EWF / March 2012

Quick presentation of Eiffel Web Framework
March the 15th, 2012

EWF what is that?

EWF stands for **E**iffel **W**eb **F**ramework

Becoming the common platform to build web application with Eiffel.

Server application runs on any platform/HTTP server thanks to the Connectors

Also provide utility, client, ... components

EWF : community project

This is fully open source

Written in void-safe Eiffel.

Works with ISE Eiffel 6.8, 7.0, 7.1

Tested on Windows, Linux,

with apache2, iis, CGI, FastCGI

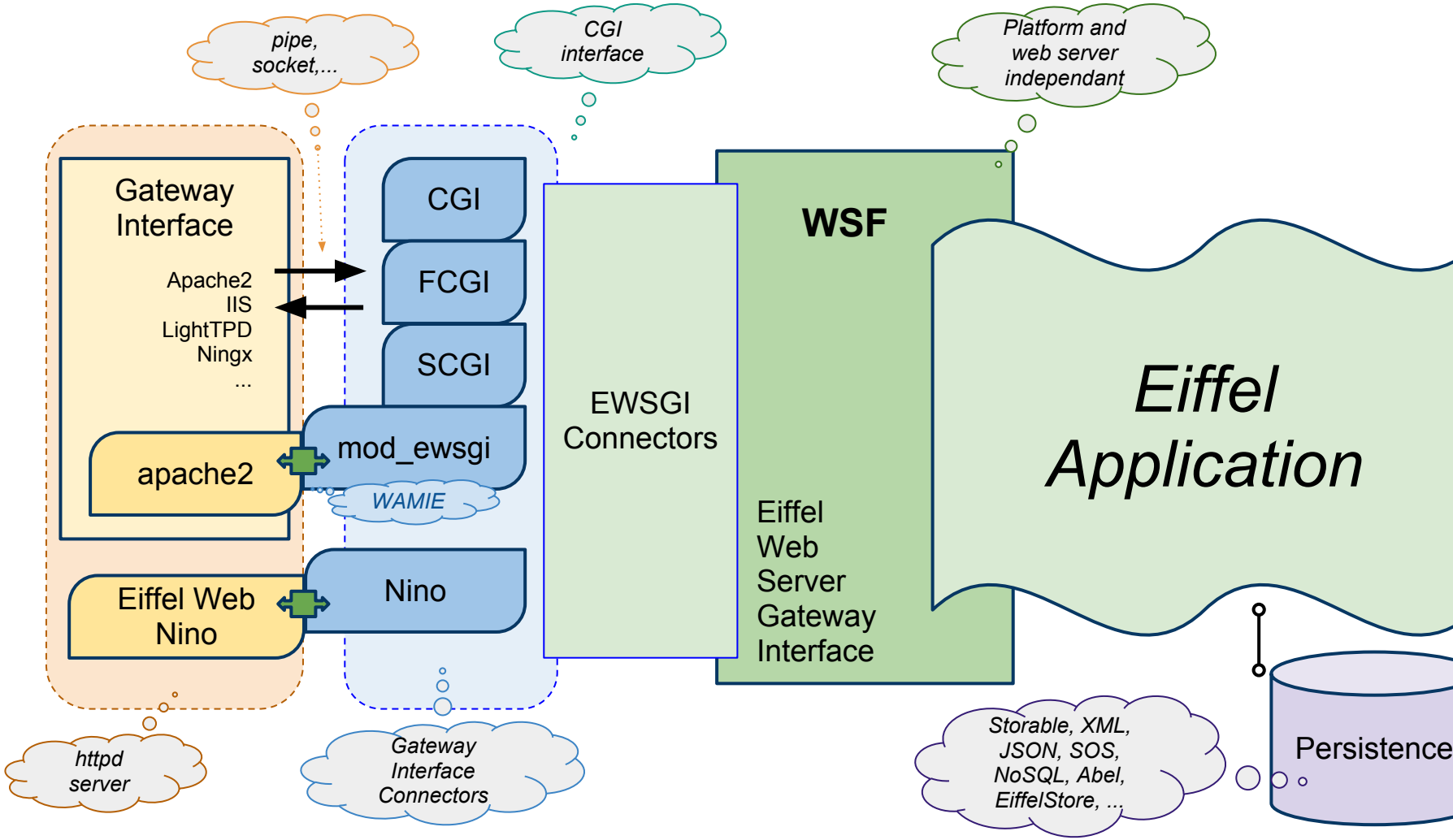
and EiffelWebNino

Home page:

http://eiffel-world.github.com/Eiffel-Web-Framework/

Main contributors:

Jocelyn Fiat and Javier Velilla



Notes:

WAMIE: Writing Apache Module In Eiffel
Nino: Web server written in Eiffel
SOS: Simple Object Storage
Abel: Unified Persistence layout

Connectors: web server support in EWSGI library

Also client side

Also provides utility, client, … components

- **http client** (simple client based on libcurl)
- **error**
- **CONneg:** Content negociation
- **URI template** /order/{order-id}
- **http**: for status code constants and related
- **encoder:** base64,url,html,xml,json,utf8,…

Main class for server

Note: we dropped the WSF_ prefix in those slides

REQUEST : access data related to the http request
RESPONSE : media to send data back to the client
HTTP_HEADER : user friendly HTTP header builder

