



Ocsigen

Vincent Balat

Epita — 12 mars 2014

Affiliation

Ocsigen est logiciel libre issu d'un projet de recherche (Univ Paris Diderot, CNRS et Inria).

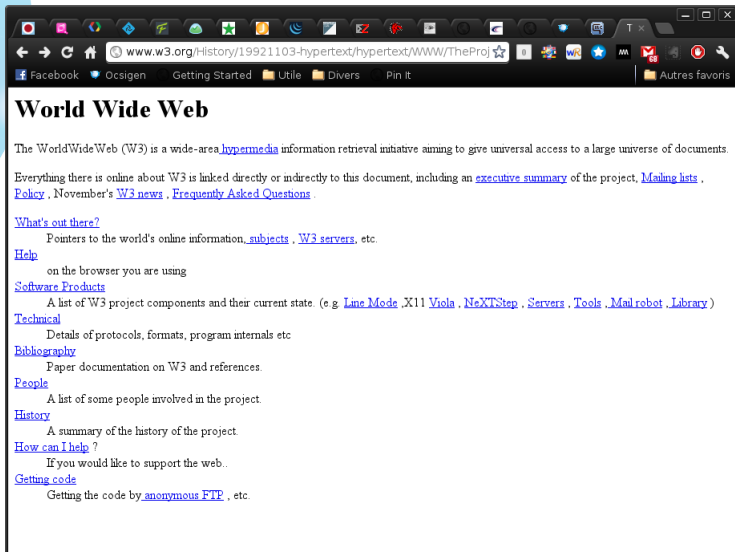


2 projets ANR



Collaborations avec UPMC, Inria Sophia Antipolis et Université Paris Nord.

L'évolution du Web



L'évolution du Web



WIKIPÉDIA
The Free Encyclopedia

- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia

▼ Interaction

- Help
- About Wikipedia
- Community portal
- Recent changes
- Contact Wikipedia

► Toolbox

► Print/export

▼ Languages

- العربية
- Català
- Česky
- Deutsch
- Ελληνικά
- Español
- Français
- Galego
- 한국어
- Italiano
- ქართული
- Latina
- Bahasa Melayu
- Nederlands
- 日本語

Log in / create account

Article Talk

Read Edit View history

Search

OCaml

From Wikipedia, the free encyclopedia
(Redirected from Objective Caml)

OCaml (/ɔ.ɔ kæməl/ *oh-kah-el*), originally known as **Objective Caml**, is the main implementation of the **Caml** programming language, created by **Xavier Leroy**, **Jérôme Vouillon**, **Damien Doligez**, **Didier Rémy** and others in 1996. OCaml extends the core Caml language with object-oriented constructs.

OCaml's toolset includes an interactive toplevel **interpreter**, a **bytecode compiler**, and an optimizing **native code compiler**. It has a large standard library that makes it useful for many of the same applications as **Python** or **Perl**, as well as robust modular and object-oriented programming constructs that make it applicable for large-scale software engineering. OCaml is the successor to **Caml Light**. The acronym **CAML** originally stood for **Categorical Abstract Machine Language**, although OCaml abandons this abstract machine.

OCaml is a **free open source** project managed and principally maintained by **INRIA**. In recent years, many new languages have drawn elements from OCaml, most notably **F#** and **Scala**.

Contents [hide]

- 1 Philosophy
- 2 Features
- 3 Code examples
 - 3.1 Hello World
 - 3.2 Summing a list of integers
 - 3.3 Quicksort
 - 3.4 Birthday paradox
 - 3.5 Church numerals
 - 3.6 Arbitrary-precision factorial function (libraries)
 - 3.7 Triangle (graphics)
 - 3.8 Fibonacci Sequence
- 4 Derived languages
 - 4.1 MetaOCaml
 - 4.2 Other derived languages
- 5 Software written in OCaml
- 6 See also
- 7 References
- 8 External links

OCaml

Paradigm(s)	multi-paradigm; imperative, functional, object-oriented
Appeared in	1996
Developer	INRIA
Stable release	3.12.1 (July 4, 2011; 8 months ago)
Typing discipline	static; strong, inferred
Dialects	F#, JoCaml, MetaOCaml, OCamlP3I
Influenced by	Caml Light, Standard ML
Influenced	F#, Scala, ATS, Opa
Implementation language	Programming language
OS	Cross-platform
License	Q Public License (compiler) LGPL (library)
Website	caml.inria.fr/index.en.html
 Objective Caml at Wikibooks	

[edit]

L'évolution du Web

facebook

Adresse électronique
Mot de passe
Connexion
Garder ma session active Mot de passe oublié ?

**Ocsigen**
est sur Facebook.
Pour communiquer avec Ocsigen, inscrivez-vous sur Facebook dès maintenant.
Inscription Connexion

**Ocsigen**
17 personnes aiment ça · 2 personnes en parlent

Logiciels
The Ocsigen project is aimed at proposing clean and safe tools for developing and running client/server Web applications.

 17

À propos Photos Mentions J'aime

Message sur le mur Photo
Exprimez-vous

**Ocsigen** a partagé un lien.
Il y a environ une heure

Ocsigen Js_of_ocaml 1.1 released. (WebGL support, bug fixes ...) http://ocsigen.org/js_of_ocaml/

**Js_of_ocaml**
ocsigen.org
Js_of_ocaml is a compiler of OCaml bytecode to Javascript. It makes it possible to run OCaml programs in a Web browser.

Également sur
 <http://ocsigen.org/> *

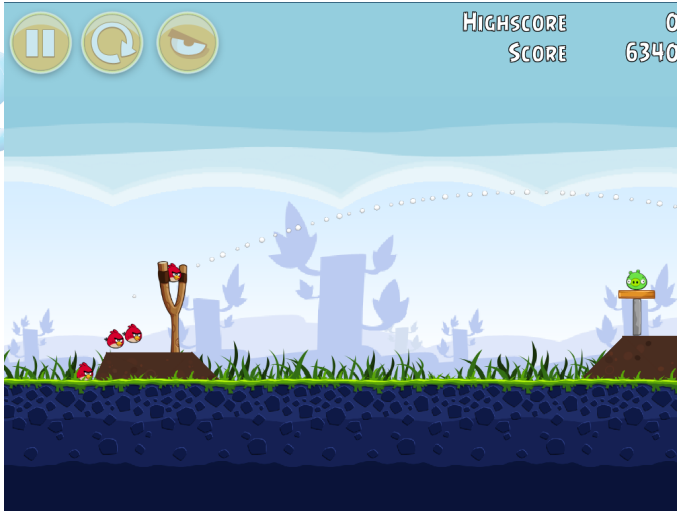
**Ocsigen** a partagé un lien.
9 mars

Ocsigen Server 2.0.4 released (internal changes)
<http://ocsigen.org/ocsigenserver/>

**Ocsigen server**
ocsigen.org
Ocsigen Server is a full featured Web server. It implements most features of the HTTP protocol, and has a very powerful



L'évolution du Web



L'évolution du Web

The screenshot displays the Deezer website interface. At the top, there's a navigation bar with links for 'Actualités', 'Ma musique', 'Radio', 'Offres Premiums', 'Découvrir Deezer', 'Créer un compte', 'S'identifier', and 'S'identifier avec Facebook'. Below this is a music player for 'Sleeper - Dan San'. The main banner features the artist **NADÉAH** with the text 'GAGNEZ 50X2 PLACES POUR SON CONCERT À LA CIGALE DE PARIS LE 2 AVRIL'. There are buttons for 'JOUER !' and 'Écouter l'album'. Below the banner is a horizontal menu with categories: GROUNDTATION, **NADÉAH**, SPOEK MATHAMBO, LEE FIELDS, OLIVIER DEPARDEON, and THE INSPECTOR CLUZO. The 'ACTUALITÉ PAR GENRE' section shows a grid of 12 album covers: Pop (Emeli Sande), Hip-Hop (DJ Lordjazz), Rock (The Inspector Cluzo), Chanson française (Aliose), Alternative (Phantogram), Dance (Various Artists), Metal (Soufly), Electro (Carbon Airways), RnB / Soul, Jazz, Bandes Originales, and Musiques du Monde. On the right, the 'DEEZER ACTU' section features two promotional banners: 'VOTEZ POUR LE PRIX ADAMI DEEZER DE TALENTS' and 'JOUEZ ET GAGNEZ DES PLACES' for the 'Printemps de Bourges' festival. At the bottom right, there's a Facebook link for Deezer with 580,558 likes.

