HirondML PROJECT
(Supervisor : Emmanuel Chailloux)

Verlaguet Julien

15 septembre 2004

Contents

1 Presentation

1.1 Abstract

In this report we present an Objective Caml librairy implementing migrating fair threads. The aim of HirondML project is to define a semantic for distributed and concurrent programming which would be efficient and easy to understand.

The current implementation is experimental, many weaknesses will need to be discussed in a further work.

2 The librairy

2.1 The fair threads

It is important to note that HirondML is based on Fair Threads, a reactive like cooperative thread library.

2.2 The restrictions

The migrate librairy won't work if you don't respect the following rules

1. All the computers have to be Linux 32 bit platforms
2. They all have to execute the same binarie file
3. All the librairies used in the program have to be static
4. The program has to be compiled with ocamlopt (and not ocamlc)
5. All the I/O operations have to use the replacement functions from the Migrate library (don't use Unix, neither Pervasives).
6. Please note that number of I/O replacement functions implemented is incredibly high (= 2) (cf Migrate.read_line and read).

2.3 The semantic

So, what is all that about, to make short let's make a simple example

```
let chat home=
  let s=Migrate.read_line () in
  Migrate.migrate other_comp
  output_string s ;
  flush stdout ;
  chat home
```

Let's have a look at this program. Basically to understand all what you need to know is that "Migrate.migrate computer_nbr" means : I want the current continuation to be executed on the computer computer_nbr.

Of course, the first question that comes in your mind is : but how are all the variables going to be to be linked ? Some of them have to be copied ! !

The answer is pretty simple, if u want to know if a variable is going to be copied, just question yourself : does this variable exist on the targeted computer ? If not, then you know you are working on a copy.

This implies many precautions that the programmer should take when using the Migrate library. You have to be aware that if you are declaring a variable on one computer that you wish to use on another computer it will cost you time. To be more precise, the variable is going to be marshallized and sent on the network. Here is an example to illustrate that problem :

What you shouldn't do :

```
let test home=
  let tab=Array.make 1000 1 in
  Migrate.migrate other_comp
  output_int s.(0);
  flush stdout
;;
```

What you should do :

```
let test home=
  let tab=Array.make 1000 1 in
  let x=s.(0) in
  Migrate.migrate other_comp
  output_int x;
  flush stdout
;;
```

In the first case the entire array (tab) is marshallized, when in the second case the only variables sent on the network is x.

2.4 Capturing global variables

Assume now that all the local variables from the current continuation are copied during a migration. The global variables exist on all of the connected computers, they won't be serialized when a migration takes place. If a global variable of a particular computer is required, the simplest way to capture it (perhaps to work on another computer with it), is to create a local reference on it.

Here is a little example to illustrate the global variable capturing mechanism (note that Migrate.computer is a reference to the computer where the execution takes place) :

```
let global_tab=Array.make 10 !Migrate.computer;;
let test home=
  let tab=ref global_tab in
  we are capturing the global_tab of this computer
  Migrate.migrate other_comp;
  tab is now a reference to a copy of global_tab
  output_bool (!(tab).(0) =!Migrate.computer);
  this prints false
  flush stdout
;;
```

3 All you need to know

Here is all what u need to know to make your program work.

Write a text file with the addresses of all the computers you wish to connect to your network. Example :

127.0.0.1 :12345
168.92.0.1 :13423
...

Assume now that the computer 0 has the address 127.0.0.1 (port 12345), the computer 1 the address 168.92.0.1 ...

now to launch a thread on a particular computer all you need to type is :

```
Migrate.create (fun home -> ...) comp_nbr;
```

where ... is the code of your thread and comp_nbr is the number of the computer where you wish to launch it. Be aware that comp_nbr is passed as an argument to the thread function.

Here is a few function that might be usefull (for more details see doc/index.html) : cooperate, link, create_scheduler, computer (the reference to the computer where u are), nbr_comp (a reference to the number of computer launched)

And finally, here is how to launch the programs. On the first computer (!computer = 0), type :

./your_program -mynum 0 -list file_with_the_computer_addresses

On the second one, just change the -mynum to 1, ..., til all the computer are launched. Be aware that the order is important!!