 3. Request Elaborating

Based on the requests we listed before, we can simply take "user" as the handler-string to identify all requests that need to be processed by USER_HANDLER, and design request-strings and corresponding template usages for user related requests as following:

Request	***Request-str.***	***Templates***	***Notes***
Display registration form, *Display update-details form,* *Display add-user form,* *Display edit-user form*	**userform**	*userform.html*	*Using an additional 'mode' variable to distinguish the requests and same HTML template file*
Register user, *Update user's own profile,* *Add a user,* *Save a user*	**submit**	*userform.html*	*Same 'mode' variable/field as above, using same template (different sections)*
Display Login form	**loginform**	*loginform.html*	
login	**login**	*loginform.html*	*Same template as 'loginform', alternative sections*
Logout	**logout**	*eventlist.html*	*Let to be handled by EVENT_HANDLER, display event list*
Display user details	**details**	*userdetails.html*	
List all user accounts	**list**	*userlist.html*	
Suspend an account	**suspend**	*userlist.html*	
Activate an account	**activate**	*userlist.html*	
Delete a user	**delete**	*userlist.html*	

Some requests can have very similar web pages. It’s nice to be able to combine them to let share the same HTML template with the marker based rewriting solution with HTML_TEMPLATE_VIEW. And we also like to reuse the same command-string by adding a variable/field to distinguish the two variations of it, instead of adding more command-strings.

Step 4. Layout and Configuration File Design

We will put the application under *cgi-bin* directory, save template and user, event data in a directory one level above, and put *image*, *style sheet*, *java-scripts* in a directory under web server’s document directory, so for a typical Apache server, it looks like:

```
+cgi-bin
   -informatics_events.cgi
   -informatics_events.conf
+informatics_events
   -<html template files>
   +data
   +sessions
+htdocs
   +informatics_events
      -<javascript and other files>
      +css
      +images
```

Together with just designed command-string, template mapping information and other application variables we’d like to leave outside of the Eiffel application, we can setup our application configuration file including requests for USER_HANDLER as follows (display login form in default):

```
[general]
app_path=/cgi-bin/informatics_events.exe
default_request=user
default_command=loginform
notfound_request=other
notfound_command=notfound
stylesheet=/informatics_events/css/style01.css
javascript=/informatics_events/sorttable.js
image_path=/informatics_events/images/
error_template_page=..\informatics_events\errorpage.html

[database]
host=
port=
socketfile=
database=
username=
password=

[session]
session_files_folder=..\informatics_events\sessions\
expiration=600
session_id_length=12

[constants]
app_data_folder=..\informatics_events\data\
```

```
users_file_name=users
event_list_data_file_name=events
event_id_generator_data_file_name=event_id

[Request_Handler]
handler=USER_HANDLER
default_template=..\informatics_events\userform.html
loginform_template=..\informatics_events\loginform.html
login_template=..\informatics_events\loginform.html
details_template=..\informatics_events\userdetails.html
list_template=..\informatics_events\userlist.html
suspend_template=..\informatics_events\userlist.html
activate_template=..\informatics_events\userlist.html
delete_template=..\informatics_events\userlist.html
```

Step 5. Web Site Design

You can let web design professionals produce the whole website design first, and then doing the necessary reverse engineering by starting to design the Web application; here we have taken the approach that starts with the application, as the requirements are all well known and quite clear in all aspects.

By reusing the neat and nice site design from the CSÁRDÁS[3] project, we can get a set of static web pages for user related requests ready. Find them in *step05.zip*.