DEEZER Actualités Ma musique Radio Offres Premiums Découvrir Deezer Créer un compte S'identifier S'identifier avec Facebook

Sleeper - Dan San

Recherchez un titre, un album, ...

NADÉAH

GAGNEZ 50X2 PLACES POUR SON CONCERT À LA CIGALE DE PARIS LE 2 AVRIL

JOUER !

Écouter l'album

GROUNDTATION **NADÉAH** SPOEK MATHAMBO LEE FIELDS OLIVIER DEPARDEON THE INSPECTOR CLUZO

ACTUALITÉ PAR GENRE

Pop Hip-Hop Rock Chanson française

Emeli Sande DJ Lordjazz The Inspector Cluzo Aliose

Alternative Dance Metal Electro

PHANTOGRAM Various Artists Soufly CARBON AIRWAYS

RnB / Soul Jazz Bandes Originales Musiques du Monde

DEEZER sur Facebook 580,558

DEEZER

Printemps de Bourges 24-29 AVRIL 2012

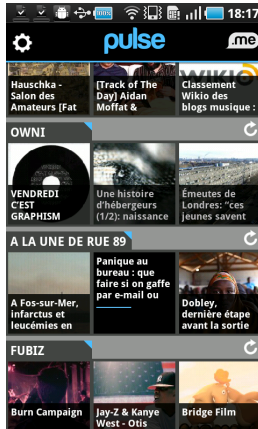
JOUEZ ET GAGNEZ DES PLACES

THE MILLION DOLLAR HOTEL

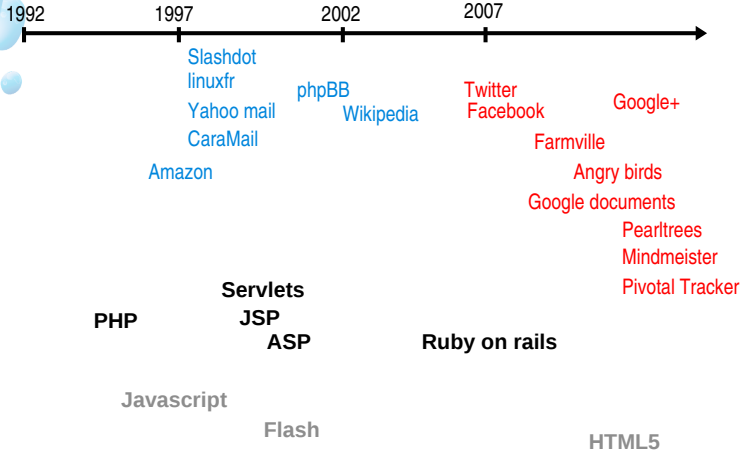
LA MÊME BOUTIQUE

igen

L'évolution du Web



L'évolution du Web



Qu'est-ce qu'Ocsigen ?

Un framework Web pour :

- les applications **HTML5** avec beaucoup d'interactions **client / serveur**
- les sites **Web** traditionnels (signets, bouton back, liens, etc)
- Les deux ensemble (le programme client ne s'arrête pas pendant la navigation)

Ocsigen est distribué sous licence LGPL

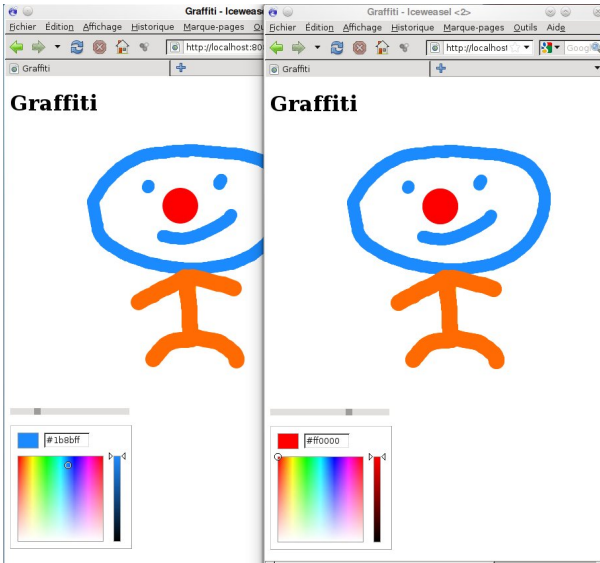
Exemple

<http://ocsigen.org/graffiti>

Exemple : une application de dessin



Exemple : une application de dessin collaborative



Le code de *Graffiti* en entier

```
{shared{
  open Elion_pervasive
  open HTML5_R
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_bus.create ~name:"graff" `Json.t<messages>

(client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##strokeWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
  })

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
  (fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d_) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(100.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages) Elion_bus.t = `b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedown Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)])) ());

  ));

Lwt.return
html
  (head
    (title (pcdata "Graffiti"))
    [link ~rel:["Stylesheet"] ~href:(uri_of_string "/css/style.css") ();
     script ~a:[a_src (uri_of_string "/graffiti_closure.js")] (pcdata "")];)
  (body [h1 (pcdata "Graffiti")]))
```

Le code de *Graffiti* en entier

```
{shared{
  open Elion_pervasive
  open HTML5_R
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_bus.create ~name:"graff" `Json.t<messages>

(client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##strokeWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
(fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html_2d_) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(100.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages) Elion_bus.t = `b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedown Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) (););

  ));

Lwt.return
html
(head
  (title (pcdata "Graffiti"))
  [link ~rel:[ "stylesheet" ] ~href:(uri_of_string "/css/style.css") ();
  script ~a:[a_src (uri_of_string "/graffiti_closure.js")] (pcdata "")];)
(body [h1 (pcdata "Graffiti")])])
```

► Court

Le code de *Graffiti* en entier

```
{shared{
  open Elion_pervasive
  open HTML5_R
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_bus.create ~name:"graff" `Json.t<messages>

(client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##strokeWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
  })

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
  (fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html_2d_) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(100.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages) Elion_bus.t = `b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedown Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))));

  ));

Lwt.return
  (html
    (head
      (title (pcdata "Graffiti"))
      [link ~rel:["Stylesheet"] ~href:(uri_of_string "/css/style.css") ();
       script ~a:[a_src (uri_of_string "/graffiti_closure.js")] (pcdata "")]);
    (body [h1 (pcdata "Graffiti")])))
```

► Court

► Un seul code

Le code de *Graffiti* en entier

```
{shared(  
  new Elion_pervasive  
  new HTML5.R  
  let width = 700  
  let height = 400  
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)  
})  
  
module My_appl = Elion_output.Elion_appl (struct  
  let application_name = "Graffiti"  
end)  
  
let b = Elion_bus.create ~name:"graff" `Json.t<messages>  
  
(client{  
  open Event_arrows  
  let draw ctx (color, size, (x1, y1), (x2, y2)) =  
    ctx##strokeStyle <- (Js.string color);  
    ctx##strokeWidth <- float size;  
    ctx##beginPath();  
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);  
    ctx##stroke()  
  })  
  
let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit  
{fun () -> Elion_services.onload  
  
  (( let canvas = Dom_html.createCanvas Dom_html.document in  
    let ctx = canvas##getContext (Dom_html._2d) in  
    canvas##width <- width; canvas##height <- height;  
    ctx##lineCap <- Js.string "round";  
    Dom.appendChild Dom_html.document##body canvas;  
  
    let slider = jsnew Goog.Ui.slider(Js.null) in  
    slider##setMinimum(1.); slider##setMaximum(80.);  
    slider##render(Js.some Dom_html.document##body);  
  
    let pSmall = jsnew Goog.Ui.hsvPalette  
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in  
    pSmall##render(Js.some Dom_html.document##body);  
  
    let x = ref 0 and y = ref 0 in  
    let set_coord ev =  
      let x0, y0 = Dom_html.elementClientPosition canvas in  
      x := ev##clientX - x0; y := ev##clientY - y0 in  
    let compute_line ev =  
      let oldx = !x and oldy = !y in  
      set_coord ev;  
      let color = Js.to_string (pSmall##getColor()) in  
      let size = int_of_float (Js.to_float (slider##getValue())) in  
      (color, size, (oldx, oldy), (!x, !y))  
    in  
    let (b : messages Elion_bus.t) = `b in  
    let line ev =  
      let v = compute_line ev in  
      let _ = Elion_bus.write b v in  
      draw ctx v  
    in  
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));  
    ignore (run (mousedown canvas  
      (arr (fun ev -> set_coord ev; line ev)  
      >>> first [mousedown Dom_html.document (arr line);  
        mouseup Dom_html.document >>> (arr line)]))));  
  
  ));  
  
Lwt.return  
html  
  (head  
    (title (pcdata "Graffiti"))  
    [link ~rel:[ "stylesheet" ] ~href:(uri_of_string "/css/style.css") ();  
      script ~a:[a_src (uri_of_string "/graffiti_closure.js")] (pcdata "")];)  
  (body [h1 (pcdata "Graffiti")])
```