REQUEST_ROUTER,
REQUEST_HANDLER, and
REQUEST_HANDLER_CONTEXT

SERVICE and DEFAULT_SERVICE_LAUNCHER

Server Service

Execute request handling

Service

```
deferred class WSF_SERVICE
feature -- Execution
        execute (req: WSF_REQUEST; res: WSF_RESPONSE)
                deferred
                end
end
-----------------------
class MY_SERVER
inherit WSF_SERVICE
create make_and_launch
feature
   make_and_launch
     local
        s: DEFAULT_SERVICE_LAUNCHER
     do
        create package_manager.make (package_root)
        create s.make_and_launch (agent execute)
     end
-- implement `execute'
```

Server Request

request data coming from the client

REQUEST

WSF_REQUEST

- meta variables *(see CGI meta variables)*

  also available as functions: path_info, content_type, request_method, ...
- query parameter(s): *extracted from QUERY_STRING*
- form parameter(s) *and* uploaded_files
- cookie(s): *extracted from input_data*
- raw_input_data *if enabled and input data processed ...*
- script_url (..) *to return url related to current query*

Note: many plural functions return ITERABLE [WSF_VALUE],
to allow other WGI implementations to use LIST, HASH_TABLE,
or any DS_ flavours

../..

REQUEST 2/3

WSF_REQUEST ../..

- wgi_connector: WGI_CONNECTOR

Even if EWF is portable, an application might want to know the underlying connector have adapt its behavior, or display more information.

- execution_variable(s): *in addition to other variables and parameters, this can be use to keep some data globally associated to the request. A usual need would be SESSION, or Authorization which is computed once.*

- mime_handler (..) *and* register_mime_handler (..)

The mime handler are used to handle the input data according to the content-type (multipart_form_data, application/x-www-form-urlencoded). This allow the user to replace default behavior and register new (XForm, ...)

REQUEST 3/3

WSF_REQUEST ../..

- input: INPUT_STREAM
- chunked_input: detachable CHUNKED_INPUT_STREAM
- is_chunked_input: BOOLEAN

Access to the input stream, either normal or chunked depending on the input transfer encoding.

So far, the INPUT_STREAM does not provide direct access to the underlying socket, pipe, file, ... however it provides read_character, read_string, ...

To be discussed...

input.associated_file: detachable FILE could be an option. Either it is available or not ...

Server Response

sending back to the client

RESPONSE 1/2

WSF_RESPONSE

- set_status_code

Depending on the connector, the status code can be sent or not

- put_header(_*)

3 Methods to put http header data

- put_string (string)

Put string to the client

- put_substring (string, start_index,end_index)

Put substring to the client (mainly for performance)

- flush

Flush the data, in case there is any buffering

../..

RESPONSE 2/2

WSF_RESPONSE

../..

- put_chunk

When "Transfer-Encoding: chunked", put a chunk of response

- put_response (RESPONSE_MESSAGE)

Put an object containing status code, header and content to be sent

- redirect_* (..)

Various user friendly methods to send a redirection to the client

*So far **no direct access** to the eventual output stream/socket/file/...*
to be discussed for future version

Server
Request dispatcher / Router

dispatch request according to the request uri

REQUEST_ROUTER ... 1/4

Goal: Provide friendly components to dispatch requests according to the request URI

Examples:

- GET /order/123 -> return order identified by 123
- POST /order/ -> create new order
- POST /order/123 -> modify order identified by 123

Do it yourself thanks to REQUEST.request_uri, or use the Router components

REQUEST_ROUTER ... 2/4

The WSF library provides (for now) 2 kinds of router

- REQUEST_URI_ROUTER
  router using simple URI : equivalent to starts_with (..)

- REQUEST_URI_TEMPLATE_ROUTER
  router using URI Template: matching the URI template such as /order/{order-id}

How to use ? ...

REQUEST_ROUTER .. 3/4

Inherit from URI_TEMPLATE_ROUTED_SERVICE
define router, create_router, and setup_router

For instance

```
setup_router
    do
        router.map_with_request_methods (/orders/{order-id},
            orders_handler, <<"GET", "POST">>)
        router.map_with_request_methods (/orders/,
            new_orders_handler, <<"POST">>)
        router.map_with_request_methods (/orders/,
            all_orders_handler, <<"GET">>)
    end
```

Just "map" a **resource** uri template, with a request **handler**, and precise the allowed request **methods**. ../..

REQUEST_ROUTER .. 4/4

And then handler should implement

```
execute (ctx: REQUEST_URI_TEMPLATE_HANDLER_CONTEXT;
            req: WSF_REQUEST;
            res: WSF_RESPONSE)
        do
                ...
        end
```

The `ctx' provides additional information such as

- path_parameter(s): to get value associated with {order-id}
- uri_template: matching uri template that routed to this handler
- path: associated path in the url
- query_parameter(s)... from REQUEST
- request: associated REQUEST object

Server Values

Values for variables, parameters, ...

Values

deferred WSF_VALUE *with a `name'*

- WSF_STRING: *foo=bar*
- WSF_MULTIPLE_STRING: *foo=abc&foo=bar*
- WSF_TABLE: *foo[]=a&foo[]=b*
- WSF_UPLOADED_FILE: *uploaded file from form (POST)*
- WSF_ANY: *use to store anything in execution_variable(s)*

WSF_STRING has for instance

- *name: STRING_32*
- *string: STRING_32*
- *url_encoded_name: STRING_8*
- *url_encoded_string: STRING_8*

Questions ?

…

Questions ?

…

Questions ?

…