Ocsigen

Web applications with Js_of_ocaml and Eliom

Vincent Balat

OUPS meetup — 8 july 2014

The project

created in 2004, more than 200 000 l.o.c., LGPL

People:

Vincent Balat, Jérôme Vouillon,

Pierre Chambart, Grégoire Henry, Benedikt Becker, Benjamin Canou, Boris Yakobowski, Jérémie Dimino, Gabriel Radanne, Hugo Heuzard,

Charly Chevalier, Enguerrand Decorne, Romain Calascibetta, Jacques-Pascal Deplaix, Grégoire Lionnet, Raphaël Proust, Stéphane Glondu, Jérôme Maloberti, Gabriel Kerneis, Arnaud Parant, Christophe Lecointe, Denis Berthod, Gabriel Cardoso, Piero Furiesi, Jaap Boender, Baptiste Strazzulla, Thorsten Ohl, Gabriel Scherer, Séverine Maingaud, Simon Castellan, Jean-Henri Granarolo, Archibald Pontier, Nataliya Guts, Cécile Herbelin, Charles Oran, Jérôme Velleine, Pierre Clairambault …

université PARIS DIDEROT PARIS 7

cnrs

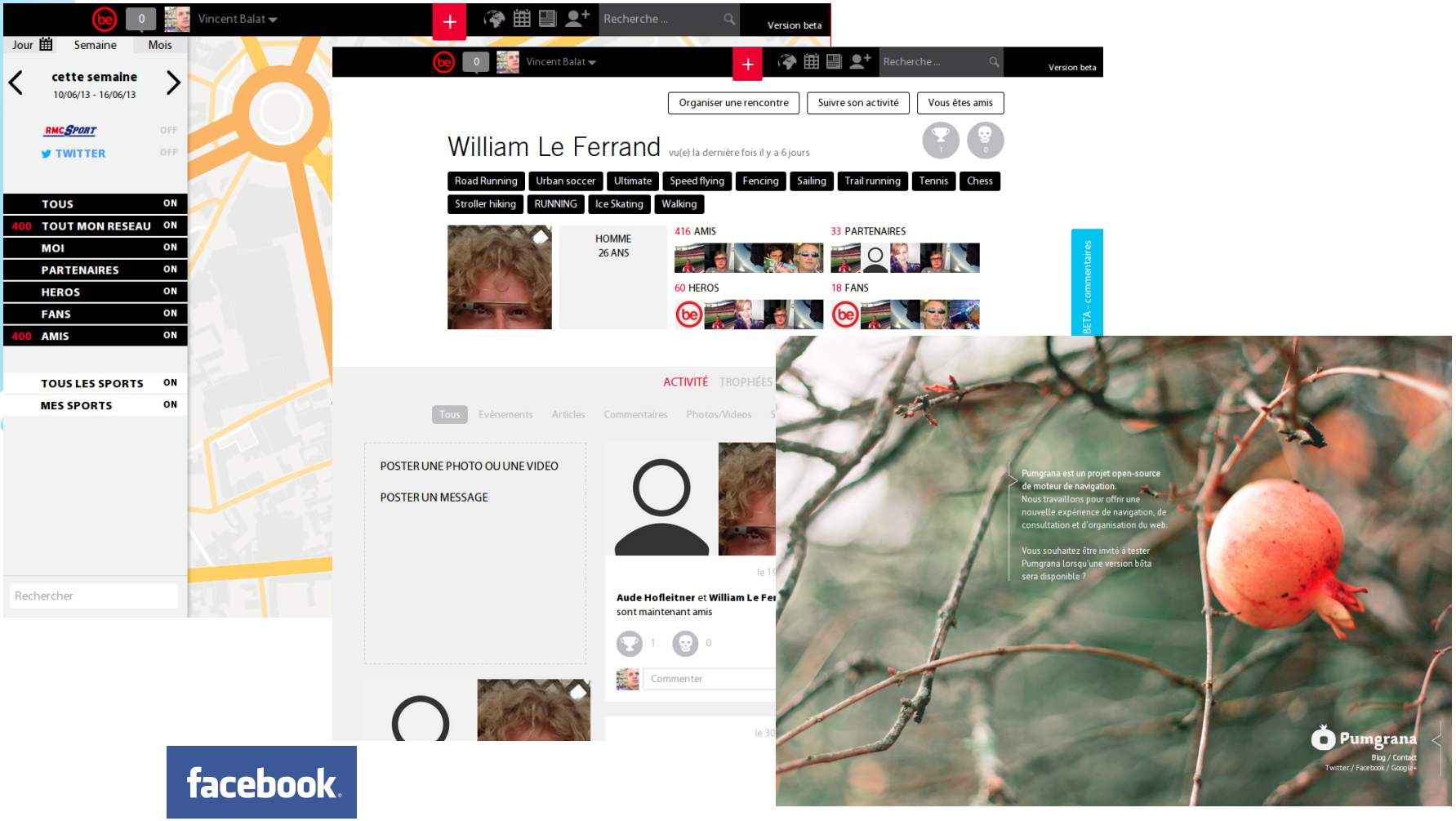
AGENCE NATIONALE DE LA RECHERCHE ANR

IRILL

Inria INVENTEURS DU MONDE NUMÉRIQUE

ocsigen fresh air in web programming

Users



Some companies and open source projects:

BeSport, NYU CGSB Genomics Core, Pumgrana, Facebook, Life.tl, Ashima Arts, Metaweb/Freebase, Hypios, Ocamlcore, Ocamlpro, Baoug, Nleyten...

ocsigen
fresh air in web programming

Some of our projects

ocsigen eliom

More information

Write client/server Web applications in very few lines of OCaml code!

ocsigen js_of_ocaml

More information

An OCaml to Javascript compiler.

ocsigen lwt

More information

A cooperative threading library for OCaml.

ocsigen server

More information

A full-featured and extensible Web server.

> And many other projects ...

ocsigen
fresh air in web programming

What for?

Static Web sites

Dynamic Web sites (server only)

Browser apps (client only)

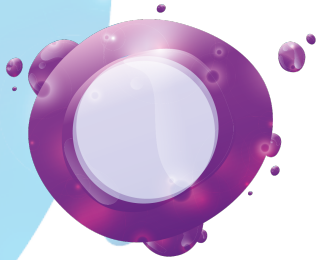
Web apps (client-server)

Mobile apps (HTML5 based)

1

Running OCaml programs in a Browser

1 Running OCaml programs in a Browser



ocsigen
fresh air in web programming
js_of_ocaml

A compiler that translates OCaml to JS

1 Running OCaml programs in a Browser

ocsigen
fresh air in web programming
js_of_ocaml

A compiler that translates OCaml's bytecode to JS

Any program can be translated

- even if it is using non-open source libraries