► Court

► Un seul code

► côté server

► code partagé

► code client , compilé vers JS

Une application Web client/server comme un seul programme

Avantages :

- Même langage
- Mêmes structures de données — pas besoin de conversions
- Code partagé
- Générer des portions de la page soit sur le server (indexation...) soit le client (mises à jour)
- Appels de fonctions distants
- Utiliser des variables serveur depuis le côté client
- ...

Les buts d'Ocsigen

Expressivité

- Expressivité du langage lui-même
- Concepts de haut niveau adaptés aux besoins des développeurs Web.

Sécurité

- Garanties de sécurité offertes par le langage ou par Ocsigen (pas d'injection de code possible,...)

Améliorer la fiabilité, Réduire le temps de débogage

Grâce à un **langage compilé, avec un typage statique puissant**

- Le HTML respecte les recommandations du W3C (Vérifié à la compilation !)
- URLs générées dynamiquement $\Rightarrow$ pas de liens morts
- ...



OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion



OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion



OCaml

- Très expressif: fonctionnel, orienté objet, modules paramétrés ...
- Système de types très riche (typage statique !)
- Langage compilé (rapide)
- Syntaxe extensible
- Nombreuses bibliothèques et bindings
- Communauté d'utilisateurs, utilisateurs industriels

Quelques utilisateurs d'OCaml



Jane Street Capital

Transactions financières

Xen -- Citrix

15% des machines virtuelles, dont Amazon

Airbus

OCaml Labs (Cambridge UK)

Ocamlpro (Paris)

La plupart des **prouveurs de programmes**

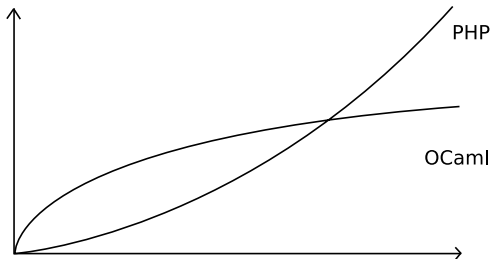
Microsoft (F#), Dassault,...

Facebook

...

Influence du typage sur le temps de développement

Development time



Complexity of the program

Un compilateur d'OCaml vers JS.

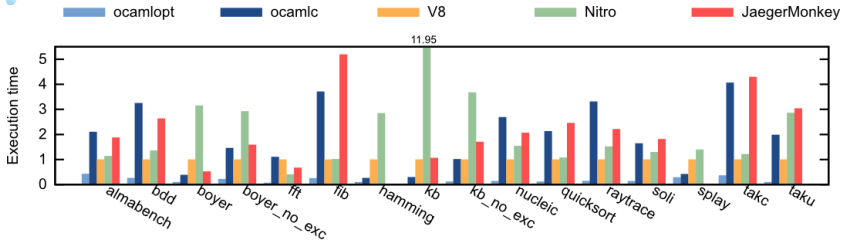
Un compilateur de **bytecode** OCaml
vers JS.

Un compilateur de **bytecode** OCaml vers JS.

Très facile à utiliser

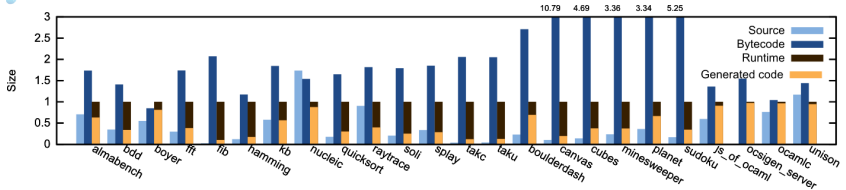
Possibilité d'utiliser des bibliothèques déjà compilées (même non libres)

Un compilateur de **bytecode** OCaml vers JS.



Rapide !

Un compilateur de **bytecode** OCaml vers JS.



Concis !

Js_of_ocaml : Appel des fonctions JS

```
{shared|
  name: Elion_pervasives
  name: HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"graff" Json.t::messages)

(client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
(fun () -> Elion_services.onload

  [( let canvas = Dom.html.createCanvas Dom.html.document in
    let ctx = canvas##getContext (Dom.html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom.html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom.html.document##body);

    let psmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    psmall##render(Js.some Dom.html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom.html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (psmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mouseDowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemoves Dom.html.document (arr line);
          mouseup Dom.html.document >>> (arr line)]))) ();
  ));

  Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[" Stylesheet "] ~href:(uri_of_string ".css/style.css") ();
       script ~src:(uri_of_string ".js/graffiti_oclosure.js")] (pcdata "");])
    (body [hi (pcdata "Grafitti")])))
```

Js_of_ocaml : Appeler des fonctions JS

```

shared{
  open Elixir_pervasive
  open HTML.M
  let width = 400
  let height = 700
  type messages = (string * int * (int * int) * (int * int)) deriving (Show)
}

module My_app1 = Elixir
  import Elixir
  let application_name = "groffiti"
end

let b = Elixir.bus.create

{client{
  open Event_arrow
  let draw ctx (color, size, x0, y0, x1, y1) => {
    ctx##stroke width <- float size
    ctx##beginPath()
    ctx##moveTo(float x0, float y0)
    ctx##stroke()
  }}

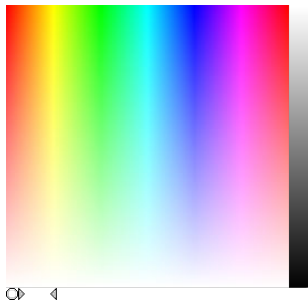
let main service = My_app1
  (fun () -> Elixir {
    (( let canvas = Dom_html.createCanvas Dom_html.document
      let ctx = canvas.getContext()
      canvas##width <- width
      canvas##height <- height
      ctx##lineCap <- "round"
      Dom_html.appendChild Dom_html.document canvas
    let slider = jsnew Google.UI.slider(3, null) in
      slider##setMin(0)
      slider##setMax(100)
      slider##render()
    let pSmall = jsnew Google.UI.huePalette
      (3, null, 3, null, 15, time (3, string "google-hsv-palette-sm")) in
      pSmall##render()
    let x = ref 0 and y = ref 0 in
      set_coord ev => {
        let x0, y0 = Dom_html.elementClientPosition canvas in
          x := ev##clientX - x0; y := ev##clientY - y0 in
        let compute_line ev =
          let color = 1x and only = 1y in
            set_coord ev;
            let color = 3x.to_string (pSmall##getColor()) in
              let size = int.of_float (3x.to_float (slider##getValue())) in
                (color, size, (x0, y0), (x, y))
            in
          let b = messages Elixir.bus -> 3x in
            let line ev =
              let v = compute_line ev in
                let _ = Elixir.bus.write b v in
                  draw ctx v
            in
            ignore (Lwt_stream.iter (draw ctx) (Elixir.bus.stream b));
            ignore (run (mousemoves canvas
              (arr (fun ev -> set_coord ev; line ev)
                >>> first [mousemoves Dom_html.document (arr line);
                  mouseup Dom_html.document >>> (arr line)]))) ();
          ))
      Lwt.return
      (head
        (title (pcdata "Groffiti"))
        (link <rel="stylesheet" href=(uri_of_string "/css/style.css") ();
          script <src={js_of_string "/groffiti_closure.js"} (pcdata "true");
          body [H1 [pcdata "Groffiti"]]))
      )
    }
  )
}

