Software Engineering
Prof. Dr. Bertrand Meyer
March 2007 – June 2007

Exercise session week 2

Introduction to CGI and EiffelWeb

Marco Piccioni
Some things that you may have heard about Common Gateway Interface:

- It is an old technology
- It is a slow technology
- It has some security issues
Yes, CGI is a pretty old technology

- Arose from discussions on a mailing list in 1993
- Rob McCool drafted the initial specification and provided a reference implementation in the NCSA HTTPd web server
- It allowed the creation of the first dynamic web site
- HTTPd was discontinued in 1998, but its code lived for a while in the Apache project
- Apache now runs approximately two-thirds of the internet web servers
CGI: is it really slow?

- It can be slow, mainly because:
  - It spawns a new process on the server at each request
  - As a consequence, it limits resource reuse techniques, like reusing db connections, in-memory caching, etc.

- This problem is relevant with interpreted languages, and has been addressed in two ways:
  - Storing compiled versions of the scripts on the server at first invocation
  - Embedding the interpreter directly into the web server so that it can be executed without creating a new process

- The problem is irrelevant with compiled languages
CGI and security

- CGI scripts that you deploy may (un)intentionally leak information that can help hackers break in
  - It is safer to keep CGI scripts in one directory (typically cgi-bin), setting permissions so that only the administrator can install them
- Scripts sources kept in a server directory might be read, understood and modified by a hacker
- Again, this problem is much less relevant with compiled languages, because in the cgi-bin directory you have just a binary file
Anyone who is not using Java or .NET, more or less

As an example, you may pick a Wiki, or Hotmail

It is supported by many programming languages:

Interpreted

- Any Unix shell, AppleScript
- Perl, PHP, Python, Ruby

Compiled

- Fortran, Pascal, RPG, TCL, Visual Basic
- C, C++, Eiffel
The Common Gateway Interface (CGI) is a standard protocol for interfacing external application software with an information server, commonly a web server. This allows the server to pass requests from a client web browser to the external application. The web server can then return the output from the application to the web browser.
How CGI works

- Taking the Web server's point of view, certain locations (e.g. http://www.sample.com/cgi-bin/sample.cgi) are defined to be served by a CGI program.

- Whenever a request to a matching URL is received, the corresponding program is called, with any data that the client sent as input.

- Output from the program is collected by the Web server, augmented with appropriate headers, and sent back to the client.
A first look to the classes we will need

Here are the most important EiffelWeb library classes that we will need to build the application:

- CGI_INTERFACE
- CGI_ENVIRONMENT
- CGI_FORMS
- HTML_PAGE
- CGI_RESPONSE_HEADER
BON diagram: the classes we will need
EiffelWeb: Accessing input values

- The browser sends a stream containing the data relative to the user entry and selection that will be available at the application start.

- EiffelWeb stores each data element and its associated name within a hash table, the feature `form_data` of class `CGI_INTERFACE`.
Class CGI_FORMS

- You can access values of input data from your code with the interface defined in class CGI_FORMS

- It allows you to:
  - Retrieve text entries (text fields, passwords, text areas)
  - Know whether a button was pressed or not
  - ...
Much of the information needed by CGI applications is made available via environment variables.

In class CGI_ENVIRONMENT you can find some useful piece of information about:

- Request method (GET, POST, etc)
- Query string (parameters after "?" in URLs)
- HTTP content length and type
- Cookies (to manage sessions)
- ...
Building responses: HTML_PAGE

- The response has to contain an HTTP header in order to be understood by the browser.
- Responses may include an HTML page, a re-direction or an error notification.
- You can use class `HTML_PAGE` to build an HTML page.
You may then send the header followed by your text using the features send_to_browser of class CGI_RESPONSE_HEADER:

```plaintext
response_header.generate_text_header
response_header.send_to_browser
response_header.Output.put_string (page.out)
```
What you will need

