Developing a Set Top Box Middleware in Erlang

Controlling Miscellaneous Stuff in an Embedded System

Samuel Rivas

LambdaStream S.L. samuel.rivas@lambdastream.com

London, 2009

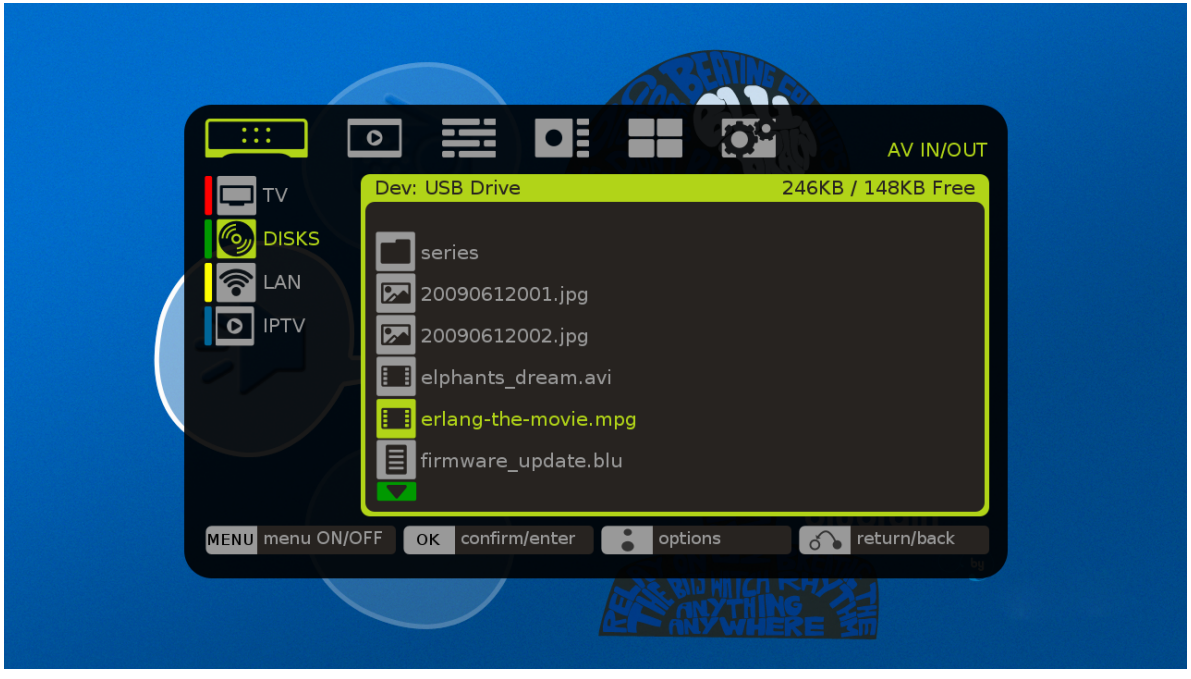
The Target System

- Home Entertainment Device, with optional hard disk and many inputs and outputs.
- DTV, local file playback, streaming, PVR
- Export and import network file systems
- User applications

Many questions

- How to interface with the user?
- How to control all the possibilities of the hardware?

Many questions



- How to control every possible concurrent task?

Many questions

- How to allow third parties to develop applications?
- How to take advantage of Internet access?

We Can Do It

You'll need a middleware to handle that many concurrent tasks.
Should we suggest Erlang?