```

```
let draw ctx (color, size, (x1, y1), (x2, y2)) =  
  ctx##strokeStyle <- color;  
  ctx##lineWidth <- float size;  
  ctx##beginPath();  
  ctx##moveTo(float x1, float y1);  
  ctx##lineTo(float x2, float y2);  
  ctx##stroke()
```

Dessiner des lignes

O'Closure: un binding pour la bibliothèque de widgets Google Closure





OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion

Structure

```
(shared)
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"graff" Json.t<messages>

{client
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##strokeWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
  }

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
(fun () () -> Elion_services.onload

  {([ let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d_) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemove Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)])) ());
  ]));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "Stylesheet " ] ~href:(uri_of_string "/css/style.css") ();
     script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "");])
  (body [h1 [pcdata "Grafitti"]])])
```

Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    (link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ())
    (script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata ""));)
  (body [h1 [pcdata "Grafitti"]])))
```

Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    (link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ())
    (script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata ""));)
  (body [h1 [pcdata "Grafitti"]])))
```

Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemoves Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    (link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ())
    (script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata ""));)
  (body [h1 (pcdata "Grafitti")])))
```

Structure

open, constantes, type

```
{shared}
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
{fun () () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

  Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[ "stylesheet " ~href:(uri_of_string ".css/style.css") ();
        script ~a:[a_src (uri_of_string ".graffitti_oclosure.js")] (pcdata "")];])
    (body [h1 [pcdata "Grafitti"]])))
```

Structure

open, constantes, type

Initialisation de l'appl

```
{shared}
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
{fun () () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemoves Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

  Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
       script ~a:[a_src (uri_of_string "/graffitti_oclosure.js")] (pcdata "")];)
    (body [h1 [pcdata "Grafitti"]])))
```

Structure

open, constantes, type

Initialisation de l'appl
Bus

```
{shared}
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
{fun () () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

  Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
       script ~a:[a_src (uri_of_string "/graffitti_oclosure.js")] (pcdata "")];)
    (body [h1 [pcdata "Grafitti"]])))
```

Structure

open, constantes, type

Initialisation de l'appl
Bus

fonction draw

```
{shared}
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t:messages

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
{fun () () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

  Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[ "stylesheet " ] ~href:(uri_of_string "./css/style.css") ();
        script ~a:[a_src (uri_of_string "./graffitti_oclosure.js")] (pcdata "")]);
    (body [h1 [pcdata "Grafitti"]])))
```

Structure

open, constantes, type

Initialisation de l'appl
Bus

fonction draw

Service

```
(shared)
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t:messages

(client)
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovevs Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
  (html
    (head
      (title (pcdata "Grafitti"))
      [link ~rel:[ "stylesheet " ~href:(uri_of_string ".css/style.css") ();
        script ~a:[a_src (uri_of_string ".graffitti_oclosure.js")] (pcdata "")];])
    (body [h1 (pcdata "Grafitti")])))
```

Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = Jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = Jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedown Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
     script ~a:[a_src (uri_of_string "/graffiti_oclosure.js")] (pcdata "")];)
  (body [h1 (pcdata "Grafitti")])])
```

HTML5 page

Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovev Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
    script ~a:[a_src (uri_of_string "/graffitti_oclosure.js")] (pcdata "")];)
  (body [h1 (pcdata "Grafitti")]))])
```

Code spécifique à la page

HTML5 page

Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = Jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = Jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x <- ev##clientX - x0; y <- ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovev Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
     script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "")];)
  (body [h1 [pcdata "Grafitti"]])))
```

Canvas

Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = Jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = Jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x <- ev##clientX - x0; y <- ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovev Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

  Lwt.return
    (html
      (head
        (title (pcdata "Grafitti"))
        [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
         script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "")];)
      (body [h1 [pcdata "Grafitti"]])))
```

Canvas

Slider

Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = Jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = Jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovev Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
     script ~a:[a_src (uri_of_string "/graffitti_oclosure.js")] (pcdata "")];)
  (body [h1 [pcdata "Grafitti"]])))
```

Canvas

Slider

Palette de couleurs

Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemovev Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "./css/style.css"] ();
    script ~a:[a_src (uri_of_string ".graffitti_oclosure.js")] (pcdata "");])
  (body [h1 (pcdata "Grafitti")])])
```

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

Structure

```
{shared}
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
{fun () () -> Eliom_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let v = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mouseevents canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemoves Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)])) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
    script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "");])
  (body [h1 [pcdata "Grafitti"]]))
```

Canvas

Slider

Palette de couleurs

Fonction
de calcul des coordonnées
et envoi au serveur

Réactions aux év. serveur

Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let v = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mouseevents canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first (mousemoves Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
     script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "");])
  (body [h1 (pcdata "Grafitti")])])
```

Canvas

Slider

Palette de couleurs

Fonction

de calcul des coordonnées
et envoi au serveur

Réactions aux évé. serveur
Evé. souris

Structure

```
(shared)
open Eliom_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t:messages

{client
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
  }

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  {fun () () -> Eliom_services.onload
    (( let canvas = Dom_html.createCanvas Dom_html.document in
      let ctx = canvas##getContext (Dom_html._2d) in
      canvas##width <- width; canvas##height <- height;
      ctx##lineCap <- Js.string "round";
      Dom.appendChild Dom_html.document##body canvas;

      let slider = jsnew Goog.UI.slider(Js.null) in
      slider##setMinimum(1.); slider##setMaximum(80.);
      slider##render(Js.some Dom_html.document##body);

      let pSmall = jsnew Goog.UI.hsvPalette
        (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
      pSmall##render(Js.some Dom_html.document##body);

      let x = ref 0 and y = ref 0 in
      let set_coord ev =
        let x0, y0 = Dom_html.elementClientPosition canvas in
        x := ev##clientX - x0; y := ev##clientY - y0 in
      let compute_line ev =
        let oldx = !x and oldy = !y in
        set_coord ev;
        let color = Js.to_string (pSmall##getColor()) in
        let size = int_of_float (Js.to_float (slider##getValue())) in
        (color, size, (oldx, oldy), (!x, !y))
      in
      let (b : messages Eliom_bus.t) = b in
      let line ev =
        let v = compute_line ev in
        let ~ = Eliom_bus.write b v in
        draw ctx v
      in
      ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
      ignore (run (mouseevents canvas
        (arr (fun ev -> set_coord ev; line ev)
          >>> first [mousemoves Dom_html.document (arr line);
            mouseup Dom_html.document >>> (arr line)])) ());
    ));

  Lwt.return
    (html
      (head
        (title (pcdata "Grafitti"))
        [link ~rel:[ "stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
        script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "")];)
      (body [h1 (pcdata "Grafitti")])))
```

open, constantes, type

Initialisation de l'appl
Bus

fonction draw

Service

Canvas

Slider

Palette de couleurs

Fonction

de calcul des coordonnées
et envoi au serveur

Réactions aux évé. serveur
Évé. souris

HTML5 page

Ex de com. client-serveur : le bus

```
{shared}
new Elion_pervasive
new HTML5.M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Show)
})

module My_appl = Elion_output.Elion
  let application_name = "graffiti"
end

let b = Elion_bus.create ~name:"graff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
})

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
(fun () -> Elion_services.onload

[
  let canvas = Dom_html.createCanvas Dom_html.document in
  let ctx = canvas##getContext (Dom_html.2d) in
  canvas##width <- width; canvas##height <- height;
  ctx##lineCap <- Js.string "round";
  Dom.appendChild Dom_html.document##body canvas;

  let slider = jsnew Goog.Ui.slider(Js.null) in
  slider##setMinimum(1.); slider##setMaximum(80.);
  slider##render(Js.some Dom_html.document##body);

  let pSmall = jsnew Goog.Ui.hsvPalette
    (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
  pSmall##render(Js.some Dom_html.document##body);

  let x = ref 0 and y = ref 0 in
  let set_coord ev =
    let x0, y0 = Dom_html.elementClientPosition canvas in
    x := ev##clientX - x0; y := ev##clientY - y0 in
  let compute_line ev =
    let oldx = !x and oldy = !y in
    set_coord ev;
    let color = Js.to.string (pSmall##getColor()) in
    let size = int of float (Js.to.float (slider##getValue())) in
    (color, size, (oldx, oldy), (!x, !y))
  in
  let (b : messages Elion_bus.t) = b in
  let line ev =
    let v = compute_line ev in
    let _ = Elion_bus.write b v in
    draw ctx v
  in
  ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
  ignore (run (mousedown canvas
    (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousemove Dom_html.document (arr line);
    mouseup Dom_html.document >>> (arr line)])) ());
]);

Lwt.return
  (html
    (head
      (title (pcdata "graffiti"))
      (script (pcdata "
        // JavaScript code for the graffiti application
        // ... (omitted) ...
      ")
    )
  )
)
```