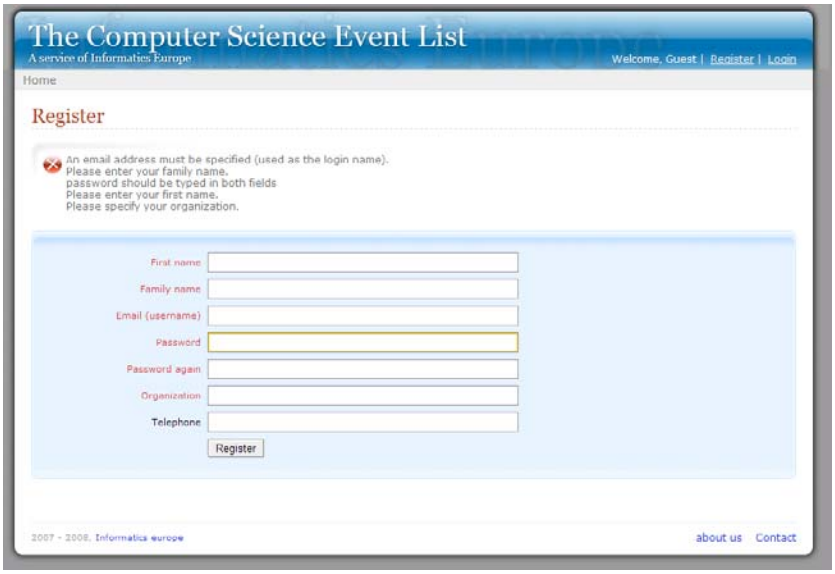

Fig 5.1 User registration with failure messages

Fig 5.2 User details

Step 6. Create Template Pages

As described in the Framework Documentation, here we will restructure designed Web pages into some templates for our application, by either replace dynamic content with markers, or organize them into sections.

- General Content

We'd like to replace the path to this Web application, to image folder, style sheet and JavaScript files with markers, so they can be updated with configuration file instead of changing these entries in every template file.

- Dynamic Menu

{#MAIN_MENU#} and {#USER_MENU#} will be used to replace those two dynamic menus displayed on top of all pages.

- Form Content

In the user registration form we will replace form title and button text with markers, so that they can be updated according to the actual context. Most importantly, we will insert markers to enable form data updating; the simplest way is to use exactly the field's name as marker name, and insert the marker in the corresponding place; for example, insert marker as '*value*' for text input box,

```
<td><input type="text"  name="first_name" value="{#first_name#}" /></td>
```

Insert marker to put selection as the first option for list boxes,

```
<select name="user_role" title="Role">
    {#user_role#}
    <option value=""></option>
    <option value="administrator">Administrator</option>
    <option value="user">User</option>
</select>
```

Insert marker for '*checked="checked"*' to check a checkbox:

```
<input type="checkbox" name="suspend" value="1 {#suspend#} />
```

And for text input boxes, a function is supplied to replace all markers when a corresponding field with valid value can be found in the request message. However, to be on the safe side, to validate form data it is recommended to implement the corresponding routines for each form, or to restore data to the form.

- Individual Sections

As mentioned before, we'd like to use one template page for several different requests, or display form validation messages only when invalid inputs are submitted. In such cases we can simply mark these content as sections, then enable or disable them when necessary.

- Alternative Sections

If several sections are mutually exclusive, we can define them as alternative sections in sequence, and switch them conveniently when necessary.

As for the template *userform.html*, we can construct it easily based on the static web page *user_register_failed.html* created before, by replacing the relevant path, the menu with markers, and by inserting markers for form title, button text, form data and a hidden ‘*mode*’ input field. Then we can wrap the HTML code that displays error messages as a VALIDATION_ERROR_MESSAGES section, and add a default disabled section ADMINISTRATOR_INFORMATION for form content that will only be displayed for administrators. Finally we can organize the form, the “*registration success*” message and the “*user updated*” message as an alternative section named USER_FORM; we have got *userform.html* ready for those several cases which will share this template page, although more content can be still introduced into this template.

These templates are friendly to be checked / updated as normal HTML files. All remaining markers, including disabled sections, can be cleaned by a handler before sending back the result page to the Dispatcher.

Check *step06.zip* for all 4 updated templates as well as for the user related requests.

Step 7. USER_HANDLER Implementation

It’s time to do some actual work to get some requests processed. Let’s start with the INFORMATICS_HANDLER class.

- INFORMATICS_DISPATCHER

We can put some common processing routines in INFORMATICS_HANDLER class:

- initialize *user_manager*, include the system defined administrator account if not yet exists;
- identify current user with information saved in session if possible;
- we can do a preliminary checking on user requests, whether user has enough permission for actual request, otherwise redirect to a ‘*Permission Denied*’ page;
- update dynamic menus based on actual context;
- rewrite some globally used markers, clean up unused markers;
- implement here several common routines which will be used by other handlers.