- EiffelStudio 5.7: for Mac, Linux, or Windows users with a Microsoft C compiler (e.g. with Microsoft Visual Studio installed)
  - [https://www2.eiffel.com/download](https://www2.eiffel.com/download)

- EiffelStudio 6.0 beta: for Windows users that do not have a Windows compiler installed; the second link is for getting a free Microsoft C compiler (comes with SDK)

- A nice Apache web server / MySQL distribution:

- ODBC MySQL drivers
Compiling the example

- Launch EiffelStudio.
- Click Add project
- Browse to $ISE_EIFFEL\examples\web\basic\.
  - $ISE_EIFFEL is the EiffelStudio installation dir
- Choose web.ecf
- Choose the location where the project will be compiled, by default the same directory containing the configuration file.
- Compile using the option Project/Finalize.../ and answer “Yes” to the following question (C compilation & linking)
- Also choose to keep assertions in the following window
- Pick the binary in dir
  $ISE_EIFFEL\examples\web\EIFGENs\web\F_code
Compiling the example: solving a little bug

- If you are working with Eiffelstudio 5.7 under Windows you will probably have a problem.

- Reason: some refactoring was performed for some library classes and directories, without consequently updating the xml configuration file called web.ecf in
  - $ISE_EIFFEL/library/web

- Solution: in this web.ecf (note that there are two files with the same name in different dirs), delete the following line and recompile:
  - `<cluster name="cgi_in_out" location="cgi_in_out" recursive="true"/>`
Installing and running the example

- Copy the binary file `web.exe` (if you are working under windows, otherwise the binary file called `web`) in the `cgi-bin` directory of your web server.

- Copy the file `sample.html` in a directory on the web server (default for xampp is “htdocs”).

- Access the page `sample.html` that you copied on the web server with a web browser:
  - `http://localhost/sample.html`
<html>
  <head>
    <title>EiffelWEB Example</title>
  </head>
  <body>
    <p>Please enter your name in the box below:</p>
    <form action="/cgi-bin/web" method="post" name="info">
      <input type="text" name="name" size="50">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
class SAMPLE
    inherit
        CGI_INTERFACE --this provides the basic support
create
    make -- we are using feature make from CGI_INTERFACE
feature -- Access
   Debug_mode: BOOLEAN is True -- The only way to debug: assertions!
   -- Should exception trace be displayed in case a crash occurs?
page: HTML_PAGE
   -- Page that is sent back to the browser.
feature -- Basic Operations
   --see next page--
end -- class SAMPLE
execute is

   -- Perform form entries processing, and send back the answer to the browser.
do

   if field_defined("name") then
       create page.make
       -- Add the <head> and <title> tags.
       page.add_html_code
       ("<HEAD><TITLE>EiffelWEB Example</TITLE></HEAD>")
       -- Display the name entered in the body of the page.
       page.add_html_code
       ("<BODY><H1>Hello " +text_field_value("name")
        +"</H1></BODY>")
   end

--see next page--
The main feature: execute

-- the response is now sent back to the browser

response_header.generate_text_header
response_header.send_to_browser
response_header.Output.put_string (page.out)
rescue --in case something goes terribly wrong
  io.error.putstring ("crash in `compute'
from DOWNLOAD_INTERACTION\%N")
end

--end of feature execute--
end -- class SAMPLE
A closer look to the code

- Here are the most important EiffelWeb library classes that we will need to build the application:

  - CGI_INTERFACE
  - CGI_ENVIRONMENT
  - CGI_FORMS
  - HTML_PAGE
  - CGI_RESPONSE_HEADER
EiffelWeb: Accessing input values

- The browser sends a stream containing the data relative to the user entry and selection at the applications start.