let b =

Elion_bus.create ~name:"graff" Json.t<messages>

Ex de com. client-serveur : le bus

```
{shared|
  open Elion_pervasives
  name HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Show)
})

module My_appl = Elion_output.Elion
  let application_name = "graffiti"
end

let b = Elion_bus.create ~name:"graff" Json.t<messages>

{client|
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx#strokeStyle <- (Js.string color);
    ctx#lineWidth <- float size;
    ctx#beginPath();
    ctx#moveTo(float x1, float y1); ctx#lineTo(float x2, float y2);
    ctx#stroke()
  })

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
  (fun () -> Elion_services.onload

  {
    let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas#getContext (Dom_html.2d) in
    canvas#width <- width; canvas#height <- height;
    ctx#lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document#body canvas;

    let slider = jsnew (Js.null, 0) (fun () -> slider) in
    slider#setMinValue (0); slider#setMaxValue (100);
    slider#render(Dom_html.document#body);

    let pSmall = jsnew (Js.null, 1, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall#render(Dom_html.document#body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev#clientX - x0; y := ev#clientY - y0 in
    let compute_line ev =
      let oldx = x and oldy = y in
      set_coord ev;
      let color = Js.to.string (pSmall#getColor()) in
      let size = int of float (Js.to.float (slider#getValue())) in
      (color, size, (oldx, oldy), (x, y))
    in
    let (b : messages Elion_bus.t) = b in
    let line =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemove Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());
  });

  Lwt.return
    (html
      (head
        (title (pcdata "Grafitti"))
        (script (pcdata "
          (function() {
            // Load the client-side JavaScript code
            // ... (omitted) ...
          })
        ))
      )
    )
  )
}
```

let b =

Elion_bus.create ~name:"graff" Json.t<messages>

Elion_bus.write %b v

Ex de com. client-serveur : le bus

```

{shared
  open Ellom_pervasive
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (stop
1}

```

```
let b =
```

```
Elion_bus.create ~name:"graff" Json.t<messages>
```

```
let b = EltonBus.create ~name:"groff" !son.t<messages>
```

```

client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (.35, string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}

```

```
let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
  (fun () () -> Elion_services.onload
```

```
(( let canvas = Dom_html.createCanvas Dom_html.document in
  let ctx = canvas##getContext (Dom_html._2d_) in
  canvas##width <- width; canvas##height <- height;
  ctx##lineCap <- Js.string "round";
  Dom.appendChild Dom_html.document##body canvas;
```

```
let slider = jsnew GrowableSlider(35, null) in
slider##setMinimum(1)
slider##render()
```

```
let pSmall = jsnew Goog.Ui.hsvPalette
  (Js.null, Js.all, Js.some (Js.string "goog-hsv-palette-sm")) in
pSmall##render(= some Dom.html.document##body);
```

```
let x = ref 0
let set_coord v =
    let x, y = Dom_html.elementClientPosition canvas in
    x := ev##clientX - x0; y := ev##clientY - y0 in
let compute = fun ev ->
    let oldx, oldy = x and oldy = ly in
    set_coord v;
    let color = Js.to_string (pSmall##getColor()) in
    let size = int of_float (Js.to_float (slider##getValue)) in
```

```
in
let (b : message) = Lwt_stream.i
let line =
let v = comp ~line ev in
let _ = Elv105.write b v in
```

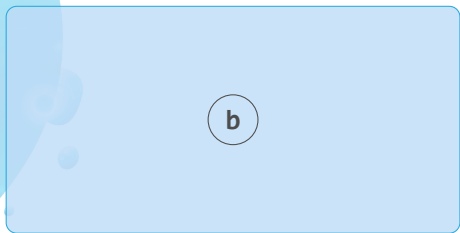
```
draw ctx
1s
ignore (let stream (iter (draw ctx) (Elion_bus.stream b)));
ignore (run (mousemoves canvas
  (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousemoves Dom_html.document (arr line);
      mouseup Dom_html.document >>> (arr line)])) ());
```

```
Lwt.return
(html
  (head
    (title (pdata "Graffiti"))
    (link rel "stylesheet" href(url of shofar fontstyle css) (
```

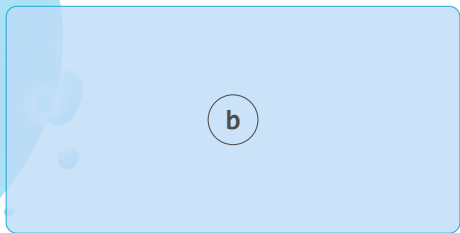
```
1  ... slider(3s, null) in
2  ... slider#bus.write %b v
```

```
Lwt_stream.iter (draw ctx) (Eliom_bus.stream %b)
```

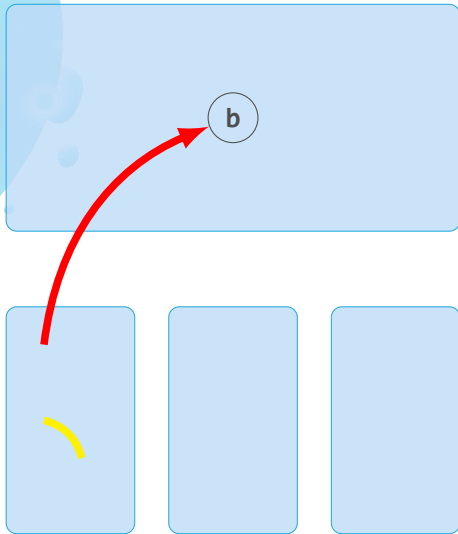
Bus client-server



Bus client-server

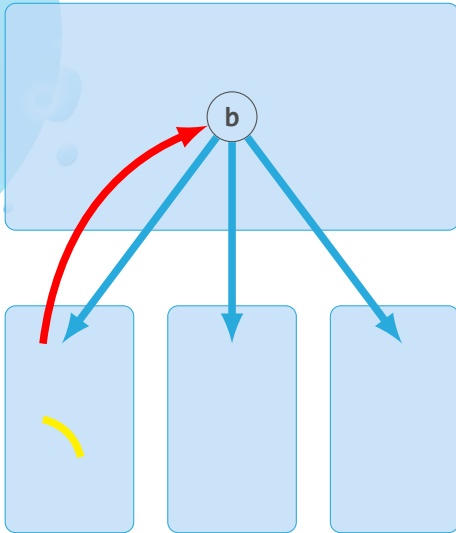


Bus client-server



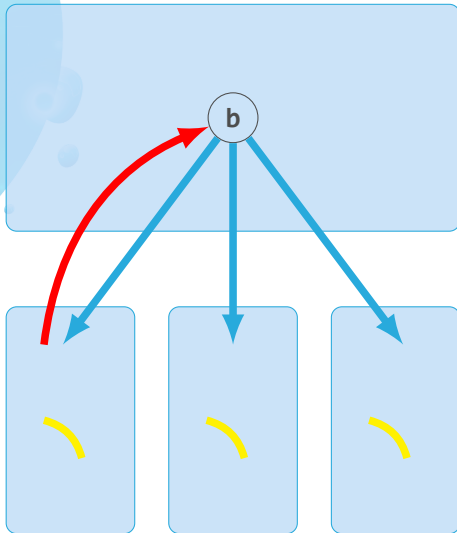
call **write**

Bus client-server



listen on bus **b**

Bus client-server



listen on bus **b**



OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion

Services

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}
```

```
module My_appl = Elion_appl
let application_name = "graffiti"
end

let b = Elion_bus.create ~name:"graff" Json.t.messages

{client
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- Js.string color;
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}
```