- provided it is not using preemptive threads
- provided the external functions are implemented in JS

Interact with JS

- using external functions written in JS
- using unsafe calls
- or type safe calls using a syntax extension:

```
elt##style##background <- Js.string "#45fe65";

Dom_html.document##body##appendChild(elt);
```

To bind a library in type safe manner,
you just need to write the types of all the functions/objects you need.

Typing HTML

```
html
  (head (title (pcdata "Typing html")) [])
  (body [h1 [pcdata "Non-valid page"];
         p [p [pcdata "a paragraph in a paragraph"]]]))))
```

Typing HTML

```
html
  (head (title (pcdata "Typing html")) [])
  (body [h1 [pcdata "Non-valid page"];
        p [p [pcdata "a paragraph in a paragraph"]]]))
```

```
Error: This expression has type
         ([> Html5_types.p ] as 'a) Eliom_content.Html5.F.elt
       but an expression was expected of type
         ([< Html5_types.p_content_fun ] as 'b) Eliom_content.Html5.F.elt
       Type 'a = [> `P ] is not compatible with type
         'b =
           [< `A of Html5_types.phrasing_without_interactive
            | `Abbr
            | `Audio of Html5_types.phrasing_without_media
            | `Audio_interactive of Html5_types.phrasing_without_media
            | `B
            ...
           ]
```

Services

Raw HTML links:

```
Raw.a ~href:"http://desertsuperstar.tumblr.com/page/8"
      [pcdata "Click"]
```

Services

Raw HTML links:

```
Raw.a ~href:"http://desertsuperstar.tumblr.com/page/8"
      [pcdata "Click"]
```

Links using *services*:

```
let myservice = Eliom_service.Http.external_service
                ~prefix:"http://desertsuperstar.tumblr.com"
                ~path:["page";""]
                ~get_params:(suffix (int "num"))
                ()

... a ~service:myservice [pcdata "Click"] 4
```

Services

Raw HTML links:

```
Raw.a ~href:"http://desertsuperstar.tumblr.com/page/8"
      [pcdata "Click"]
```

Links using *services*:

```
let myservice = Eliom_service.Http.external_service
              ~prefix:"http://desertsuperstar.tumblr.com"
              ~path:["page";""]
              ~get_params:(suffix (int "num"))
              ()

... a ~service:myservice [pcdata "Click"] 4
```