- EiffelWeb stores each data element and its associated name within a hash table, the feature `form_data` of class `CGI_INTERFACE`.
Class CGI_INTERFACE

defered class
  CGI_INTERFACE

inherit
  CGI_ENVIRONMENT

  BASIC_ROUTINES
    export
      {NONE} all -- all features from BASIC_ROUTINES are private now
    end

  CGI_FORMS

  CGI_ERROR_HANDLING

feature -- Initialization

  --see next page--
Feature make in class CGI_INTERFACE

feature -- Initialization

make is

  -- Initiate input data parsing and process information.
  local
  retried: BOOLEAN
  do
    if not retried then
      parse_input
      execute -- this is deferred in class CGI_INTERFACE
    else
      if debug_mode then
        handle_exception
      end
    end
  rescue
    retried := True
  retry
end
Other features of class CGIINTERFACE

execute is
  -- Process user provided information.
  deferred
end

feature {CGI_INTERFACE}  -- Access: note the restricted visibility
  debug_mode: BOOLEAN is
    -- Is Current application executed in debug mode?
    deferred
  end

feature {CGI_FORMS}  -- Access: note the restricted visibility
  form_data: HASH_TABLE [LINKED_LIST [STRING], STRING]
    -- User provided data.
Much of the information needed by CGI applications is made available via environment variables.

In class CGI_ENVIRONMENT you can find some useful piece of information about:

- Request method (GET, POST, etc)
- Query string (parameters after “?” In URLs)
- HTTP content length and type
- Cookies
- ...

Class CGI_ENVIRONMENT
**Class CGI_ENVIRONMENT: interface**

```
class interface
    CGI_ENVIRONMENT

create
    default_create

feature -- Cookies
    cookies: HASH_TABLE [STRING_8, STRING_8]
        -- Cookie Information relative to data.

feature -- Environment variable setting
    set_environment_variable (variable, val: STRING_8)
        -- Set environment variable `variable' to `val'.
```
Environment variables

feature -- Headerline based environment variables

http_accept: STRING_8
  -- MIME types which the client will accept.

http_user_agent: STRING_8
  -- Browser the client is using to send the request.

feature -- Not request-specific environment variables

gateway_interface: STRING_8
  -- Revision of the CGI specification to which this server complies.

server_name: STRING_8
  -- Server's hostname, DNS alias, or IP address.

server_software: STRING_8
  -- Name and version of information server answering the request.
Request specific environment variables

feature -- Request specific environment variables

  auth_type: STRING_8
    -- Protocol-specific authentication method used to validate user.

  content_length: STRING_8
    -- Length of the said content as given by the client.

  content_type: STRING_8
    -- Content type of data.

  path_info: STRING_8
    -- Extra path information, as given by the client.

  path_translated: STRING_8
    -- Translated version of PATH_INFO provided by server.

  query_string: STRING_8
    -- Information which follows ? in URL referencing CGI program.

  remote_addr: STRING_8
    -- IP address of the remote host making the request.
Class CGI_ENVIRONMENT: features

remote_host: STRING_8
    -- Hostname making the request.

remote_ident: STRING_8
    -- User name retrieved from server if RFC 931 supported.

remote_user: STRING_8
    -- Username, if applicable.

request_method: STRING_8
    -- Method with which the request was made.

script_name: STRING_8
    -- Virtual path to the script being executed.

server_port: STRING_8
    -- Port number to which request was sent.

server_protocol: STRING_8
    -- Name and revision of information protocol of this request.
You can access values of input data from your code with the interface defined in class *CGI_FORMS*.