```
let main_service = My_appl.register_service ~path:"/" ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.unload
```

```
(( let canvas = Dom.html.createCanvas Dom.html.document in
  let ctx = canvas.getContext (Dom.html._2d) in
  canvas##width <- width; canvas##height <- height;
  ctx##lineCap <- Js.string "round";
  Dom.appendChild Dom.html.document#body canvas;
```

```
let slider = Jsnew Goog.UI.slider(Js.null) in
slider##setMinimum(1.); slider##setMaximum(80.);
slider##render(Js.some Dom.html.document#body);
```

```
let pSmall = Jsnew Goog.UI.hsvPalette
(Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
pSmall##render(Js.some Dom.html.document#body);
```

```
let x = ref 0 and y = ref 0 in
let set_coord ev =
  let x0, y0 = Dom.html.elementClientPosition canvas in
  x <- ev##clientX - x0; y <- ev##clientY - y0 in
let compute_line ev =
  let oldx = !x and oldy = !y in
  set_coord ev;
  let color = Js.to_string (pSmall##getColor()) in
  let size = int_of_float (Js.to_float (slider##getValue())) in
  (color, size, (oldx, oldy), (!x, !y))
```

```
in
let (b : messages Elion_bus.t) = %b in
let line ev =
  let v = compute_line ev in
  let _ = Elion_bus.write b v in
  draw ctx v
```

```
in
ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
ignore (run (mousedowns canvas
  (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousemove Js.some Dom.html.document (arr line);
      mouseup Dom.html.document >>> (arr line)]))) ());
```

```
});

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "stylesheet " ] ~href:(uri_of_string "./css/style.css") ();
     script ~a:[a_src (uri_of_string "./graffitti_closure.js")] (pcdata "")];)
  (body [h1 (pcdata "Grafitti")]))
```

let main_service = register_service

~path:[""]

~get_params:unit

(fun () () -> ...)

```
(shared)
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
))

module My_app1 = Elion_app1
let application_name = "My_app1"
end

let b = Elion_bus.create ~name:"greff" Json.t.messages

{client
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- (Js.float size);
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}

let main_service = My_app1.register_service ~path:"/" ~get_params:Elion_parameters.unit
  (fun () () -> Elion_services.onload

    (( let canvas = Dom_html.createCanvas Dom_html.document.in
        let ctx = canvas.getContext (Dom_html_id "c") in
        canvas##width <- width; canvas##height <- height;
        ctx##lineCap <- Js.string "round";
        Dom_html.appendChild Dom_html.document#body canvas;

        let slider = Jsnew Goog.UI.slider(Js.null) in
        slider##setMinimum(1.); slider##setMaximum(80.);
        slider##render(Js.some Dom_html.document#body);

        let pSmall = Jsnew Goog.UI.hsvPalette
          (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
        pSmall##render(Js.some Dom_html.document#body);

        let x = ref 0 and y = ref 0 in
        let set_coord ev =
          let x0, y0 = Dom_html.elementClientPosition canvas in
          x := ev##clientX - x0; y := ev##clientY - y0 in
        let compute_line ev =
          let oldx = !x and oldy = !y in
          set_coord ev;
          let color = Js.to_string (pSmall##getColor()) in
          let size = int_of_float (Js.to_float (slider##getValue())) in
          (color, size, (oldx, oldy), (!x, !y))
        in
        let (b : messages Elion_bus.t) = b in
        let line ev =
          let v = compute_line ev in
          let _ = Elion_bus.write b v in
          draw ctx v
        in
        ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
        ignore (run (mousedown canvas
          (arr (fun ev -> set_coord ev; line ev)
            >>> first [mousedown Dom_html.document (arr line);
              mouseup Dom_html.document >>> (arr line)]))) ());

    ));

