From OCaml to Javascript at Skydeck

jake.donham@skydeck.com

What is Skydeck?

- a tool for managing your mobile phone
- reads your mobile phone call log
- presents it back to you in a useful way
    - attach people to phone numbers
    - view calls by person
    - when did I last call Steve?
    - who did I call yesterday?
    - etc.

Where does the data come from?

- from your phone carrier's web site
    - you give Skydeck your credentials
    - we download bills and usage from carrier site
        - with a Firefox extension
        - with a standalone XULrunner app
        - from our servers (a farm of XULrunners)

- via our web API
    - 3rd party can add new data sources

Where does OCaml come into this?

- most of our system is written in OCaml
    - bill parsing, web servers, etc.

- but the web is Javascript
    - Mozilla apps are Javascript
    - Javascript is not my favorite programming language
        - too forgiving
        - heavy syntax for functional code

- sad programmers

OCamljs

- we wrote OCamljs
    - Javascript back-end for OCaml compiler
- wrote our Mozilla app in OCaml
- we are happy

Really?

```
match referer with
  | None -> r#send body
  | Some re ->
  (* see http://developer.mozilla.org/en/docs/Setting_HTTP_req ...
  let os = XPCOM.getService_observer_service () in
  let observe (s : #XPCOM.supports) _ _ =
    let hc = s#_QueryInterface XPCOM.httpChannel in
    if hc == r#_get_channel#_QueryInterface XPCOM.httpChannel
    then hc#setRequestHeader "Referer" re false in
  let observer = Ocamljs.obj [
    "observe", Ocamljs.jsfun3 observe
  ] in
  os#addObserver observer "http-on-modify-request" false;
  r#send body;
  os#removeObserver observer "http-on-modify-request";
```

Benefits of OCaml for downloader

- types types types
- can give types to the complicated Mozilla API
- continuation passing style enforced by types
- transparent RPC to server
- tool support (Camlp4, ocamlbuild)

How does OCamljs work?

- ocamlc compiles to "lambda" intermediate language
- ocamljs translates lambda to Javascript
- almost everything in the front-end comes for free
  - type checking
  - module system
  - Camlp4
- objects not free
  - we want OCaml objects = JS objects

Example

OCaml:

```
module Test =
struct
  type foo = Bar of int | Baz of bool | Quux

  let f = function
    | Bar i -> "Bar " ^ string_of_int i
    | Baz b -> "Baz " ^ (if b then "true" else "false")
    | Quux -> "Quux"
end
```

Example

```
module Test =
struct
  type foo = Bar of int | Baz of bool | Quux

  let f = function
    | Bar i -> "Bar " ^ string_of_int i
    | Baz b -> "Baz " ^ (if b then "true" else "false")
    | Quux -> "Quux"
end
```

Lambda:

```
(setglobal Test!
  (let
    (f/65
       (function param/73
         (switch* param/73
          case int 0: "Quux"
          case tag 0:
           (apply (field 15 (global Pervasives!)) "Bar "
             (apply (field 19 (global Pervasives!))
               (field 0 param/73)))
          case tag 1:
           (apply (field 15 (global Pervasives!)) "Baz "
             (if (field 0 param/73) "true" "false")))))
    (makeblock 0 f/65)))
```

Example

```
(setglobal Test!
  (let
    (f/65
       (function param/73
         (switch* param/73
          case int 0: "Quux"
          case tag 0:
           (apply (field 15 (global Pervasives!)) "Bar "
             (apply (field 19 (global Pervasives!))
               (field 0 param/73)))
          case tag 1:
           (apply (field 15 (global Pervasives!)) "Baz "
             (if (field 0 param/73) "true" "false")))))
    (makeblock 0 f/65)))
```

Javascript:

```
var oc$Test$ =
  function () {
    var f$65 =
      _f(function (param$73) {
          if (typeof param$73 == "number")
            switch (param$73) { case 0: return "Quux"; default: return ...
          else
            switch ($t(param$73)) {
            case 0:
              return __(oc$Pervasives$[15],
                ["Bar ", _(oc$Pervasives$[19], [param$73[0]])]);
            case 1:
              return __(oc$Pervasives$[15],
                ["Baz ", param$73[0] ? "true" : "false"]);
            default: return null;}
        });
    return $(f$65);
  }();
```

Gory details

- partial application / overapplication
- tail recursion via trampolines
- heap representation
    - block -> array + tag
    - int (nativeint, int32), float, char -> number
    - bool -> number, bool
        - since JS comparison ops return bool
    - string -> string, number array
        - support mutable strings

Interfacing with Javascript

- with "external" like with C
  - naming convention for methods, accessors
  - special externals for raw Javascript

- with object type
  - naming convention for accessors

- OCamljs included libraries:
  - some Mozilla API
  - some built-in Javascript
  - OCaml stdlib

Work in progress

- orpc for Javascript
    - orpc generates RPC code from OCaml signatures
        - works with Ocamlnet
    - Javascript backend passes heap rep
        - on client, just eval it
        - on server, must check that it's valid for type
- jslib
    - Camlp4 parser, pretty-printer, quotations for JS

Future work / dreams

- finish object support
  - write Javascript objects in OCaml
- use jslib to support inline Javascript in OCaml code
- improve performance
- web programming
  - like Google Web Toolkit

Using OCaml at a startup

- a good idea!
- better tools let you work faster
- static checking keeps you on course
- you get a clean slate
- you need to hire great people
    - OCaml is fun!

Thanks!

- Skydeck is hiring
    - http://skydeck.com/jobs

- http://code.google.com/p/ocamljs
- http://code.google.com/p/orpc2