Typing of links and forms

Writing services in OCaml

```
let myservice2 =
  Eliom_registration.Html_text.register_service
    ~path:["aaa"; "bbb"]
    ~get_params:Eliom_parameter.any
    (fun params () ->
       Lwt.return "<html>
                     <head><title>A</title></head>
                     <body>B</body>
                   </html>")
```

Writing services in OCaml

```
let myservice2 =
  Eliom_registration.Html5.register_service
    ~path:["aaa"; "bbb"]
    ~get_params:Eliom_parameter.any
    (fun params () ->
       Lwt.return (html (head ...) (body [...])))
```

Writing services in OCaml

```
let myservice2 =
  Eliom_registration.Html5.register_service
    ~path:["aaa"; "bbb"]
    ~get_params:Eliom_parameter.any
    (fun params () ->
       Lwt.return (html (head ...) (body [...])))
```

Services may return:

HTML, Files, Redirections Actions, Applications, JSON, etc.

Writing services in OCaml

```
let myservice2 =
  Eliom_registration.Html5.register_service
    ~path:["aaa"; "bbb"]
    ~get_params:Eliom_parameter.any
    (fun params () ->
       Lwt.return (html (head ...) (body [...])))
```

Services may return:

HTML, Files, Redirections Actions, Applications, JSON, etc.

→ **See tutorial *"Writing a RESTful app with Eliom"***

NEW! (contribution by Domoco)

ocsigen

fresh air in web programming

Client-server communication

Syntax %v makes possible to use on *client side* a value v defined on *server side*.

The value is send together with the page.

```
let s = "hello"
```

```
p [pcdata %s]
```

6 Client-server communication

Syntax %v makes possible to use on *client side* a value v defined on *server side*.

The value is send together with the page.

If f is a server side function, calling %f will do a remote procedure call.

```
let f l = List.map succ l
```

```
... match %f [1; 2; 3] with [] -> ...
```

6 Client-server communication

Syntax %v makes possible to use on *client side* a value v defined on *server side*.

The value is send together with the page.

If f is a server side function, calling %f will do a remote procedure call.

```
let f l = List.map succ l
```

```
... match %f [1; 2; 3] with [] -> ...
```

Function f must be exported explicitely:

```
let f = server_function Json.t<int list> f
```

Client and server sections

```
{server{

  let myservice = ...

}}
{client{

  ... a ~service:%myservice [pcdata "Click"] ()

}}
```

Client and server sections

```
{shared{

  type toto = ...

}}
{server{

  let myservice = ...

}}
{client{

  ... a ~service:%myservice [pcdata "Click"] ()

}}
```

Client values

```
let mybutton s =
  let d = div [pcdata "click"] in
  let _ =
    {{ Lwt_js_events.clicks %d
         (fun ev ->
            Dom_html.window##alert(Js.string s))
    }}
  in
  d
```

Client values

```
let mybutton s =
  let d = div [pcdata "click"] in
  let _ =
    {{ Lwt_js_events.clicks %d
         (fun ev ->
            Dom_html.window##alert(Js.string s))
    }}
  in
  d
```

→ Makes possible to generate pages from server side

(good for indexing by search engines)

Client values

```
let mybutton s =
  let d = div [pcdata "click"] in
  let _ =
    {{ Lwt_js_events.clicks %d
         (fun ev ->
            Dom_html.window##alert(Js.string s))
    }}
  in
  d
```

Client values

```
let mybutton s =
  let d = div [pcdata "click"] in
  let _ =
    {{ Lwt_js_events.clicks %d
         (fun ev ->
            Dom_html.window##alert(Js.string s))
    }}
  in
  d
```

In a shared section you can call the function from server or client sides according to your needs

8 Client values

```
let mybutton s =
  let d = div [pcdata "click"] in
  let _ =
    {{ Lwt_js_events.clicks %d
         (fun ev ->
            Dom_html.window##alert(Js.string s))
    }}
  in
  d
```

In a shared section you can call the function from server or client sides according to your needs

→ See tutorial *"Client-server widgets"*

Client values

```
{server{

  let client_fun = {{ fun x -> x+1 }}

}}
{client{
        ... %client_fun 3 ...
}}
```

Other features

- Service identification mechanism
- Advanced sessions, with scope (browser, tab, user …)
- Bidirectional client-server communication
- Client-server reactive pages NEW! **(contribution by Besport)**
- …

Documentation

The starting point: `ocsigen.org/tuto/`

- A short tutorial for writing client-server page widgets
- A short tutorial for sending basic untyped pages
- Tutorial: write a collaborative drawing application in 80 lines
- Tutorial: write a RESTful API using Eliom
- …

+ manual and API documentation for each project