Lwt.return
  (html
    (head
      (title (pcdata "Greffit"))
      [link ~rel:[ "Stylesheet " ] ~href:(uri_of_string "./css/style.css") ();
        script ~a:[a_src (uri_of_string "./greffit_oclosure.js")] (pcdata "")];
      (body [h1 [pcdata "Greffit"]])))
```

let main_service = register_service

~path:[""]

~get_params:unit

(fun () () -> ...)

- HTML
- Fichier
- Redirection
- Action
- Application
- ...

Programmer l'interaction Web traditionnelle avec les services d'Ocsigen

- 1 Les services sont des valeurs de première classe
- 2 Liens et formulaires sont vérifiés statiquement
- 3 Les données générées sont vérifiés statiquement
- 4 Création dynamique de nouveaux services
- 5 Mécanisme puissant d'identification de services, basé sur :
 - L'URL ou un identifiant de service (comme paramètre)
 - Le méthode HTTP (GET or POST)
 - Le nom des paramètres
 - La session (navigateur)
 - L'onglet qui fait la requête
 - ...

service : (GET and POST parameters) $\longrightarrow$ page

Ocsigen vérifie la présence des paramètres
et les traduit automatiquement vers des types OCaml
(integers, booleans,... et même des ensembles ou listes !)

Liens et formulaires

Les services sont des *valeurs de première classe*

Liens et formulaires

Les services sont des *valeurs de première classe*

`<a>` ne prend pas l'*URL* comme paramètre,
mais le *service* !

⇒ *Pas de liens cassés !*

Liens et formulaires

Les services sont des *valeurs de première classe*

`<a>` ne prend pas l'*URL* comme paramètre,
mais le *service* !

⇒ *Pas de liens cassés !*

*Conformité des formulaires par rapport aux services
vérifiée à la compilation*

Mécanisme d'identification de services

Ocsigen choisit le service automatiquement

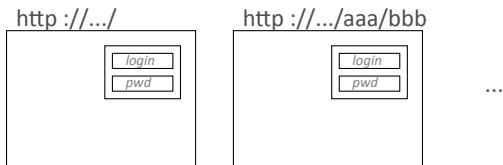
en fonction :

- De l'URL et/ou d'un paramètre interne
- De la méthode HTTP (GET ou POST)
- Du nom des paramètres
- La session de l'utilisateur
- ...

—→ Le programmeur choisit le style d'identification dont il a besoin.

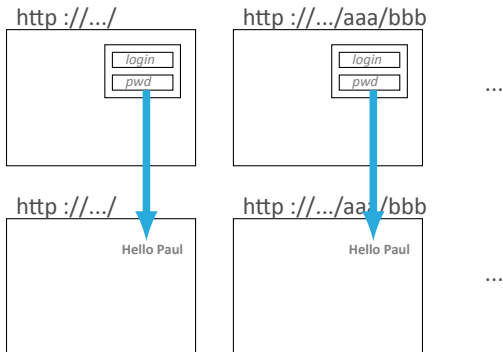
Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page



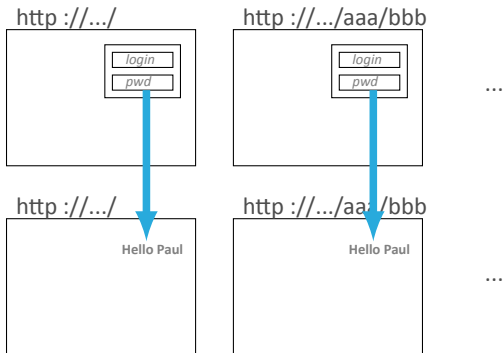
Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page



Mécanisme d'identification de services

Exemple : connection d'utilisateur depuis n'importe quelle page



L'action de « se connecter » est un service *caché* disponible pour toutes les URLs.

Deux sortes d'interaction Web

Exemple : réserver un billet d'avion

Ocsitravel

Fichier Édition Affichage Historique Marque-pages Outils Aide

Ocsitravel

Votre compte client Vos réservations Votre panier : 74.10€ Aide

OCSI TRAVEL

ACCUEIL Bienvenue TRAIN France, Europe VOL, VOITURE Ferry HÔTEL Location, Camping SÉJOURS Week-end, Loire BONS PLANS Dernière Minute VOYAGINE Reportages, vidéos

Accueil > Votre billet de train

Simplifiez vos réservations. Connectez-vous ou Créez votre compte.

Où et quand souhaitez-vous partir ?

Voyage ☒ Aller Simple ☐ Aller-Retour

Départ Arrivée

Aller le à partir de Retour le à partir de

Qui participe à ce voyage ?

Nombre de passagers Code Offre Spéciale

Voyageur 1

Passager

Carte et Abonnement

Programme de fidélité

Comment souhaitez-vous voyager ?

Votre confort ☒ 1e classe ☐ 2e classe

Sélectionnez le pays de réception ou de retrait des billets :

HORAIRES RÉSERVER

action Web

Je veux réserver un billet
Lisbonne -- Moscou

TROUVEZ LES MEILLEURS PRIX
VERS 50 DESTINATIONS

publicité

Ocsitravel

Fichier Édition Affichage Historique Marque-pages Outils Aide

Ocsitravel

Votre compte client Vos réservations Votre panier : 74.10€ Aide

OCSI TRAVEL

ACCUEIL Bienvenue TRAIN France, Europe VOL, VOITURE Ferry HÔTEL Location, Camping SÉJOUR Week-end, Loisirs BONS PLANS Dernière Minute VOYAGINE Reportages, vidéos

Accueil > Votre billet de train

○	19/09/2011 2:14 Lisbon	1775,43€
○	19/09/2011 23:17 Moscow	
○	19/09/2011 22:14 Lisbon	1,42€
○	19/09/2011 22:17 Moscow	
○	19/09/2011 00:14 Lisbon	42,17€
○	26/09/2011 12:17 Moscow	

Terminé

Le site propose plusieurs choix

Dans une autre fenêtre,
Je veux réserver un billet
San Francisco -- Antananarivo

19/09/2011 2

19/09/2011 2

19/09/2011 0

26/09/2011 1

San Francisco -- Antananarivo

Quel est votre mode de transport ?

Voyage * ☐ Aller Simple ☒ Aller-Retour

Départ * San Francisco

Arrivée * Antananarivo

Aller le * 01/10/2010 à partir de 07h

Retour le * à partir de 07h

Qui participe à ce voyage ?

Nombre de passagers 1

Code Offre Speciale

Voyageur 1

Passager

26-59 ans

Carte et Abonnement

Programme de fidélité

Comment souhaitez-vous voyager ?

Votre confort

☒ 1e classe

☐ 2e classe

Sélectionnez le pays de réception ou de retrait des billets : FRANCE

HORAIRES

RÉSERVER

TROUVEZ LES MEILLEURS PRIX
VERS 50 DESTINATIONS

Destination	Prix
San Francisco	44.90€
Antananarivo	22.00€

Ocsit

Fichier Édition Affichage Historique Marque-pages Outils Aide

Ocsitravel

ACCUEIL
Bienvenue

TRAIN
France, Europe

Accueil > Votre billet de train

Ocsitravel

Votre compte client Vos réservations Votre panier : 74.10€ Aide

ACCUEIL
Bienvenue

TRAIN
France, Europe

VOL, VOITURE
Ferry

HÔTEL
Location, Camping

SÉJOUR
Week-end, Loisirs

BONS PLANS
Dernière Minute

VOYAGINE
Reportages, vidéos

Accueil > Votre billet de train

Encore une fois, plusieurs choix

○

19/09/2011 22:14 San Francisco
19/09/2011 22:17 Antananarivo

○

19/09/2011 00:14 San Francisco
26/09/2011 12:17 Antananarivo

○

19/09/2011 22:14 San Francisco
19/09/2011 22:17 Antananarivo

○

19/09/2011 00:14 San Francisco
26/09/2011 12:17 Antananarivo

75,43€

918,42€

42,22€

○ 19/09/2011 2:14 Lisbon
19/09/2011 23:17 Moscow

○ 19/09/2011 22:14 Lisbon
19/09/2011 22:17 Moscow

○ 19/09/2011 00:14 Lisbon
26/09/2011 12:17 Moscow

Je retourne à la première fenêtre

1,42€ Francisco Mananarivo 918,42€

42,17€ Francisco Mananarivo 42,22€

○ 19/09/2011 2:14 Lisbon
19/09/2011 23:17 Moscow

Je retourne à la première fenêtre

○ 19/09/2011 22:14 Lisbon
19/09/2011 22:17 Moscow

1,42€ Francisco 918,42€
ananarivo

○ 19/09/2011 00:14 Lisbon
26/09/2011 12:17 Moscow

42,17€ Francisco 42,22€
Je veux continuer avec le premier billet !

Ocsitravel

Fichier Édition Affichage Historique Marque-pages Outils Aide

Ocsitravel

Votre compte client Vos réservations Votre panier : 74.10€ Aide

OCSITRavel

ACCUEIL Bienvenue TRAIN France, Europe VOL, VOITURE Ferry HÔTEL Location, Camping SÉJOUR Week-end, Loisirs BONS PLANS Dernière Minute VOYAZINE Reportages, vidéos

Accueil > Votre billet de train

19/09/2011 2:14 Lisbon	1775,43€
19/09/2011 23:17 Moscow	
19/09/2011 22:14 Lisbon	1,42€
19/09/2011 22:17 Moscow	
19/09/2011 00:14 Lisbon	42,17€
26/09/2011 12:17 Moscow	

Terminé

Ocsitravel

Votre compte client Vos réservations Votre panier : 74.10€ Aide

HÔTEL Location, Camping SÉJOUR Week-end, Loisirs BONS PLANS Dernière Minute VOYAZINE Reportages, vidéos

Francisco	75,43€
ananarivo	
Francisco	918,42€
ananarivo	
Francisco	42,22€
ananarivo	

Les deux fenêtres évoluent indépendamment

19/09/2011 2:14 Lisbon	1775,43€
19/09/2011 23:17 Moscow	
Mais le panier est <i>partagé</i> par les deux fenêtres	
19/09/2011 22:14 Lisbon	
19/09/2011 22:17 Moscow	
19/09/2011 00:14 Lisbon	42,17€
26/09/2011 12:17 Moscow	

Francisco	75,43€
ananarivo	
Francisco	918,42€
ananarivo	
Francisco	42,22€
ananarivo	

Panier :

Sessions

- « État » côté serveur pour un navigateur
- Gestion automatiques des cookies de session
- Possibilité de créer des services *de session* !

Panier :

Sessions

- « État » côté serveur pour un navigateur
- Gestion automatiques des cookies de session
- Possibilité de créer des services *de session* !

Pages dépendant d'interactions précédentes :

Panier :

Sessions

- > « État » côté serveur pour un navigateur
- > Gestion automatiques des cookies de session
- > Possibilité de créer des services *de session* !

Pages dépendant d'interactions précédentes :

Création dynamique de services

- > L'histoire de l'interaction est gardée automatiquement en mémoire (côté serveur)
- > Vous pouvez retourner dans le passé (bouton ``back"")
ou changer de fenêtre de navigateur

Sessions revisitées

Données de session sauvegardées dans des *références* avec une **portée (scope)**.

Portée :

- Site
- Session de navigateur (cookie)

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Session de navigateur (cookie)
- **Processus client (onglet)**

Exemple : Si vous avez plusieurs instances d'un jeu dans plusieurs onglets, le score est une référence de portée « onglet ».

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- **Groupes de sessions (utilisateur)**
- Session de navigateur (cookie)
- Processus client (onglet)

Exemple : partager le panier entre plusieurs terminaux !

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Groupes de sessions (utilisateur)
- Session de navigateur (cookie)
- Processus client (onglet)
- **Requête**

Garder de l'information pendant la génération d'une page.

Sessions revisitées

État côté serveur mis en œuvre grâce à des *références* avec une **portée (scope)**.

Portée :

- Site
- Groupes de sessions (utilisateur)
- Session de navigateur (cookie)
- Processus client (onglet)
- Requête

Les *services* ont aussi une portée.



Web 1.0 + Web 2.0 = ?

Web 1.0 + Web 2.0 = ?

Avec Ocsigen, le programme côté client ne s'arrête pas quand vous cliquez sur un lien !

Web 1.0 + Web 2.0 = ?

Avec Ocsigen, le programme côté client ne s'arrête pas quand vous cliquez sur un lien !

- > Les applications Ocsigen client-server sont 100% compatibles avec l'interaction Web traditionnelle (signets, bouton back) !
 - > Vous pouvez garder un état côté client.
- > Certaines portions de la page peuvent rester après avoir changé de page.
- > La musique ou les vidéos ne s'arrêtent pas.

Exemple : Site de streaming vidéo

(continuer sa navigation pour choisir d'autres albums sans arrêter la musique)



OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion

Génération de HTML

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_bus.create ~name:"groff" Json.t<messages>

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##lineWidth <- float size;
  ctx##beginPath();
  ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
  ctx##stroke()
}

let main_service = My_appl.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () () -> Elion_services.onload

  (( let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.UI.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-se")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemoves Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());

  ));

Lwt.return
(html
  (head
    (title (pcdata "Grafitti"))
    [link ~rel:[ "Stylesheet " ] ~href:[uri_of_string "/css/style.css"] ();
     script ~a:[a_src (uri_of_string "/graffitti_closure.js")] (pcdata "")];)
  (body [h1 (pcdata "Grafitti")])])
```