It allows you to:
- Retrieve text entries (text fields, passwords, text areas)
- Know whether a button was pressed or not
- ...
Useful features in class CGI_FORMS

defered class interface
  CGI_FORMS

feature -- Access

button_value (field_name: STRING_8; overriding_value: STRING_8): BOOLEAN
  -- Is Button relative to 'field_name' selected?

menu_values (field_name: STRING_8): LINKED_LIST [STRING_8]
  -- Selected values for a list, whose name
  -- is 'field_name'.

text_field_value (field_name: STRING_8): STRING_8
  -- First (unique?) value for a text field.
  -- Applies for a password and a text area too.

feature -- Implementation

form_data: HASH_TABLE [LINKED_LIST [STRING_8], STRING_8]
  -- Table in which is contained all the information
  -- relative to the different user inputs.
feature -- Advanced Access

    fields: ARRAY [STRING_8]
      -- Names of fields in the form.

    value_count (field_name: STRING_8): INTEGER_32
      -- Number of values for a field.

    value_list (field_name: STRING_8): LINKED_LIST [STRING_8]
      -- List of values for a field.

feature -- Report

    field_defined (field_name: STRING_8): BOOLEAN
      -- Is field `field_name' defined?

end -- class CGI_FORMS
The response has to contain an HTTP header in order to be understood by the browser.

Responses may include an HTML page, a re-direction or an error notification.

You can use the class `HTML_PAGE` to build an HTML page.
Class HTML_PAGE

class interface
    HTML_PAGE

create
    make,
    make_from_template

feature -- Initialization

    make
        -- Create an HTML page.

    make_from_template (fi_n: STRING_8)
        -- Create an HTML page from a template whose path name is
        -- 'fi'. The template may contains special symbols/words, which
        -- will allow smart replacing (see feature 'replace').

feature -- Access

    out: STRING_8 -- Usable copy of the output.
Class HTML_PAGE: features

feature -- Basic Operations

    add_html_code (s: STRING_8)
        -- Add html code 's'.

    insert_hidden_field (name, value: STRING_8)
        -- Insert hidden field with name 'name' and value 'value'.

    replace_marker (a_marker, s: STRING_8)
        -- Replace marker 'a_marker' by string 's'
        -- within the template.
        -- Do nothing if it does not exist.

end -- class HTML_PAGE
You may then send the header followed by your text using the features `send_to_browser` of class `CGI_RESPONSE_HEADER`:

```eiffel
response_header.generate_text_header
response_header.send_to_browser
response_header.Output.put_string (page.out)
```
Class CGI_RESPONSE_HEADER

class interface
    CGI_RESPONSE_HEADER

create
default_create

feature -- Access

header: STRING_8
    -- Message Header which will be returned to the
    -- the browser.

is_complete_header: BOOLEAN
    -- Is Current header a complete header ?

is_sent: BOOLEAN
    -- Is current header sent to the browser ?
Class CGI_RESPONSE_HEADER: settings

feature -- Advanced Settings

set_cookie (key, value, expiration, path, domain, secure: STRING_8)
  -- Set a cookie on the client's machine
  -- with key 'key' and value 'value'.

set_expiration (a_date: STRING_8)
  -- Set the expiration date before which the page needs
  -- to be refreshed

setPragma (aPragma: STRING_8)
  -- Set the pragma which indicates whether
  -- the page accepts to be cached
  -- or not. An example of pragma is "no-cache"
Class CGI_RESPONSE_HEADER: features

feature -- Basic Operations

generate_http_redirection (an_url: STRING_8; is_secure: BOOLEAN)
   -- Generate CGI secure re-direction, via 'https' protocol if
   -- secure, via http if not.

generate_text_header
   -- Generate header for a future text (generally HTML)
   -- you are going to send.

reinitialize_header
   -- Re-initialize header.
   -- May be called if the header built so far
   -- has to be re-build from scratch.
CGI_RESPONSE_HEADER: other features

return_status (a_status: INTEGER_32; a_message: STRING_8)
  -- Set the status of the user request.
  -- A complete list of status may be found at:
  -- http://www.w3.org/hypertext/WWW/protocols/HTTP/HTRESP.html
  -- See also CGI_COMMON_STATUS_TYPES

send_to_browser
  -- Send the header to browser.
  -- This operation has to be performed before
  -- you send anything else to the browser.

end -- class CGI_RESPONSE_HEADER
End exercise session
week 2