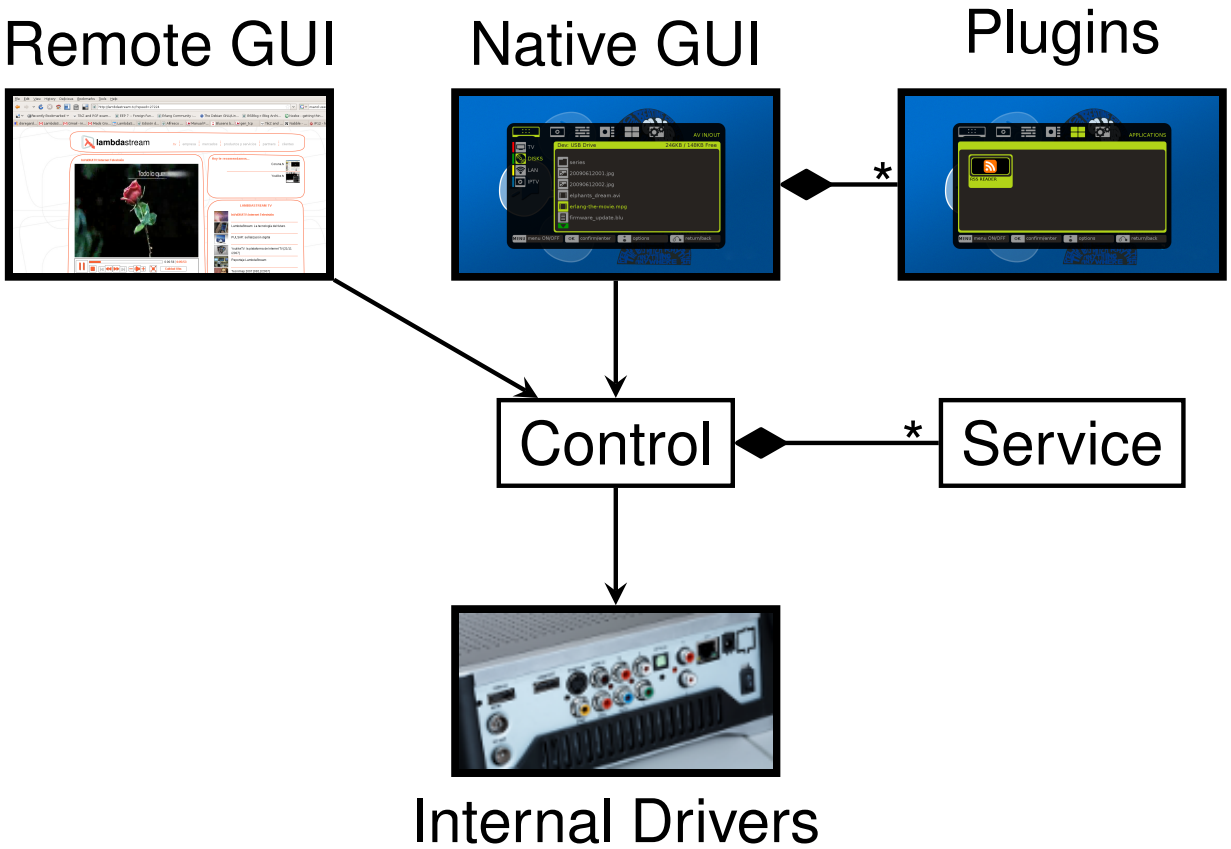
Challenges

- Crosscompile Erlang to the target platform
- Define a versatile architecture to:
  - Allow third party plugins
  - Interface low level drivers and SO services
- Build a flexible engine for the local GUI
- Find a solution to allow remote GUIs

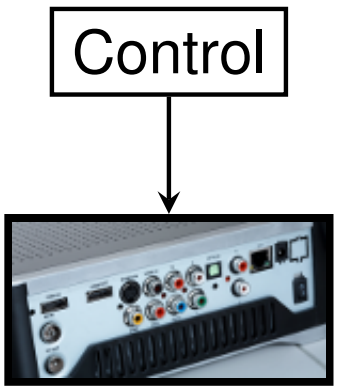
Crosscompile Erlang

- We failed crosscompiling R11B-5 for mipsel
- But we managed to crosscompile R12B-1
- Haven’t tried any harder

Architecture



Talking to C Drivers



Internal Drivers

- We cannot use linked-in drivers because some drivers require the executing process to finish to release memory and resources.
- Ports are tedious and cumbersome.
- Must handle living threads and fast, finite operations.
- EDTK and DryverI wouldn't fit with living C threads sending messages.

Yet another framework to connect C and Erlang

Generic Erlang–C Connection Framework

- Handles the communication between C and Erlang using a port.
- Delegates the specific behaviour in pluggable modules (compiled .so files).
- Generic API to call C funcions from the Erlang side.
- Each call from Erlang runs in a separate thread in the C side.
- C side can send messages back to Erlang.

Yet another framework to connect C and Erlang

Erlang Example

```
play(Url) ->
    Binary = list_to_binary(Url),
    dfb_facade:sync_call(
      [{play_url, <<Binary/binary, 0>>}]).
```

Yet another framework to connect C and Erlang

Registering C calls (simplified source)

```
void DFBERL_LOAD_FUNC(State state) {
  MethodRunner runner;

  runner = methodRunnerNew("play_url", play);
  registerRunner(state, runner);
}
```

Yet another framework to connect C and Erlang

C Example (simplified source)

```
gboolean play(Call call, Context context) {

  url = (gchar *) ERL_BIN_PTR(call -> args);

  if (av_play(url) != AV_RETURN_OK) {
    return FALSE;
  }
  sendResponse(context -> encoder, call -> id,
               erl_format("ok"));
  return TRUE;
}
```

Yet another framework to connect C and Erlang

Handling Data

The easy example

```
<<Binary/binary, 0>>

url = (gchar *) ERL_BIN_PTR(call -> args);
```

But types are seldom so simple

- Too much code to adapt C and Erlang data types.
- Too many heavyweight memory copying operations.
- Too many sanity checks.