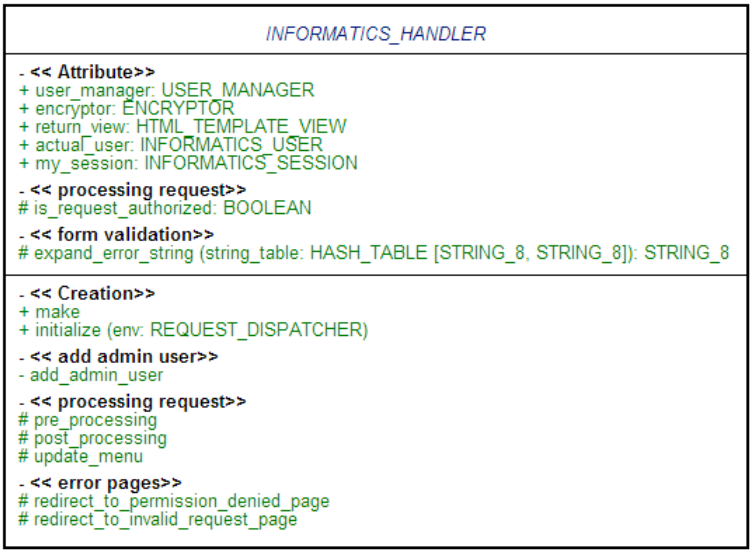

Fig 5.3 INFORMATICS_HANDLER revised

- USER_HANDLER

Nothing special but to implement corresponding routines for all predefined requests, include form validation and data restoring routines. Please refer to step07.zip for implementation details.

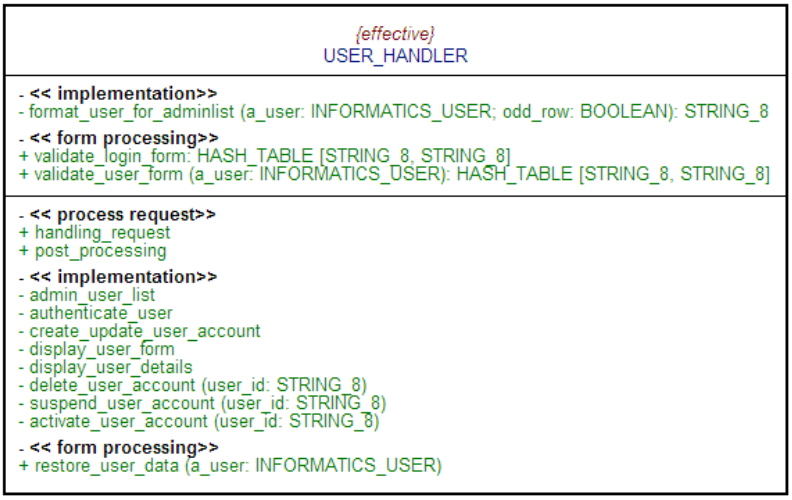

Fig 5.4 USER_HANDLER revised

If you have access to a web server, you can copy the compiled application executable and configuration file to the *cgi-bin* folder, and deploy the other files as HTML template pages, image files, and style sheets to the server. Hopefully all user related requests are working!

Step 8. Rest of the Work

As we did for user related requests, we can implement the same for EVENT_HANDLER. First get HTML template pages ready, then it’s purely Eiffel programming, doing necessary processing and create result HTML pages with marker based rewriting. As for GENERAL_HANDLER, create the template pages and that’s all.

Check the *step08.zip* or *`informatics_events`* sample application for the final implementation. You can find other dependent files under *html* folder in the same package.

References

[1] Informatics Europe, http://www.informatics-europe.org/

[2] The existing Computer Science Event List implementation, http://events.informatics-europe.org/

[3] Software Engineering Course Project CSÁRDÁS, http://se.inf.ethz.ch/teaching/ss2007/252-0204-00/project.html