



Exploiting the full power of OCaml in Web programming

Vincent Balat

OCaml Meeting 2008 January 26th, 2008, Paris

1 Overview

A few words about PPS



Proofs, Programs and Systems

Guy Cousineau, Pierre-Louis Curien, Jérôme Vouillon, Roberto Di Cosmo, ...

Web programming in OCaml

- OCamlnet
- Mod_caml (Cocanwiki ...)
- WDialog
- ASXCaml (?)
- ...

The Ocsigen project

Goal:

Find new programming techniques for the Web

- Improving accessibility and robustness
- High level concepts to simplify programmers work
- Better fitted to Web 2.0

Two projects: Ocsigen and WebSiCoLa

Idea born in 2000. Beginning of implementation: january 2005

The Ocsigen project

One main idea:

continuation based Web programming — Christian Queinnec

Other functional Web programming tools:

Seaside

Links

Hop

Wash

PLT scheme

The Ocsigen project

Ocsigen 1.0 (First step)

- Full-featured Web server
- Eliom: continuation based Web programming
- Ocsimore: more libraries for Ocsigen

<http://www.ocsigen.org>

A free software project

Open source community

Community of users (Web developpers) : Nurpawiki, Lambdium ...

Authors and contributors: Vincent Balat, Jérôme Vouillon, Gabriel Kerneis, Denis Berthod, Piero Furiesi, Jaap Boender, Stéphane Glondu, Thorsten Ohl, Nataliya Guts, Jérôme Velleine, Pierre Clairambault, Dario Teixeira, Janne Hellsten, ...

The Web server

- HTTP protocol
- Powerful extension mechanism
- Static pages
- access control
- Compression of data
- CGI module
- Reverse proxy
- Eliom

Cooperative threads

- Only one thread of execution
- Never use blocking functions: go back to another continuation instead

LWT by Jérôme Vouillon Cooperative threads in monadic style

```
let r = request () in in  $\mapsto$  request () >>= fun r ->  
parse_answer r parse_answer r
```

+ preemptive threads for non-cooperative libraries

2 Eliom

2.1 Static typing of pages

Static checking of HTML

Several output modules:

Text	Untyped pages
Xhtml	Type checking with polymorphic variants
OcamlDuce	Type checking with OCamlDuce (XHTML, XML)
CSS	
File	
Redirection	

```
let create_page sp mytitle mycontent =  
  Lwt.return  
    << <html>  
      <head><title>$str:mytitle$</title></head>  
      <body><h1>$str:mytitle$</h1>$list:mycontent$</body>  
    </html> >>
```

```
let create_page2 sp mytitle mycontent =  
  Lwt.return  
    (html  
      (head (title (pcdata mytitle)) [])  
      (body ((h1 [pcdata mytitle])::mycontent)))
```

2.2 Functional services

Services

Links/forms = function calls (services)

```
let mainpage = new_service ~path:[] ~get_params:unit ()
```

```
let () = register mainpage  
(fun sp () () ->  
  Mylib.create_page sp  
    "Messages"  
    [Mylib.display_message_list ()])
```

```
let msgpage = new_service ~path:[] ~get_params:(int "n") ()
```

```

let () = Xhtml.register msgpage
  (fun sp n () ->
    Mylib.create_page sp
      ("Message_␣"^(string_of_int n))
      [Mylib.display_message n])

```

Eliom services

A link towards a service with one `int` parameter:

```
a msgpage sp [pdata "click"] 4
```

Benefits

- No broken links
- Static checking of the types of parameters

Coservices: Solving the back-button problem

Create dynamically new services

	Services	Coservices
Text		
Xhtml		
OcamlDuce		
CSS		
File		
Redirection		

```

let () = register addmsgpage
  (fun sp () msg ->
    let ok =
      new_coservice ~max_use:1 ~fallback:mainpage ~get_params:unit ()
    in
    register ~sp ~service:ok
      (fun sp () () ->
        Mylib.register_message msg;
        Mylib.create_page sp
          "Message_␣added"
          [Mylib.display_message_list ()]);

```

```

Mylib.create_page sp
  "Confirm_ this_ Message?"
  [p [pdata msg];
   p [
    a ok sp [pdata "Yes"] (); pdata "_";
    a mainpage sp [pdata "Cancel"] ()]
  ]
)

```

2.3 Taxonomy of services

Actions

Create dynamically new services

	Services	Coservices
Text		
Xhtml		
OcamlDuce		
CSS		
File		
Redirection		
Action		

Non-attached services

POST and GET parameters

Sessions

Full taxonomy of services

	Services		Coservices	
	attached	non-attached	attached	non-attached
Text				
Xhtml				
OcamlDuce				
CSS				
File				
Redirection				
Action				

+ POST services

+ session services

Example of non-attached services: connection of users

```

let connect_action =
  new_post_service' ~name:"connect" ~post_params:(string "login") ()
let disconnect_action =
  new_post_service' ~name:"disconnect" ~post_params:unit ()

let () = Actions.register disconnect_action
  (fun sp () () ->
    Eliomsessions.close_session ~sp () >>= fun () ->
    Lwt.return [])

let () = Actions.register connect_action
  (fun sp () login ->
    Eliomsessions.close_session ~sp () >>= fun () ->
    Eliomsessions.set_volatile_data_session_group
      ~set_max:(Some 10) ~sp login;
    Lwt.return [])

```

Conclusion

Summary of Eliom's concepts

- Static checking of pages
- Full set of service kinds
 - Strong use of continuation based Web programming
 - Very precise control of URLs
 - Highly related to concrete needs of Web developers
 - Reduces by a lot the number of lines of code
 - No dead links
 - Typing of page parameters and forms
- Automatic sessions
 - Automatic cookie management
 - Session coservices
 - Timeouts
 - Session data (persistent or not)
 - Garbage collection of data
 - Session groups

Future

Version 1 (a few weeks).

Then:

- Higher level features for Eliom
- High level layout for Web sites
- Organizing data. Content management
- Executing code on the client
 - Writing a distributed program in OCaml
 - Checking the correctness of a client side program with respect to a Web site
 - ...