Yet another framework to connect C and Erlang

Handling Data

- Avoid erl-interface implementing efficient, ad-hoc marshaling/unmarshaling operations for each type
- Write a code generator for that.

Type Definition

```
{transcoder_info,
 {complex, [{"Type", transcoder_type},
            {"BitRate", integer},
            {"Width", integer},
            {"Height", integer}]}}.
```

GUI

Use the erlang to C adaptation?

Drawing graphics and handling input events

- Control can access DirectFB as a driver.
- But GUI developers wouldn't use erlang calls to develop.

Experimental engine for interactive applications

- Screen descriptions using a declarative language
- Handle user input and focus management
- Incremental drawing to increase performance
- Way too much work

GUI
HTML + Javascript?

Why not use an HTML browser?

Advantages

- GUI developers are familiar with the model
- Easy to use for remote interfaces as well
- Can surf the web!

Drawbacks

- Without advanced CSS + Javascript, interfaces look aged
- Fully capable browsers are heavyweight
- Devised for mice, not for remote controllers

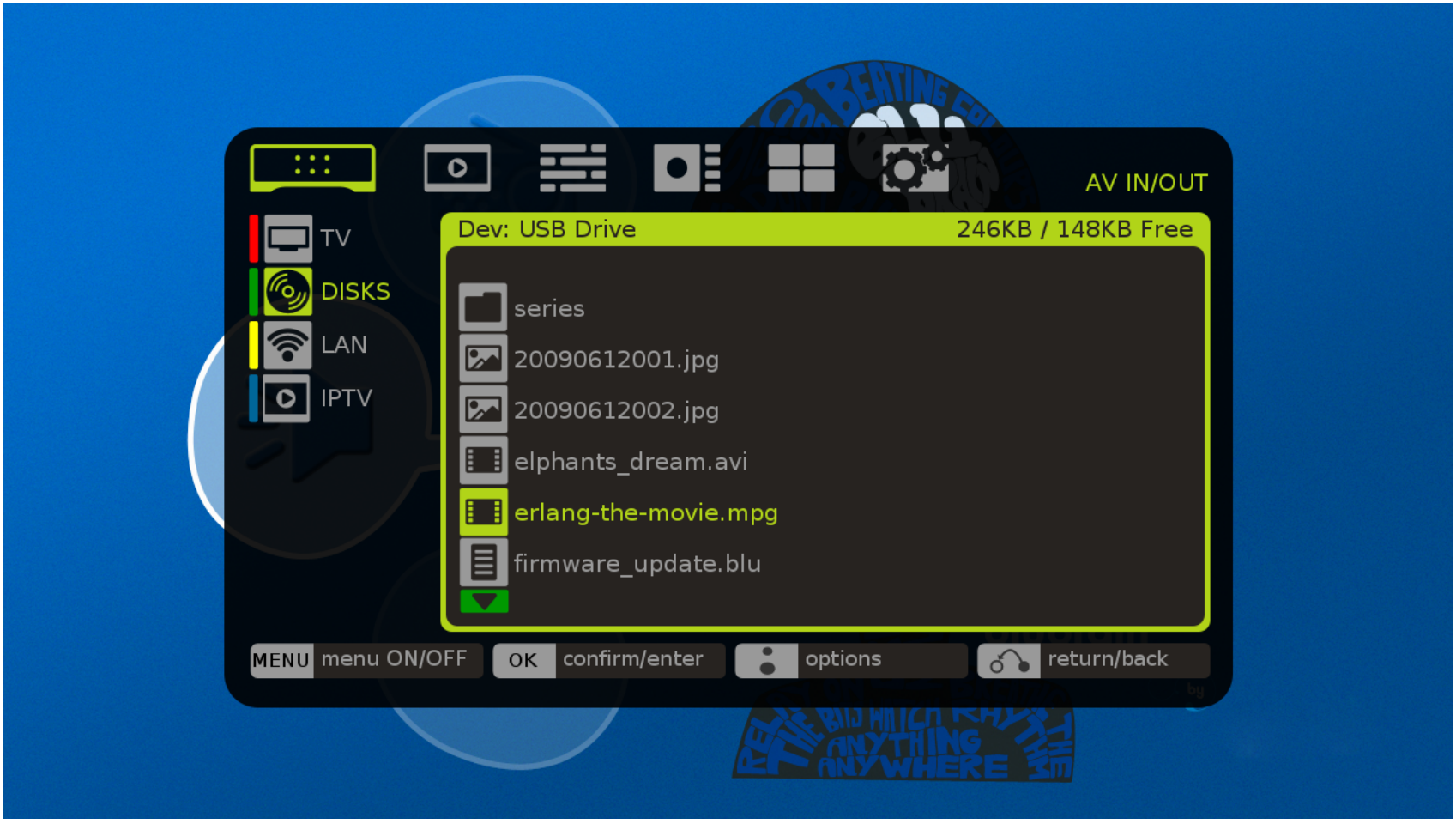
Use Webkit in our GUI

WebKit

- Small storage and memory footprint
- Powerful javascript engine
- CSS 3
- GTK backend that sits on top of DirectFB
- There are some embedded devices already using it

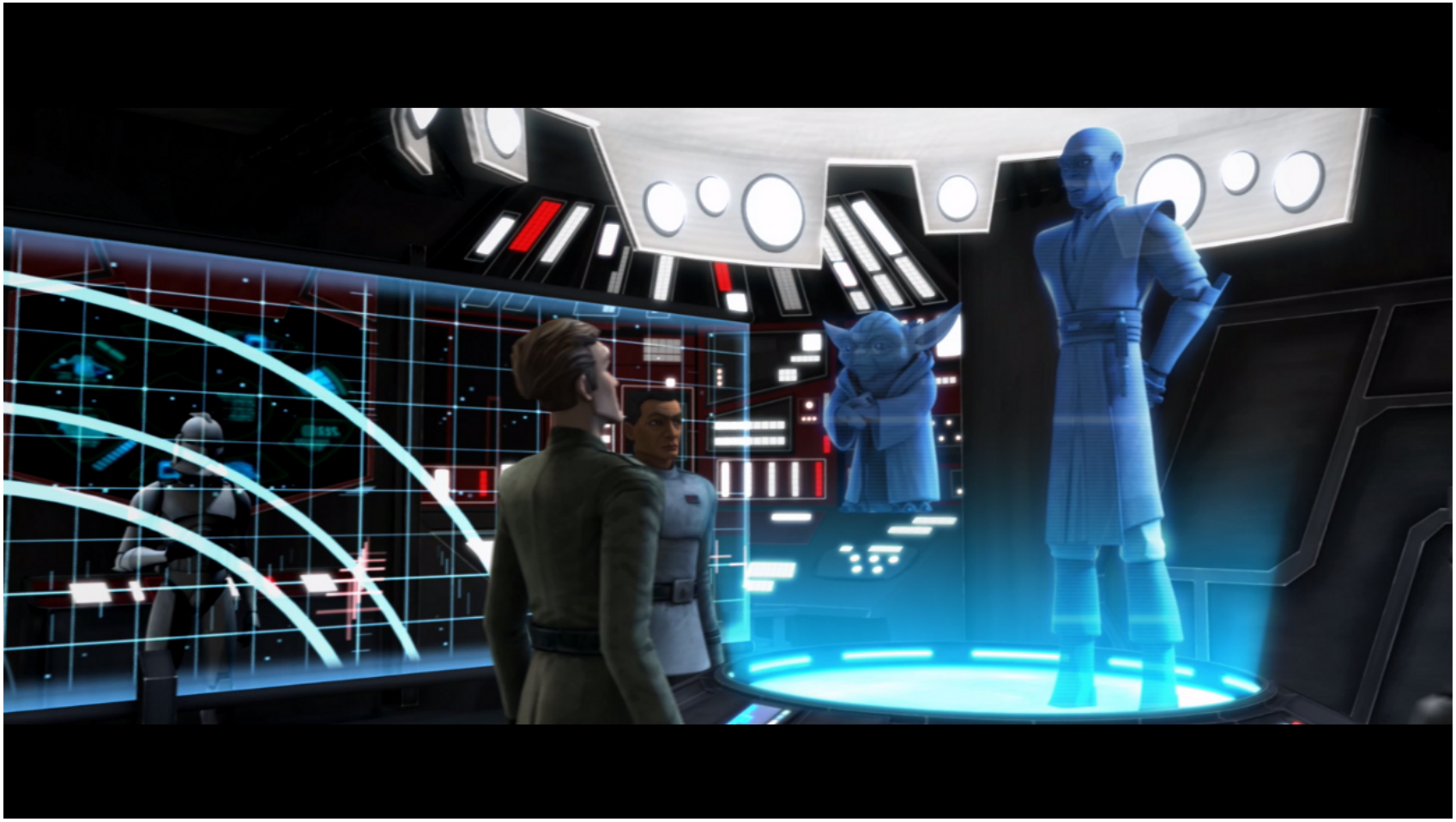
Advanced JavaScript GUI

Based on Focus and Key Press Events



Advanced JavaScript GUI

Alphachannel Blending With Video Layers



Advanced JavaScript GUI

Alphachannel Blending With Video Layers

Advanced Javascript GUI

Talking to C drivers from JavaScript?