Génération de HTML

```
{shared{
  open Elion_pervasive
  open HTML5.H
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}
```

```
module My_appl = Eliom_output.Eliom_appl (struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>
```

```
{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1) (x2, y2)) =
    ctx##strokeStyle <- [30, String]
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo (float x1, float y1); ctx##strokeOff float x2 float y2;
    ctx##stroke()
}}
```

```
let main_service = My_app.register_service ~path:("t")
  (fun () () -> Eltorm_services.unload
```

```
{  
  let canvas = Dom_html.createCanvas(Dom_html.document.in  
  let ctx = canvas.getContext(Dom_html._2d);  
  canvas.width <- width; canvas.height <- height;  
  ctx.lineWidth <- 15; ctx.strokeStyle <- "red";  
  Dom_html.appendChild(Dom_html.document.body, canvas);  
}
```

```
let slider = jsnew Goog.UI.slider(Js.null) in
slider##setMinimum(1.); slider##setMaximum(80.);
slider##render(1, some Dom.html document##body)

```

```
let pSmall = jsnew Goog.Ui.hsvPalette
  (Js.null, Js.null, Js.some (Js.string "hsv-1"))
```

```
let x = ref 0 and y = ref 0 in
let set_coord ex =
```

```

let x0, y0 = L[n].elementClientPosition to
x := ev##clientX - x0; y := ev##clientY - y0 if
let compute_line ev =

```

```
set_coord ev;
let_color = js_to_string (pSmall##getColor())
let_size = int of_float (js_to_float (slider##
```

```
in
let (b : messages Elion_bus.t) = %b in
```

```
let v = compute_line ev in
let _ = Eliom_bus.write b v in
done; ctx v
```

```

in
ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.st
ignore (run (mousedown canvas

```

```
>>> first [mousemoves Dom_html.docu
mouseup Dom_html.docu
```

```
Lwt.return
(html
```

```
(title (pcdata "Graffiti"))
[link ~rel:[ 'Stylesheet ' ] ~href:(uri_of_
  ~script ~:~?~.css ~uri_of_stylesheet ~.css~?~.css)]
```

```
(body [h1 [pcdata "Graffiti"]]))))
```

```
(html
(head
```

```
(title (pcdata "Graffiti"))
```

```
[link ~rel :[ `Stylesheet ] ~href :(uri_of_str
```

```
script ~a :[a_src (uri_of_string "./graffiti
```

```
(body [h1 [pdata "Graffiti"]]))
```

$$s.t) = x_b \text{ in}$$

```

b v in
... etc. (534m)

```

```

v -> set_coord e
mousemoves Dom_h
mouseup Dom_html

```



```
ffitf"))
sheet ] ~href:(u
uri_of_string ".
ffitf"11)))
```

Générer du HTML

La validité du HTML est vérifiée à la compilation !

Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```

Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</hmtl>
```

Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</hmtl>
```



Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</hmtl>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><title>Hello</title></body>  
</html>
```

Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</hmtl>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><title>Hello</title></body>  
</html>
```



Produire du HTML valide

```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</html>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><h1>Hello</h1></body>  
</hmtl>
```



```
<html>  
  <head><title>Hello</title></head>  
  <body><title>Hello</title></body>  
</html>
```

→ rejetés à la compilation !



Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilateur

$f : () \rightarrow \text{block list}$

• `<body> f() </body>`

Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilateur

$f : () \rightarrow \text{block list}$

`<body> f() </body>`



Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilateur

$f : () \rightarrow \text{block list}$

`<body> f() </body>`



`<p> f() </p>`

Produire du HTML valide

Les programmes pouvant générer des pages invalides sont rejetés par le compilateur

$f : () \rightarrow \text{block list}$

`<body> f() </body>`



`<p> f() </p>`





OCaml



Applications Web client-serveur



Programmer l'interaction Web traditionnelle



Générer du HTML



Conclusion



Beaucoup de fonctionnalités uniques



Beaucoup de fonctionnalités uniques

Identification de services sophistiquée



Beaucoup de fonctionnalités uniques

Identification de services sophistiquée

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Identification de services sophistiquée

Typage du HTML

Typages des liens, formulaires, paramètres



Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Services de session

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Services de session

Portée

Typages des liens, formulaires, paramètres

Beaucoup de fonctionnalités uniques

Services dynamiques

Identification de services sophistiquée

Typage du HTML

Programmation client-server unifiée

Services de session

Portée

Typages des liens, formulaires, paramètres



Beaucoup de fonctionnalités uniques

Services dynamiques

Persistence du programme client

Identification de services sophistiquée

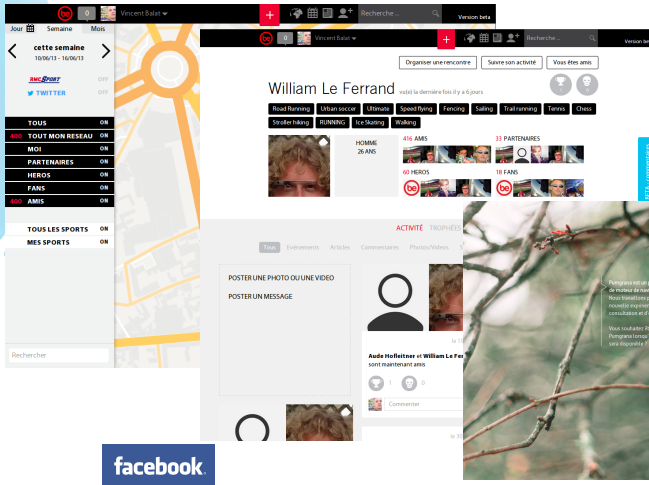
Typage du HTML

Programmation client-server unifiée

Services de session

Portée

Typages des liens, formulaires, paramètres



Some companies and open source projects :

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

Quelques-uns de nos projets

ocorigen
Next-Gen in web programming
eliom

More information

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

ocorigen
Next-Gen in web programming
js_of_ocaml

More information

An OCaml to Javascript compiler.

ocorigen
Next-Gen in web programming
lwt

More information

A cooperative threading library for OCaml.

ocorigen
Next-Gen in web programming
server

More information

A full-featured and extensible Web server.

> And many other projects ...

ocsigen.org

Logiciel libre --- Version 4.0 ce mois-ci !

Auteurs et contributeurs :

Vincent Balat, Jérôme Vouillon, Pierre Chambart, Grégoire Henry, Benedikt Becker, Raphaël Proust, Benjamin Canou, Boris Yakobowski, Jérémie Dimino
Charly Chevalier, Gabriel Radanne, Jacques-Pascal Deplaix, Stéphane Glondu, Gabriel Kerneis, Arnaud Parant, Christophe Lecointe, Denis Berthod, Gabriel Cardoso, Piero Furiesi, Jaap Boender, Thorsten Ohl, Gabriel Scherer, Séverine Maingaud, Simon Castellan, Jean-Henri Granarolo, Archibald Pontier, Nataliya Guts, Cécile Herbelin, Charles Oran, Jérôme Velleine, Pierre Clairambault ...