But, how do you ...

- Make that fancy list of files?
- Configure the network??
- Play a video in the background???

We must extend the javascript API

- Should we patch WebKit's JS engine?
- Sounds a bit scary ...

Advanced JavaScript GUI

Code Snippet

```
function files_click(oEvent) {
    var target = $(oEvent.target);
    var path =  target.attr('video_path')

    switch(target.attr('logo_id')) {
    case 'video':
        loading_start();
        player.play("file://" + path,
                    function(oError) {
                        loading_stop();
        [...]
```

Connecting the GUI to the rest of the architecture

Connect the GUI with the player, file system, etc.?

- From Erlang, we can control all the functionality, including the C drivers.
- The Erlang part is the General Manager, the interface should contact it, not the drivers

No!! We must connect the GUI with Erlang

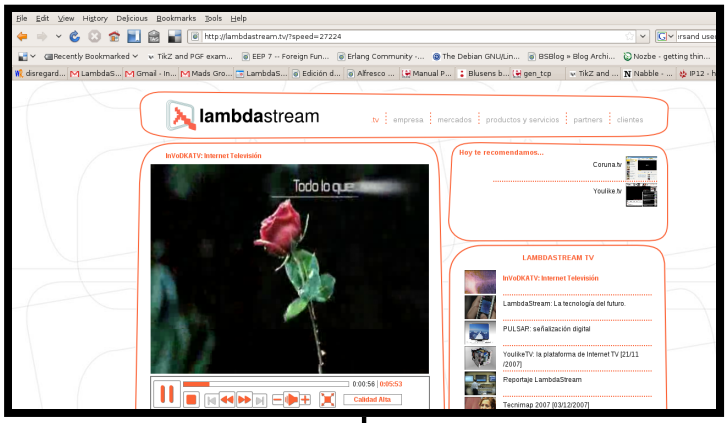
How do you connect a browser to something?

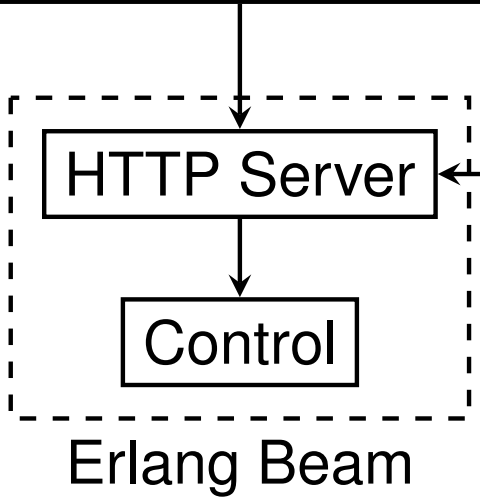
Connecting the GUI to the rest of the architecture

HTTP Server

Native GUI (localhost)

External Browser





HTTP Protocol

- The server interprets POST requests as commands
- Interchange of JSON objects
- Use comet to implement “asynchronous” communication
- Notify results and events through the comet connection

You can use that protocol in JavaScript using XmlRpcRequest

HTTP Protocol
Examples

Connecting to the server

```
POST http:/localhost/1.0/connect

{"id": "7045697"}
```

No response ...

HTTP Protocol
Examples

Starting a video playback

```
POST http:/localhost/1.0/rpc

{"id": "7045697", "cmd": "play",
 "arg": {"url": "file:///erlang-the-movie.avi"}}
```

Server Response

```
{"code":"ok,"ref":"92384"}
```

HTTP Protocol
Examples

Connecting to the server

```
POST http:/localhost/1.0/connect

{"id": "7045697"}
```

No response ... til now!

Server Response

```
{"ref":"92384","result":"ok"}
```

Conclusion

- Write C drivers to control small tasks related to the hardware and the OS
- Provide a platform for GUI developers
- Tie them to an Erlang program controlling all the stuff

Developing a Set Top Box Middleware in Erlang

Controlling Miscellaneous Stuff in an Embedded System

Samuel Rivas

LambdaStream S.L. samuel.rivas@lambdastream.com

London, 2009